ABDK CONSULTING

SMART CONTRACT AUDIT

Algebra

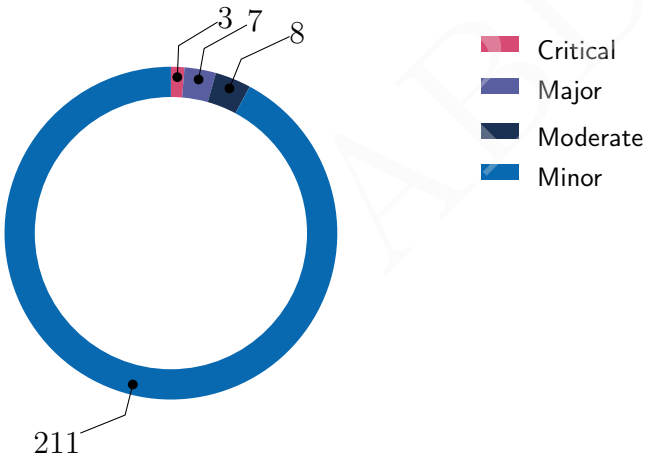
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
29th July 2022

We've been asked to review 30 files in a Github repository. We found 3 critical, 7 major, and a few less important issues. All critical issues have been fixed. We also performed an additional review for all fixes including minor issues.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Procedural	Fixed
CVF-2	Minor	Bad datatype	Info
CVF-3	Minor	Bad datatype	Info
CVF-4	Minor	Bad datatype	Info
CVF-5	Minor	Procedural	Fixed
CVF-6	Minor	Documentation	Fixed
CVF-7	Minor	Bad datatype	Info
CVF-8	Minor	Bad datatype	Fixed
CVF-9	Minor	Bad datatype	Info
CVF-10	Minor	Bad datatype	Info
CVF-11	Minor	Bad datatype	Info
CVF-12	Minor	Suboptimal	Fixed
CVF-13	Minor	Suboptimal	Fixed
CVF-14	Minor	Bad datatype	Fixed
CVF-15	Minor	Readability	Fixed
CVF-16	Minor	Overflow/Underflow	Fixed
CVF-17	Minor	Bad datatype	Info
CVF-18	Minor	Bad datatype	Info
CVF-19	Minor	Bad datatype	Info
CVF-20	Minor	Bad datatype	Info
CVF-21	Minor	Unclear behavior	Fixed
CVF-22	Minor	Procedural	Info
CVF-23	Minor	Suboptimal	Fixed
CVF-24	Minor	Documentation	Info
CVF-25	Minor	Documentation	Info
CVF-26	Minor	Readability	Info
CVF-27	Minor	Readability	Info

ID	Severity	Category	Status
CVF-28	Minor	Documentation	Info
CVF-29	Minor	Unclear behavior	Info
CVF-30	Minor	Suboptimal	Info
CVF-31	Minor	Suboptimal	Fixed
CVF-32	Minor	Suboptimal	Info
CVF-33	Minor	Readability	Fixed
CVF-34	Minor	Readability	Info
CVF-35	Minor	Unclear behavior	Info
CVF-36	Minor	Procedural	Fixed
CVF-37	Major	Overflow/Underflow	Info
CVF-38	Minor	Suboptimal	Info
CVF-39	Minor	Flaw	Info
CVF-40	Major	Overflow/Underflow	Info
CVF-41	Minor	Suboptimal	Fixed
CVF-42	Major	Overflow/Underflow	Fixed
CVF-43	Minor	Suboptimal	Info
CVF-44	Minor	Bad datatype	Info
CVF-45	Minor	Documentation	Fixed
CVF-46	Minor	Suboptimal	Info
CVF-47	Minor	Bad datatype	Fixed
CVF-48	Minor	Suboptimal	Fixed
CVF-49	Critical	Flaw	Fixed
CVF-50	Minor	Suboptimal	Info
CVF-51	Minor	Documentation	Info
CVF-52	Minor	Suboptimal	Fixed
CVF-53	Minor	Documentation	Fixed
CVF-54	Minor	Bad naming	Fixed
CVF-55	Minor	Suboptimal	Info
CVF-56	Minor	Suboptimal	Info
CVF-57	Minor	Suboptimal	Fixed

ID	Severity	Category	Status
CVF-58	Minor	Suboptimal	Fixed
CVF-59	Major	Overflow/Underflow	Fixed
CVF-60	Major	Suboptimal	Fixed
CVF-61	Minor	Suboptimal	Info
CVF-62	Minor	Bad datatype	Fixed
CVF-63	Minor	Suboptimal	Fixed
CVF-64	Minor	Bad datatype	Fixed
CVF-65	Minor	Bad datatype	Info
CVF-66	Minor	Documentation	Info
CVF-67	Minor	Bad datatype	Info
CVF-68	Minor	Suboptimal	Fixed
CVF-69	Minor	Unclear behavior	Fixed
CVF-70	Minor	Suboptimal	Fixed
CVF-71	Major	Flaw	Fixed
CVF-72	Minor	Suboptimal	Info
CVF-73	Minor	Documentation	Fixed
CVF-74	Minor	Bad datatype	Info
CVF-75	Minor	Suboptimal	Info
CVF-76	Minor	Procedural	Fixed
CVF-77	Minor	Procedural	Info
CVF-78	Minor	Documentation	Info
CVF-79	Minor	Overflow/Underflow	Info
CVF-80	Minor	Suboptimal	Fixed
CVF-81	Minor	Suboptimal	Info
CVF-82	Minor	Suboptimal	Fixed
CVF-83	Minor	Documentation	Fixed
CVF-84	Minor	Documentation	Fixed
CVF-85	Minor	Bad naming	Fixed
CVF-86	Moderate	Overflow/Underflow	Info
CVF-87	Moderate	Unclear behavior	Fixed

ID	Severity	Category	Status
CVF-88	Minor	Suboptimal	Info
CVF-89	Minor	Suboptimal	Fixed
CVF-90	Minor	Suboptimal	Fixed
CVF-91	Minor	Readability	Info
CVF-92	Minor	Suboptimal	Info
CVF-93	Minor	Suboptimal	Fixed
CVF-94	Minor	Suboptimal	Fixed
CVF-95	Minor	Suboptimal	Info
CVF-96	Minor	Procedural	Fixed
CVF-97	Minor	Bad datatype	Fixed
CVF-98	Minor	Procedural	Info
CVF-99	Minor	Documentation	Fixed
CVF-100	Minor	Documentation	Fixed
CVF-101	Moderate	Overflow/Underflow	Fixed
CVF-102	Minor	Suboptimal	Info
CVF-103	Minor	Bad naming	Fixed
CVF-104	Minor	Documentation	Fixed
CVF-105	Minor	Bad naming	Fixed
CVF-106	Critical	Procedural	Fixed
CVF-107	Moderate	Suboptimal	Fixed
CVF-108	Minor	Bad datatype	Fixed
CVF-109	Moderate	Overflow/Underflow	Fixed
CVF-110	Minor	Procedural	Fixed
CVF-111	Minor	Unclear behavior	Fixed
CVF-112	Minor	Readability	Fixed
CVF-113	Minor	Documentation	Info
CVF-114	Minor	Documentation	Fixed
CVF-115	Moderate	Overflow/Underflow	Fixed
CVF-116	Minor	Unclear behavior	Info
CVF-117	Minor	Documentation	Fixed

ID	Severity	Category	Status
CVF-118	Minor	Unclear behavior	Info
CVF-119	Minor	Documentation	Info
CVF-120	Moderate	Unclear behavior	Info
CVF-121	Minor	Readability	Fixed
CVF-122	Minor	Procedural	Fixed
CVF-123	Minor	Bad datatype	Fixed
CVF-124	Minor	Suboptimal	Info
CVF-125	Minor	Readability	Fixed
CVF-126	Minor	Suboptimal	Fixed
CVF-127	Minor	Unclear behavior	Info
CVF-128	Minor	Suboptimal	Fixed
CVF-129	Minor	Suboptimal	Fixed
CVF-130	Critical	Flaw	Fixed
CVF-131	Minor	Documentation	Fixed
CVF-132	Minor	Documentation	Fixed
CVF-133	Minor	Documentation	Fixed
CVF-134	Minor	Suboptimal	Fixed
CVF-135	Minor	Readability	Info
CVF-136	Minor	Suboptimal	Info
CVF-137	Minor	Readability	Info
CVF-138	Minor	Suboptimal	Info
CVF-139	Minor	Documentation	Fixed
CVF-140	Minor	Suboptimal	Fixed
CVF-141	Minor	Documentation	Fixed
CVF-142	Minor	Suboptimal	Fixed
CVF-143	Minor	Suboptimal	Fixed
CVF-144	Minor	Suboptimal	Fixed
CVF-145	Minor	Documentation	Fixed
CVF-146	Minor	Procedural	Info
CVF-147	Minor	Documentation	Fixed

ID	Severity	Category	Status
CVF-148	Minor	Documentation	Info
CVF-149	Minor	Suboptimal	Fixed
CVF-150	Moderate	Overflow/Underflow	Fixed
CVF-151	Minor	Suboptimal	Info
CVF-152	Minor	Suboptimal	Fixed
CVF-153	Minor	Suboptimal	Fixed
CVF-154	Minor	Documentation	Fixed
CVF-155	Minor	Documentation	Fixed
CVF-156	Minor	Documentation	Fixed
CVF-157	Minor	Documentation	Fixed
CVF-158	Minor	Bad naming	Info
CVF-159	Minor	Bad naming	Info
CVF-160	Minor	Procedural	Fixed
CVF-161	Minor	Documentation	Fixed
CVF-162	Minor	Documentation	Fixed
CVF-163	Minor	Documentation	Fixed
CVF-164	Minor	Bad datatype	Info
CVF-165	Minor	Documentation	Fixed
CVF-166	Minor	Bad datatype	Info
CVF-167	Minor	Bad datatype	Info
CVF-168	Minor	Bad datatype	Info
CVF-169	Minor	Unclear behavior	Fixed
CVF-170	Minor	Suboptimal	Info
CVF-171	Minor	Bad naming	Fixed
CVF-172	Minor	Documentation	Fixed
CVF-173	Minor	Suboptimal	Fixed
CVF-174	Minor	Procedural	Fixed
CVF-175	Minor	Suboptimal	Fixed
CVF-176	Minor	Documentation	Info
CVF-177	Minor	Suboptimal	Info

ID	Severity	Category	Status
CVF-178	Major	Documentation	Fixed
CVF-179	Minor	Documentation	Fixed
CVF-180	Minor	Documentation	Fixed
CVF-181	Minor	Documentation	Fixed
CVF-182	Minor	Bad naming	Fixed
CVF-183	Minor	Unclear behavior	Info
CVF-184	Minor	Bad datatype	Info
CVF-185	Minor	Documentation	Fixed
CVF-186	Minor	Documentation	Info
CVF-187	Minor	Documentation	Fixed
CVF-188	Minor	Documentation	Info
CVF-189	Minor	Documentation	Fixed
CVF-190	Minor	Documentation	Fixed
CVF-191	Minor	Suboptimal	Info
CVF-192	Minor	Documentation	Fixed
CVF-193	Minor	Procedural	Fixed
CVF-194	Minor	Documentation	Info
CVF-195	Minor	Documentation	Fixed
CVF-196	Minor	Bad naming	Fixed
CVF-197	Minor	Documentation	Fixed
CVF-198	Minor	Procedural	Fixed
CVF-199	Minor	Bad naming	Fixed
CVF-200	Minor	Documentation	Fixed
CVF-201	Minor	Documentation	Fixed
CVF-202	Minor	Bad naming	Fixed
CVF-203	Minor	Bad datatype	Info
CVF-204	Minor	Suboptimal	Fixed
CVF-205	Minor	Bad naming	Fixed
CVF-206	Minor	Bad datatype	Info
CVF-207	Minor	Bad datatype	Info

ID	Severity	Category	Status
CVF-208	Minor	Bad datatype	Info
CVF-209	Minor	Bad datatype	Info
CVF-210	Minor	Bad datatype	Info
CVF-211	Minor	Bad datatype	Info
CVF-212	Minor	Documentation	Fixed
CVF-213	Minor	Bad naming	Fixed
CVF-214	Minor	Suboptimal	Fixed
CVF-215	Minor	Bad naming	Fixed
CVF-216	Minor	Bad datatype	Info
CVF-217	Minor	Bad datatype	Info
CVF-218	Minor	Bad datatype	Info
CVF-219	Minor	Bad datatype	Info
CVF-220	Minor	Bad datatype	Info
CVF-221	Minor	Bad datatype	Info
CVF-222	Minor	Documentation	Fixed
CVF-223	Minor	Readability	Fixed
CVF-224	Minor	Documentation	Fixed
CVF-225	Minor	Bad naming	Fixed
CVF-226	Minor	Bad datatype	Info
CVF-227	Minor	Bad naming	Fixed
CVF-228	Minor	Readability	Info
CVF-229	Minor	Bad naming	Fixed

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	July 29, 2022	D. Khovratovich	Initial Draft
0.2	July 29, 2022	D. Khovratovich	Minor revision
1.0	July 29, 2022	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.
We have reviewed the contracts at repository:

- interfaces/callback/IAlgebraFlashCallback.sol
- interfaces/callback/IAlgebraMintCallback.sol
- interfaces/callback/IAlgebraSwapCallback.sol
- interfaces/pool/IAlgebraPoolActions.sol
- interfaces/pool/IAlgebraPoolDerivedState.sol
- interfaces/pool/IAlgebraPoolEvents.sol
- interfaces/pool/IAlgebraPoolImmutables.sol
- interfaces/pool/IAlgebraPoolPermissionedActions.sol
- interfaces/pool/IAlgebraPoolState.sol
- interfaces/IAlgebraFactory.sol
- interfaces/IAlgebraPool.sol
- interfaces/IAlgebraPoolDeployer.sol
- interfaces/IAlgebraVirtualPool.sol
- interfaces/IDataStorageOperator.sol
- base/PoolImmutables.sol
- base/PoolState.sol
- libraries/AdaptiveFee.sol
- libraries/Constants.sol
- libraries/DataStorage.sol
- libraries/FullMath.sol
- libraries/PIFee.sol
- libraries/PriceMovementMath.sol
- libraries/Sqrt.sol
- libraries/TickManager.sol

- libraries/TickTable.sol
- libraries/TokenDeltaMath.sol
- AlgebraFactory.sol
- AlgebraPool.sol
- AlgebraPoolDeployer.sol
- DataStorageOperator.sol

The fixes were provided in a new commit.

2.1 About ABDK

ABDK Consulting, established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function. The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places and that their visibility scopes and access levels are relevant. At this phase we understand overall system architecture and how different parts of the code are related to each other.

- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and is done properly. At this phase we understand user roles and permissions, as well as what assets the system ought to protect.

- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** PoolImmutables.sol

Recommendation Specifying a particular version makes it harder to migrate to newer versions. Consider specifying as "^0.7.0". Also relevant for the next files:DataStorageOperator.sol, AlgebraPoolDeployer.sol, AlgebraPool.sol, AlgebraFactory.sol, PoolState.sol, PIFee.sol, DataStorage.sol, Constants.sol, IAlgebraVirtualPool.sol.
Client Comment Version for IAlgebraVirtualPool.sol changed. Other listed files are not supposed to be compiled in another version.

Listing 1:

```
2 pragma solidity =0.7.6;
```

3.2 CVF-2

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** PoolImmutables.sol

Recommendation The type of this variable should be "IDataStorageOperator".

Listing 2:

```
9 address public immutable override dataStorageOperator;
```

3.3 CVF-3

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** PoolImmutables.sol

Recommendation The type of this variable should be "IAlgebraFactory".

Listing 3:

```
12 address public immutable override factory;
```

3.4 CVF-4

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** PoolImmutables.sol

Recommendation The type of these variables should be "IERC20".

Listing 4:

```
14 address public immutable override token0;

16 address public immutable override token1;
```

3.5 CVF-5

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** PoolImmutables.sol

Recommendation Constants are usually named IN_UPPER_CASE.
Client Comment Internally we now use upper case. External contract should not rely on it being a constant.

Listing 5:

```
19 uint8 public constant override tickSpacing = 60;

22 uint128 public constant override maxLiquidityPerTick =
    ↪ 11505743598341114571880798222544994;
```

3.6 CVF-6

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** PoolImmutables.sol

Description It is unclear where this number came from.
Recommendation Consider explaining.

Listing 6:

```
22 uint128 public constant override maxLiquidityPerTick =
    ↪ 11505743598341114571880798222544994;
```

3.7 CVF-7

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** PoolImmutables.sol

Recommendation The type of the "deployer" argument should be "IAlgebraPoolDeployer".

Listing 7:

```
24 constructor(address deployer) {
```

3.8 CVF-8

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** DataStorageOperator.sol

Recommendation The value "65535" should be a named constant.

Listing 8:

```
17 DataStorage.Timepoint[65535] public override timepoints;

67   if (timepoints[addmod(index, 1, 65535)].initialized) {
       oldestIndex = uint16(addmod(index, 1, 65535));
```

3.9 CVF-9

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** DataStorageOperator.sol

Recommendation The type of this variable should be "IAlgebraPool".

Listing 9:

```
20 address private immutable pool;
```

3.10 CVF-10

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** DataStorageOperator.sol

Recommendation The type of this variable should be "IAlgebraFactory".

Listing 10:

```
21 address private immutable factory;
```

3.11 CVF-11

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** DataStorageOperator.sol

Recommendation The argument type should be "IAlgebraPool".

Listing 11:

```
33 constructor(address _pool) {
```

3.12 CVF-12

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** DataStorageOperator.sol

Description This function should emit some event.

Listing 12:

```
42 function changeFeeConfiguration(AdaptiveFee.Configuration
   ↪ calldata _feeConfig) external override {
```

3.13 CVF-13

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** DataStorageOperator.sol

Description The expression "addmod(index, 1, 65535)" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 13:

```
67 if (timepoints[addmod(index, 1, 65535)].initialized) {
```

3.14 CVF-14

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** DataStorageOperator.sol

Recommendation The value "100000 « 64" should be a named constant.

Listing 14:

```
129 if (volumeShifted >= 100000 << 64) return 100000 << 64;
```

3.15 CVF-15

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** DataStorageOperator.sol

Recommendation For readability and simplicity should be: else return uint128(volumeShifted);

Listing 15:

```
130 volumePerLiquidity = uint128(volumeShifted);
```

3.16 CVF-16

- **Severity** Minor
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** DataStorageOperator.sol

Description Overflow is possible when converting to "uint16".
Recommendation Consider either using same conversion or changing the return type of the "AdaptiveFee.getFee" function to "uint16".
Client Comment Return type of AdaptiveFee.getFee changed to uint16. Added checks, comments and fuzzy tests.

Listing 16:

```
145 return uint16(AdaptiveFee.getFee(volatilityAverage / 15,
    ↪ volumePerLiqAverage, feeConfig));
```

3.17 CVF-17

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** AlgebraPoolDeployer.sol

Recommendation The type of this field should be "IDataStorageOperator".

Listing 17:

```
9 address dataStorage;
```

3.18 CVF-18

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** AlgebraPoolDeployer.sol

Recommendation The type of this field should be "IAlgebraFactory".

Listing 18:

```
10 address factory;
```

3.19 CVF-19

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** AlgebraPoolDeployer.sol

Recommendation The type of these fields should be "IERC20".

Listing 19:

```
11 address token0;
   address token1;
```

3.20 CVF-20

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** AlgebraPoolDeployer.sol

Recommendation The type of this variable should be "IAlgebraFactory".

Listing 20:

```
18 address private factory;
```

3.21 CVF-21

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** AlgebraPoolDeployer.sol

Recommendation As zero factory address has a special meaning, consider adding an explicit "require" statement to check that "_factory" is not zero.

Listing 21:

```
37 require(factory == address(0));
```

3.22 CVF-22

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** AlgebraPool.sol

Description We didn't review these files.
Client Comment These are audited open source files

Listing 22:

```
16 import './libraries/LowGasSafeMath.sol';
   import './libraries/SafeCast.sol';

21 import './libraries/TransferHelper.sol';
   import './libraries/TickMath.sol';
   import './libraries/LiquidityMath.sol';

28 import './interfaces/IERC20Minimal.sol';
```

3.23 CVF-23

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Description The field name is inconsistent with the comment.
Recommendation Consider renaming the field to "lastAddTimestamp".

Listing 23:

```
43 uint32 lastModificationTimestamp; // Timestamp of last adding of
   ↪ liquidity
```

3.24 CVF-24

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** AlgebraPool.sol

Description The number format of these field is unclear.
Recommendation Consider documenting.

Listing 24:

```
96 int56 tickCumulative;
   uint160 outerSecondPerLiquidity;
```

3.25 CVF-25

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** AlgebraPool.sol

Description The number format of these returned values is unclear.
Recommendation Consider documenting.

Listing 25:

```
107 int56 innerTickCumulative,
    uint160 innerSecondsSpentPerLiquidity,
```

3.26 CVF-26

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** AlgebraPool.sol

Recommendation Should be “else if“ for readability.

Listing 26:

```
147 if (currentTick < topTick) {
```

3.27 CVF-27

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** AlgebraPool.sol

Recommendation Should be “else return” for readability.

Listing 27:

```
163 return (
```

3.28 CVF-28

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** AlgebraPool.sol

Description The number format of the argument is unclear.
Recommendation Consider documenting.

Listing 28:

```
193 function initialize(uint160 initialPrice) external override {

384   uint160 currentPrice
```

3.29 CVF-29

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** AlgebraPool.sol

Description There is no range check for the argument.
Recommendation Consider adding an appropriate check.
Client Comment getTickAtSqrtRatio checks validity of initialPrice inside. Relevant comment added.

Listing 29:

```
193 function initialize(uint160 initialPrice) external override {
```

3.30 CVF-30

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Description While the "muldiv" function offers great performance in general case, for specific cases more efficient solutions do exist. For example, when the denominator is a known power of two, it would be more efficient to. implement a "mulshift" function that calculates a 512-bit product of two numbers and then shifts it right to produce a 256-bit value.
Recommendation Consider implementing and using such a function.
Client Comment Considered for the future.

Listing 30:

```
250 fees0 = uint128(FullMath.mulDiv(innerFeeGrowth0Token -
    ↪ _innerFeeGrowth0Token, currentLiquidity, Constants.Q128));

255 fees1 = uint128(FullMath.mulDiv(innerFeeGrowth1Token -
    ↪ _innerFeeGrowth1Token, currentLiquidity, Constants.Q128));
```

3.31 CVF-31

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Recommendation This could be simplified as: if (fees0 | fees1 != 0) {

Listing 31:

```
259 if (!(fees0 == 0 && fees1 == 0)) {
```

3.32 CVF-32

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Description The expression "liquidityDelta != 0" is calculated twice.
Recommendation Consider calculating once and reusing.
Client Comment It looks like the change will use more gas than it saves.

Listing 32:

```
292 if (liquidityDelta != 0) {
```

```
347 if (liquidityDelta != 0) {
```

3.33 CVF-33

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** AlgebraPool.sol

Recommendation Consider providing a named argument before the constant like 'arg: true' for readability.
Client Comment Comment added.

Listing 33:

```
312 false
```

```
329 true
```

3.34 CVF-34

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** AlgebraPool.sol

Recommendation This could be calculated in plain Solidity. No need for assembly.
Client Comment Noted.

Listing 34:

```
422 assembly {
      key := or(shl(24, or(shl(24, owner), and(bottomTick, 0xFFFFFF)
        ↪ )), and(topTick, 0xFFFFFF))
    }
```

3.35 CVF-35

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** AlgebraPool.sol

Description Applying masks on "bottomTick" and "topTick" is redundant as these values are already 24-bits wide. Is there an attack scenario where these values could be manipulated to not fit into 24 bits?
Client Comment We wouldn't want to rely on parameter cleanup here.

Listing 35:

```
423 key := or(shl(24, or(shl(24, owner), and(bottomTick, 0xFFFFFF)))
      ↪ , and(topTick, 0xFFFFFF))
```

3.36 CVF-36

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** AlgebraPool.sol

Description Unlike names of other arguments, this name has the underscore ("_") prefix.
Recommendation Consider removing the prefix for consistency.

Listing 36:

```
434 uint128 _liquidity,
```

3.37 CVF-37

- **Severity** Major
- **Status** Info
- **Category** Overflow/Underflow
- **Source** AlgebraPool.sol

Description Underflow is possible here.
Recommendation Consider using safe conversions.
Client Comment amount0Int and amount1Int are calculated from TokenDeltaMath.getToken0Delta and TokenDeltaMath.getToken1Delta which return uint256 casted into int256 afterwards. But the uint256 returned is not greater that 224-bit number since it is bounded by liquidity (uint128) shifted on 96 to the left

Listing 37:

```
457 amount0 = uint256(amount0Int);
    amount1 = uint256(amount1Int);
```

3.38 CVF-38

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Description For a very common case when the received amounts exactly equal the desired amounts, most of the code below is redundant.
Recommendation Consider implementing a shortcut to handle such case efficiently. Also, allowing a callback to actually provide token amounts that differ from the desired amounts, makes the whole logic overcomplicated. Consider removing support for such exotic scenarios.
Client Comment Such feature has been added to support tokens with built-in fees.

Listing 38:

```
471 if (receivedAmount0 < amount0) {

474 if (receivedAmount1 < amount1) {
```

3.39 CVF-39

- **Severity** Minor
- **Status** Info
- **Category** Flaw
- **Source** AlgebraPool.sol

Description The position is not recalculated before obtaining the earned fees, thus the fee amounts could be outdated.
Recommendation Consider recalculating the position before obtaining the fee amounts.

Listing 39:

```
509 (uint128 positionFees0, uint128 positionFees1) = (position.fees0
    ↪ , position.fees1);
```

3.40 CVF-40

- **Severity** Major
- **Status** Info
- **Category** Overflow/Underflow
- **Source** AlgebraPool.sol

Description Underflow is possible here.
Recommendation Consider using safe conversion.
Client Comment amount0Int and amount1Int are calculated from TokenDeltaMath.getToken0Delta and TokenDeltaMath.getToken1Delta which in this case always return negative value since liquidityDelta passed in this function is always negative in burn() function

Listing 40:

```
540 amount0 = uint256(-amount0Int);
    amount1 = uint256(-amount1Int);
```

3.41 CVF-41

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Recommendation This could be simplified as; if (amount0 | amount1 != 0) {

Listing 41:

```
543 if (amount0 > 0 || amount1 > 0) {
```

3.42 CVF-42

- **Severity** Major
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** AlgebraPool.sol

Description Overflow is possible here.
Recommendation Consider using SafeMath and safe conversion.

Listing 42:

```
544 (position.fees0, position.fees1) = (position.fees0 + uint128(
    ↪ amount0), position.fees1 + uint128(amount1));
```

3.43 CVF-43

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Description This event is emitted even if the fee doesn't change.
Recommendation Consider emitting only when the fee actually changed.

Listing 43:

```
559 emit ChangeFee(newFee);
```

3.44 CVF-44

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** AlgebraPool.sol

Recommendation The type of the "token" argument should be "IERĆ20".

Listing 44:

```
562 function _payCommunityFee(address token, uint256 amount) private
    ↪ {
```

3.45 CVF-45

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** AlgebraPool.sol

Description The semantics of the returned value is unclear.
Recommendation Consider giving a descriptive name to the returned value and/or adding a documentation comment.

Listing 45:

```
573 ) private returns (uint16) {
```

3.46 CVF-46

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Description The contract is locked inside the "_calculateSwap" function call and unlocked outside. This is error-prone and makes the contract harder to read.
Recommendation Consider using the "lock" modifier instead.
Client Comment Added comments and mentioned in the name of the function its blocking behavior.

Listing 46:

```
619 (amount0, amount1, currentPrice, currentTick, currentLiquidity,
    ↪ feeAmount) = _calculateSwap(zeroForOne, amountRequired,
    ↪ limitSqrtPrice);

642 globalState.unlocked = true;
```

3.47 CVF-47

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** AlgebraPool.sol

Recommendation The denominator 100 should be a named constant.

Listing 47:

```
635 communityFee = zeroForOne ? (feeAmount * globalState.
    ↪ communityFeeToken0) / 100 : (feeAmount * globalState.
    ↪ communityFeeToken1) / 100;

685 communityFee = zeroForOne ? (feeAmount * globalState.
    ↪ communityFeeToken0) / 100 : (feeAmount * globalState.
    ↪ communityFeeToken1) / 100;

870     uint256 delta = (step.feeAmount * cache.communityFee) / 100;

998         fees0 = (paid0 * _communityFeeToken0) / 100;

1014        fees1 = (paid1 * _communityFeeToken1) / 100;
```

3.48 CVF-48

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Description The denominator for a community fee looks too low. It allows community fee of 2% or 3% but not 2.5%.
Recommendation Consider increasing the denominator.

Listing 48:

```
635 communityFee = zeroForOne ? (feeAmount * globalState.
    ↪ communityFeeToken0) / 100 : (feeAmount * globalState.
    ↪ communityFeeToken1) / 100;

685 communityFee = zeroForOne ? (feeAmount * globalState.
    ↪ communityFeeToken0) / 100 : (feeAmount * globalState.
    ↪ communityFeeToken1) / 100;

870     uint256 delta = (step.feeAmount * cache.communityFee) / 100;

998         fees0 = (paid0 * _communityFeeToken0) / 100;

1014        fees1 = (paid1 * _communityFeeToken1) / 100;
```

3.49 CVF-49

- **Severity** Critical
- **Status** Fixed
- **Category** Flaw
- **Source** AlgebraPool.sol

Description This code is executed without locking the contract thus reentrancy is possible that allows using the same tokens in several swaps, effectively double spending them. A callback just needs to perform another call to the "swapSupportingFeeOnInputTokens" function. Thus, both calls, inner and outer, will see the same change in token balance.
Recommendation Consider putting the callback calls under the lock.

Listing 49:

```
663 uint256 balance0Before = balanceToken0();
    _swapCallback(amountRequired, 0, 0, data);
    require((amountRequired = int256(balanceToken0().sub(
       ↪ balance0Before))) > 0, 'IIA');

667 uint256 balance1Before = balanceToken1();
    _swapCallback(0, amountRequired, 0, data);
    require((amountRequired = int256(balanceToken1().sub(
       ↪ balance1Before))) > 0, 'IIA');
```

3.50 CVF-50

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Description The contract is locked inside the "_calculateSwap" function call and unlocked outside. This is error-prone and makes the contract harder to read.
Recommendation Consider using the "lock" modifier instead.
Client Comment Added comments and mentioned in the name of the function its blocking behavior.

Listing 50:

```
672 (amount0, amount1, currentPrice, currentTick, currentLiquidity,
       ↪ feeAmount) = _calculateSwap(zeroForOne, amountRequired,
       ↪ limitSqrtPrice);

692 globalState.unlocked = true;
```

3.51 CVF-51

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** AlgebraPool.sol

Description The number format of these fields is unclear.
Recommendation Consider documenting.

Listing 51:

```
696 uint8 communityFee; // The community fee of the selling token
    uint160 startPrice; // Price at the beginning of the block
    uint16 startFee; // Fee at the beginning of the block
    uint128 liquidityStart; // The liquidity at the start of a swap
700 uint128 volumePerLiquidityInBlock;
    int56 tickCumulative; // The global tickCumulative at the moment
    uint160 secondsPerLiquidityCumulative; // The global
      ↪ secondPerLiquidity at the moment

706 uint256 totalFeeGrowth; // The initial totalFeeGrowth + the fee
      ↪ growth during a swap
    uint256 totalFeeGrowthB;

711 uint16 fee; // The current dynamic fee
```

3.52 CVF-52

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Description The same word "start" is used for both, start of the block and start of the swap. This makes the code harder to read.
Recommendation Consider using "blockStart" for block start and "swapStart" or just "start" for swap start.
Client Comment Removed.

Listing 52:

```
697 uint160 startPrice; // Price at the beginning of the block
    uint16 startFee; // Fee at the beginning of the block
    uint128 liquidityStart; // The liquidity at the start of a swap
```

3.53 CVF-53

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** AlgebraPool.sol

Description The number format of these fields is unclear.
Recommendation Consider documenting.
Client Comment Mentioned Q64.96 fortmat in comments.

Listing 53:

```
717 uint160 stepSqrtPrice; // The sqrt of the price at the start

720 uint160 nextTickPrice; // The sqrt of the price calculated from
    ↪ the _nextTick
```

3.54 CVF-54

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** AlgebraPool.sol

Recommendation The name "zeroToOne" would be more clear. Zero for one could be understood as "sell token zero for token one" or "buy token zero for token one".

Listing 54:

```
727 bool zeroForOne,
```

3.55 CVF-55

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Description This function lock the pool but never unlocks it. This is a bad and error prone practice.
Recommendation Consider locking and unlocking in the same function.
Client Comment Added comments and mentioned in the name of the function its blocking behavior.

Listing 55:

```
763 globalState.unlocked = false;
```

3.56 CVF-56

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Description It would be logical to use "TickMath.MIN_SQRT_RATIO" and "TickMath.MAX_SQRT_RATIO" values for the "limitSqrtPrice" argument as "no limit" indicator.
Recommendation Consider allowing these values.
Client Comment For these purposes, the closest values can be used: "TickMath.MIN_SQRT_RATIO + 1" and "TickMath.MAX_SQRT_RATIO - 1".

Listing 56:

```
774 require(limitSqrtPrice < currentPrice && limitSqrtPrice >
    ↪ TickMath.MIN_SQRT_RATIO, 'SPL');

778 require(limitSqrtPrice > currentPrice && limitSqrtPrice <
    ↪ TickMath.MAX_SQRT_RATIO, 'SPL');
```

3.57 CVF-57

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Recommendation As long as "blockStartPrice" is supposed to be updated at most once per block, it would be more logical to use block.number here rather than block.timestamp.

Listing 57:

```
789 startPriceUpdated = blockTimestamp;
```

3.58 CVF-58

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Description This expression is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 58:

```
836 (zeroForOne == (step.nextTickPrice < limitSqrtPrice)) // move
      ↪ the price to the target or to the limit
    ? limitSqrtPrice
    : step.nextTickPrice,

851 (zeroForOne == (step.nextTickPrice < limitSqrtPrice)) // move
      ↪ the price to the target or to the limit
    ? limitSqrtPrice
    : step.nextTickPrice,
```

3.59 CVF-59

- **Severity** Major
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** AlgebraPool.sol

Description Overflow and underflow are possible here.
Recommendation Consider using SafeMath.
Client Comment 1) if step.input appears to be greater than availableInputAmount than it is recalcuted so it becomes less than availableInputAmount and feeAmount is the difference between them. Thus, step.input + step.feeAmount is always less than or equal to amountRequired. 2) Inside PriceMovementMath.movePriceTowardsTarget the output is capped by amountRequired and is always positive. Since amountRequired is negative in this case the result will not underflow or be greater than zero. 3) delta is always less than step.feeAmount because it is calculated from it via multiplying by the number which is less than denominator.

Listing 59:

```
862 amountRequired -= (step.input + step.feeAmount).toInt256(); //
      ↪ decrease remaining input amount

865 amountRequired += step.output.toInt256(); // increase remaining
      ↪ output amount (since its negative)

870 uint256 delta = (step.feeAmount * cache.communityFee) / 100;
    step.feeAmount -= delta;
```

3.60 CVF-60

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Recommendation Getting to this "else" branch means the the stop condition is reached, so there should be a "break" statement inside this "else" branch.

Listing 60:

```
920 } else if (currentPrice != step.stepSqrtPrice) {
      // if the price has changed but hasn't reached the target
      currentTick = TickMath.getTickAtSqrtRatio(currentPrice);
    }
```

3.61 CVF-61

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Description Subtractive "feesX" from "paidX" is needed only when the corresponding community fee is non zero.
Recommendation Consider refactoring to avoid unnecessary operations.
Client Comment Considered.

Listing 61:

```
1002 totalFeeGrowth0Token += FullMath.mulDiv(paid0 - fees0, Constants
       ↪ .Q128, _liquidity);

1018 totalFeeGrowth1Token += FullMath.mulDiv(paid1 - fees1, Constants
       ↪ .Q128, _liquidity);
```

3.62 CVF-62

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** AlgebraPool.sol

Recommendation The value "26" should be a named constant.

Listing 62:

```
1026 require((communityFee0 < 26) && (communityFee1 < 26));
```

3.63 CVF-63

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AlgebraPool.sol

Description This function should emit some event.

Listing 63:

```
1041 function setLiquidityCooldown(uint32 newLiquidityCooldown)
       ↪ external override onlyFactoryOwner {
```

3.64 CVF-64

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** AlgebraPool.sol

Recommendation The value "1 days" should be a named constant.

Listing 64:

```
1042 require(newLiquidityCooldown <= 1 days);
```

3.65 CVF-65

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** AlgebraFactory.sol

Recommendation The type of this variable should be "IAlgebraPoolDeployer".

Listing 65:

```
20 address public immutable override poolDeployer;
```

3.66 CVF-66

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** AlgebraFactory.sol

Description The number format of these values is unclear.
Recommendation Consider documenting. Also consider rendering numbers like this for readability: 0.003e6 - Constants.BASE_FEE
Client Comment A short comment has been added.

Listing 66:

```
30 3000 - Constants.BASE_FEE, // alpha1
   15000 - 3000, // alpha2
   360, // beta1
   60000, // beta2
   59, // gamma1
   8500, // gamma2
   0, // volumeBeta
   10, // volumeGamma
```

3.67 CVF-67

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** AlgebraFactory.sol

Recommendation The type of the "_poolDeployer" argument should be "IAlgebraPoolDeployer".

Listing 67:

```
49 constructor(address _poolDeployer, address _vaultAddress) {
```

3.68 CVF-68

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AlgebraFactory.sol

Description Conversion to "address" and then to "IDataStorageOperator is redundant, as "DataStorageOperator" already implements "IDataStorageOperator".

Listing 68:

```
64 IDataStorageOperator dataStorage = IDataStorageOperator(address(
     ↪ new DataStorageOperator(computeAddress(token0, token1))));
```

3.69 CVF-69

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** AlgebraFactory.sol

Description These events are emitted even if nothing actually changed.

Listing 69:

```
77 emit OwnerChanged(owner, _owner);

83 emit FarmingAddressChanged(farmingAddress, _farmingAddress);

89 emit VaultAddressChanged(vaultAddress, _vaultAddress);
```

3.70 CVF-70

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AlgebraFactory.sol

Description This function should emit some event.

Listing 70:

```
94 function setBaseFeeConfiguration(
```

3.71 CVF-71

- **Severity** Major
- **Status** Fixed
- **Category** Flaw
- **Source** AlgebraFactory.sol

Description There are no range checks for the arguments.
Recommendation Consider adding appropriate checks.

Listing 71:

```
95  uint32 alpha1,
    uint32 alpha2,
    uint32 beta1,
    uint32 beta2,
    uint16 gamma1,
100 uint16 gamma2,
    uint32 volumeBeta,
    uint32 volumeGamma,
    uint16 baseFee
```

3.72 CVF-72

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** AlgebraFactory.sol

Recommendation This value could be accessed as: keccak256(type(AlgebraPool).creationCode) Hadrcoding this value is error prone.

Listing 72:

```
108 bytes32 internal constant POOL_INIT_CODE_HASH = 0
    ↪ xf2fb0dfbb88642c429660d73b8c1a79b60705a52a7bf09efb56675a8156d6b5b
    ↪ ;
```

3.73 CVF-73

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** PoolState.sol

Description The number format of these fields is unclear.
Recommendation Consider documenting.

Listing 73:

```
 9 uint160 price; // The square root of the current price

14 uint8 communityFeeToken0; // The community fee represented as a
    ↪ percent of all collected fee
   uint8 communityFeeToken1;
```

3.74 CVF-74

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** PoolState.sol

Recommendation The type of this variable should be "IAlgebraVirtualPool".

Listing 74:

```
35 address public override activeIncentive;
```

3.75 CVF-75

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** PoolState.sol

Description Here, the same storage slot it read three times, only it would be enough to read it only once.
Recommendation Consider refactoring to avoid extra reads.
Client Comment Noted. In the general case, we do not know if the state will change inside the function.

Listing 75:

```
44 require(globalState.unlocked, 'LOK');
   globalState.unlocked = false;

47 globalState.unlocked = true;
```

3.76 CVF-76

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** TickManager.sol

Recommendation Specifying an unbounded version range is a bad practice as one cannot guarantee compatibility with future major versions. Consider specifying as: "^0.5.0 || ^0.6.0 || ^0.7.0 || ^0.8.0". Also relevant for the next files: PriceMovementMath.sol, TickTable.sol, TokenDeltaMath.sol, Sqrt.sol, IAlgebraPoolState.sol, IAlgebraPoolPermissionedActions.sol, IAlgebraPoolImmutables.sol, IAlgebraPoolEvents.sol, IAlgebraPoolDerivedState.sol, IAlgebraPoolActions.sol, IDataStorageOperator.sol, IAlgebraPoolDeployer.sol, IAlgebraPool.sol, IAlgebraFactory.sol, IAlgebraSwapCallback.sol, IAlgebraMintCallback.sol, IAlgebraFlashCallback.sol.
Client Comment Library versions were limited. Interface versions are left as is.

Listing 76:

```
2 pragma solidity >=0.5.0;
```

3.77 CVF-77

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** TickManager.sol

Description We didn't review these files.
Client Comment These are audited open source files.

Listing 77:

```
4 import './LowGasSafeMath.sol';
  import './SafeCast.sol';

7 import './TickMath.sol';
  import './LiquidityMath.sol';
```

3.78 CVF-78

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** TickManager.sol

Description The number format of these fields is unclear.
Recommendation Consider documenting.

Listing 78:

```
18 uint128 liquidityTotal; // the total position liquidity that
     ↪ references this tick
   int128 liquidityDelta; // amount of net liquidity added (
     ↪ subtracted) when tick is crossed left-right (right-left),

22 uint256 outerFeeGrowth0Token;
   uint256 outerFeeGrowth1Token;
   int56 outerTickCumulative; // the cumulative tick value on the
     ↪ other side of the tick
   uint160 outerSecondsPerLiquidity; // the seconds per unit of
     ↪ liquidity on the _other_ side of current tick, (relative
     ↪ meaning)
```

3.79 CVF-79

- **Severity** Minor
- **Status** Info
- **Category** Overflow/Underflow
- **Source** TickManager.sol

Description Underflow is possible here.
Recommendation Consider using SafeMath.
Client Comment The function containing this code is internal and used in the single place. Upper and Lower Ticks are always validated at this point so upper tick cannot be less than lower. Considering TickManager.cross functionality there is 3 possible cases to calculate innerFeeGrowth: 1) current tick is between Lower and Upper: Lower.outerFeeGrowth repesents fee growth to the left from Lower tick. Upper.outerFeeGrowth represents fee growth to the right from Upper tick. To find innerFeeGrowth we substruct both of them from totalFeeGrowth 2) current tick is less than Lower: Lower.outerFeeGrowth repesents fee growth to the right from Lower tick. Upper.outerFeeGrowth represents fee growth to the right from Upper tick. Lower.outerFeeGrowth is greater than Upper.outerFeeGrowth so to find innerFeeGrowth we substruct Upper.outerFeeGrowth from Lower.outerFeeGrowth 3) current tick is greater than Upper: Lower.outerFeeGrowth repesents fee growth to the left from Lower tick. Upper.outerFeeGrowth represents fee growth to the left from Upper tick. Upper.outerFeeGrowth is greater than Lower.outerFeeGrowth so to find innerFeeGrowth we substruct Lower.outerFeeGrowth from Upper.outerFeeGrowth

Listing 79:

```
52    innerFeeGrowth0Token = totalFeeGrowth0Token - lower.
          ↪ outerFeeGrowth0Token;
      innerFeeGrowth1Token = totalFeeGrowth1Token - lower.
          ↪ outerFeeGrowth1Token;

58  innerFeeGrowth0Token -= upper.outerFeeGrowth0Token;
    innerFeeGrowth1Token -= upper.outerFeeGrowth1Token;

61  innerFeeGrowth0Token = upper.outerFeeGrowth0Token - lower.
        ↪ outerFeeGrowth0Token;
    innerFeeGrowth1Token = upper.outerFeeGrowth1Token - lower.
        ↪ outerFeeGrowth1Token;
```

3.80 CVF-80

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** TickManager.sol

Description There is a named constant for this value.
Recommendation Consider using this constant instead of a hardcoded value.

Listing 80:

```
95 require(liquidityTotalAfter <=
   ↪ 1150574359834111457188079822254499 4, 'LO');
```

3.81 CVF-81

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** TickManager.sol

Recommendation As absolute values of the cumulative tick properties doesn't matter, it would be cheaper to assume that all the growth happened at "this" side of the tick and thus just leave the cumulative fields uninitialised.
Client Comment Noted

Listing 81:

```
106 // by convention, we assume that all growth before a tick was
    ↪ initialized happened _below_ the tick
if (tick <= currentTick) {
```

3.82 CVF-82

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** TickManager.sol

Description This assignment is done in both branches of the conditional statement.
Recommendation Consider moving it outside the conditional statement.

Listing 82:

```
113 data.initialized = true;

115 data.initialized = true;
```

3.83 CVF-83

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** TickManager.sol

Description This argument is not documented.
Recommendation Consider documenting.

Listing 83:

```
134 int56 tickCumulative,
```

3.84 CVF-84

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** PriceMovementMath.sol

Description The price number format is unclear.
Recommendation Consider documenting.
Client Comment Added mention of Q64.96 format to comments.

Listing 84:

```
 21   uint160 price,

 25 ) internal pure returns (uint160 resultPrice) {

 37   uint160 price,

 41 ) internal pure returns (uint160 resultPrice) {

 46   uint160 price,

 51 ) internal pure returns (uint160 resultPrice) {

138   uint160 currentPrice,
      uint160 targetPrice,

147     uint160 resultPrice,
```

3.85 CVF-85

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** PriceMovementMath.sol

Recommendation The name "zeroToOne" would be more clear. Zero for one could be understood as "sell token zero for token one" or "buy token zero for token one".

Listing 85:

```
24  bool zeroForOne

40  bool zeroForOne

49  bool zeroForOne,

137 bool zeroForOne,
```

3.86 CVF-86

- **Severity** Moderate
- **Status** Info
- **Category** Overflow/Underflow
- **Source** PriceMovementMath.sol

Description This code may throw in Solidity ^0.8.0 .
Recommendation Consider using an overflow-safe method here.
Client Comment Pragma solidity changed.

Listing 86:

```
62  if ((product = amount * price) / amount == price) {

70  require((product = amount * price) / amount == price); // if the
    ↪  product overflows, we know the denominator underflows
```

3.87 CVF-87

- **Severity** Moderate
- **Status** Fixed
- **Category** Unclear behavior
- **Source** PriceMovementMath.sol

Description There is not range check for the "fee" argument. Thus underflow is possible.
Recommendation Consider added an appropriate check.
Client Comment Argument type changed to uint16.

Listing 87:

```
142 uint24 fee

157   uint256 amountAvailableAfterFee = FullMath.mulDiv(uint256(
        ↪ amountAvailable), 1e6 - fee, 1e6);

161     feeAmount = FullMath.mulDivRoundingUp(input, fee, 1e6 - fee)
          ↪ ;

170       feeAmount = FullMath.mulDivRoundingUp(input, fee, 1e6 -
            ↪ fee);

194   feeAmount = FullMath.mulDivRoundingUp(input, fee, 1e6 - fee);
```

3.88 CVF-88

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** PriceMovementMath.sol

Description The expression "1e6 - fee" is calculated several times.
Recommendation Consider calculating once and reusing.
Client Comment Noted.

Listing 88:

```
157 uint256 amountAvailableAfterFee = FullMath.mulDiv(uint256(
      ↪ amountAvailable), 1e6 - fee, 1e6);

161   feeAmount = FullMath.mulDivRoundingUp(input, fee, 1e6 - fee);

170     feeAmount = FullMath.mulDivRoundingUp(input, fee, 1e6 - fee)
          ↪ ;

194 feeAmount = FullMath.mulDivRoundingUp(input, fee, 1e6 - fee);
```

3.89 CVF-89

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** PriceMovementMath.sol

Description The expression "-amountAvailable" is calculated several times.
Recommendation Consider calculating once and reusing.

Listing 89:

```
179 if (uint256(-amountAvailable) >= output) resultPrice =
      ↪ targetPrice;

181   resultPrice = getNewPriceAfterOutput(currentPrice, liquidity,
        ↪ uint256(-amountAvailable), zeroForOne);

188   if (output > uint256(-amountAvailable)) {
        output = uint256(-amountAvailable);
```

3.90 CVF-90

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** TickTable.sol

Description There is an immutable variable named "tickSpacing" for the value "60".
Recommendation Consider using it instead of a hardcoded value.

Listing 90:

```
 12 require(tick % 60 == 0, 'tick is not spaced'); // ensure that
      ↪ the tick is spaced
    tick /= 60; // compress tick

 68   tick := sub(sdiv(tick, 60), and(slt(tick, 0), not(iszero(smod(
        ↪ tick, 60)))))

112 boundedTick = tick * 60;
```

3.91 CVF-91

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** TickTable.sol

Recommendation These operations could be done in plain Solidity. No assembly required.
Client Comment Noted, assembly shows better gas.

Listing 91:

```
17 assembly {
     bitNumber := and(tick, 0xFF)
     rowNumber := shr(8, tick)
20 }
```

3.92 CVF-92

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** TickTable.sol

Description There is no validity check for the argument.
Recommendation Such check could be done quite efficiently: require (x & (x - 1) == 0);
Client Comment We rely on validation from the caller. Added relevant comment.

Listing 92:

```
26 function getSingleSignificantBit(uint256 word) internal pure
     ↪ returns (uint8 singleBitPos) {
```

3.93 CVF-93

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** TickTable.sol

Recommendation This could be optimised as: singleBitPos := iszero(and(word, 0x5555555555555555555555555555555555555555555555555555555555555555))

Listing 93:

```
28 singleBitPos := gt(and(word, 0
     ↪ xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
     ↪ ), 0)
```

3.94 CVF-94

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** TickTable.sol

Recommendation These lines could be optimised by replacing “gt(x, 0)” with “iszero(x)" and inverting the masks.

Listing 94:

```
29 singleBitPos := or(singleBitPos, shl(7, gt(and(word, 0
   ↪ xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000000000000000000000000000
   ↪ ), 0)))
30 singleBitPos := or(singleBitPos, shl(6, gt(and(word, 0
   ↪ xFFFFFFFFFFFFFFFF0000000000000000FFFFFFFFFFFFFFFF0000000000000000
   ↪ ), 0)))
   singleBitPos := or(singleBitPos, shl(5, gt(and(word, 0
   ↪ xFFFFFFFF00000000FFFFFFFF00000000FFFFFFFF00000000FFFFFFFF00000000
   ↪ ), 0)))
   singleBitPos := or(singleBitPos, shl(4, gt(and(word, 0
   ↪ xFFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000
   ↪ ), 0)))
   singleBitPos := or(singleBitPos, shl(3, gt(and(word, 0
   ↪ xFF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00
   ↪ ), 0)))
   singleBitPos := or(singleBitPos, shl(2, gt(and(word, 0
   ↪ xF0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0
   ↪ ), 0)))
   singleBitPos := or(singleBitPos, shl(1, gt(and(word, 0
   ↪ xCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
   ↪ ), 0)))
```

3.95 CVF-95

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** TickTable.sol

Description For a zero argument this function returns zero, which is, strictly speaking, incorrect.
Recommendation Consider adding an explicit “require” statement to check that the argument is not zero.
Client Comment We rely on validation from the caller. Added relevant comment.

Listing 95:

```
39 function getMostSignificantBit(uint256 word) internal pure
   ↪ returns (uint8 mostBitPos) {
```

3.96 CVF-96

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** TickTable.sol

Description The returned values are actually unnamed, so referring them here by names is confusing.
Recommendation Consider naming the returned values.

Listing 96:

```
59 /// @return nextTick The next initialized or uninitialized tick
     ↪ up to 256 ticks away from the current tick
60 /// @return initialized Whether the next tick is initialized, as
     ↪  the function only searches within up to 256 ticks
```

3.97 CVF-97

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** TickTable.sol

Recommendation The value "887272" should be a named constant.

Listing 97:

```
113 if (boundedTick < -887272) {
      boundedTick = -887272;
    } else if (boundedTick > 887272) {
      boundedTick = 887272;
```

3.98 CVF-98

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** TokenDeltaMath.sol

Description We didn't review these files.
Client Comment These are audited open source files.

Listing 98:

```
4 import './LowGasSafeMath.sol';
  import './SafeCast.sol';
```

3.99 CVF-99

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** TokenDeltaMath.sol

Description It is unclear why a more complicated formula is preferred over a simpler one.
Recommendation Consider explaining.

Listing 99:

```
17 /// @dev Calculates liquidity / sqrt(lower) - liquidity / sqrt(
   ↪ upper),
/// i.e. liquidity * (sqrt(upper) - sqrt(lower)) / (sqrt(upper)
   ↪ * sqrt(lower))
```

3.100 CVF-100

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** TokenDeltaMath.sol

Description The price number format is unclear.
Recommendation Consider documenting.
Client Comment Mentioned Q64.96 format in description.

Listing 100:

```
25 uint160 priceLower,
   uint160 priceUpper,

47 uint160 priceLower,
   uint160 priceUpper,

62 uint160 priceLower,
   uint160 priceUpper,

77 uint160 priceLower,
   uint160 priceUpper,
```

3.101 CVF-101

- **Severity** Moderate
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** TokenDeltaMath.sol

Description Underflow is possible here.
Recommendation Consider using SafeMath.

Listing 101:

```
31 uint256 priceDelta = priceUpper - priceLower;
```

3.102 CVF-102

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** TokenDeltaMath.sol

Description While the "muldiv" function offers great performance in general case, for specific cases there are much more efficient approaches. For example, when the denominator is a known power of two, it is possible to implement a "mulshift" function that calculates a 512-bit product of two numbers and then shifts it right to get a 256-bit result.
Recommendation Consider implementing and using such a function.
Client Comment Considered for the future.

Listing 102:

```
53 token1Delta = roundUp ? FullMath.mulDivRoundingUp(priceDelta,
   ↪ liquidity, Constants.Q96) : FullMath.mulDiv(priceDelta,
   ↪ liquidity, Constants.Q96);
```

3.103 CVF-103

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** Sqrt.sol

Description This function calculates the square root of the absolute value of "x", rather than just the square root of "x".
Recommendation Consider explaining this and/or reflecting in the function name.

Listing 103:

```
5 function sqrt(int256 _x) internal pure returns (uint256 result)
   ↪ {
```

3.104 CVF-104

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** PIFee.sol

Description The number format of a fee is unclear.
Recommendation Consider documenting.

Listing 104:

```
5 uint16 internal constant maxFee = 50000;
```

3.105 CVF-105

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** PIFee.sol

Recommendation The name "zeroToOne" would be more clear. Zero for one could be understood as "sell token zero for token one" or "buy token zero for token one".

Listing 105:

```
8 bool zeroForOne,
```

3.106 CVF-106

- **Severity** Critical
- **Status** Fixed
- **Category** Procedural
- **Source** PIFee.sol

Recommendation This should be removed before doing final tests.

Listing 106:

```
14 return currentFee; // MOCK, WIP
```

3.107 CVF-107

- **Severity** Moderate
- **Status** Fixed
- **Category** Suboptimal
- **Source** PIFee.sol

Description This doesn't allow fee to ever go down after reaching "maxFee".
Recommendation Consider removing this check.

Listing 107:

```
16 if (currentFee == maxFee) {
     return maxFee;
   }
```

3.108 CVF-108

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** PIFee.sol

Recommendation Numeric values used here should be named constants.

Listing 108:

```
24   deviationPercent = (((startPrice * 1e3) / currentPrice)**2 - 1
       ↪ e6);

26   deviationPercent = (((currentPrice * 1e3) / startPrice)**2 - 1
       ↪ e6);

31 if (deviationPercent < 500) {
     fee = uint16(((deviationPercent + 125) * startFee) / 625);

34   fee = uint16(((deviationPercent + 8500) * startFee) / 9000);
```

3.109 CVF-109

- **Severity** Moderate
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** PIFee.sol

Description Underflow is possible here.
Recommendation Consider using SafeMath.

Listing 109:

```
24 deviationPercent = (((startPrice * 1e3) / currentPrice)**2 - 1e6
   ↪ );

26 deviationPercent = (((currentPrice * 1e3) / startPrice)**2 - 1e6
   ↪ );
```

3.110 CVF-110

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** FullMath.sol

Recommendation Specifying an unbounded version range is a bad practice as one cannot guarantee compatibility with future major versions. Consider specifying as: "^0.4.0 || ^0.5.0 || ^0.6.0 || ^0.7.0 || ^0.8.0".

Listing 110:

```
2 pragma solidity >=0.4.0;
```

3.111 CVF-111

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** FullMath.sol

Description This code is actually incompatible with Solidity 0.8.0+.
Recommendation Consider fixing the version requirement to reflect this fact.

Listing 111:

```
2 pragma solidity >=0.4.0;
```

3.112 CVF-112

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** FullMath.sol

Description The code below looks like it is always executed, while it is executed only when a * b doesn't fit into a word.
Recommendation Consider putting the rest of the function into an explicit "else" branch.

Listing 112:

```
119 }
```

3.113 CVF-113

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** DataStorage.sol

Description The number format of the returned value is unclear.
Recommendation Consider documenting.
Client Comment Added comment with maximum value. Type uint256 left for fuzzy testing.

Listing 113:

```
36 ) private pure returns (uint256 volatility) {
```

3.114 CVF-114

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** DataStorage.sol

Description The comment doesn't help to understand the code.
Recommendation Consider explaining the formulas in more details.
Client Comment Comments have been detailed.

Listing 114:

```
37 // On the interval from the previous timpoint to the current
   // tick and the and tick are straight lines that satisfy the
      ↪ equations
   // yt = k*x + b   and  yat = p*x + q
40 // so: ((k*x + b) - (p*x + q))^2 = ((k-p)*x + (b-q))^2 = (k-p)^2
      ↪  * x^2 + 2(k-p)(b-q)x + (b-q)^2
   // sum from 0 to N require to use arithmetic and squares
      ↪ progressions
   int256 K = (tick1 - tick0) - (avgTick1 - avgTick0);
   int256 B = (tick0 - avgTick0) * dt;
   int256 sumOfSquares = (dt * (dt + 1) * (2 * dt + 1)) / 6;
   int256 sumOfSequence = (dt * (dt + 1)) / 2;
   volatility = uint256((K**2 * sumOfSquares + 2 * B * K *
      ↪ sumOfSequence + (dt) * B**2) / dt**2);
```

3.115 CVF-115

- **Severity** Moderate
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** DataStorage.sol

Description Overflow and underflow are possible here.
Recommendation Consider using SafeMath.
Client Comment Added references to the boundaries of parameter values in the comments. With correct parameters, overflows/underflows are impossible, and the result always fits into 88 bits.

Listing 115:

```
42 int256 K = (tick1 - tick0) - (avgTick1 - avgTick0);
   int256 B = (tick0 - avgTick0) * dt;
   int256 sumOfSquares = (dt * (dt + 1) * (2 * dt + 1)) / 6;
   int256 sumOfSequence = (dt * (dt + 1)) / 2;
   volatility = uint256((K**2 * sumOfSquares + 2 * B * K *
      ↪ sumOfSequence + (dt) * B**2) / dt**2);
```

3.116 CVF-116

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** DataStorage.sol

Description This constraint is not checked.
Recommendation Consider checking it explicitly.

Listing 116:

```
50 /// @dev blockTimestamp _must_ be chronologically equal to or
   ↪ greater than last.blockTimestamp, safe for 0 or 1
   ↪ overflows
```

3.117 CVF-117

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** DataStorage.sol

Description This argument is not documented.

Listing 117:

```
62 int24 prevTick,
```

3.118 CVF-118

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** DataStorage.sol

Description This modifies the passed structure in place, so the caller will see the changes after the call. Such side effects make the code more error-prone.
Recommendation Consider allocating, populating, and returning a new structure.

Listing 118:

```
69 last.initialized = true;
70 last.blockTimestamp = blockTimestamp;
   last.tickCumulative += int56(tick) * delta;
   last.secondsPerLiquidityCumulative += ((uint160(delta) << 128) /
   ↪ (liquidity > 0 ? liquidity : 1));
   last.volatilityCumulative += uint88(_volatilityOnRange(delta,
   ↪ prevTick, tick, last.averageTick, averageTick));
   last.averageTick = averageTick;
   last.volumePerLiquidityCumulative += volumePerLiquidity;
```

3.119 CVF-119

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** DataStorage.sol

Description Here zero liquidity is treated as one liquidity. The rationale behind such decision is unclear.
Recommendation Consider explaining.
Client Comment In the case of an empty pool, just the time delta is recorded.

Listing 119:

```
72 last.secondsPerLiquidityCumulative += ((uint160(delta) << 128) /
   ↪ (liquidity > 0 ? liquidity : 1));
```

3.120 CVF-120

- **Severity** Moderate
- **Status** Info
- **Category** Unclear behavior
- **Source** DataStorage.sol

Description Here volatility is added without weight. Is this fine?
Client Comment It's fine, since volatility resampled with 1 sec frequency. So we are using values for each second.

Listing 120:

```
73 last.volatilityCumulative += uint88(_volatilityOnRange(delta,
   ↪ prevTick, tick, last.averageTick, averageTick));
```

3.121 CVF-121

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** DataStorage.sol

Description Two ways of returning a value are mixed here: assigning the return value and "return" statement. This makes code harder to read.
Recommendation Consider refactoring like this: res = a > currentTime; if (res == b < currentTime) res = a <= b;

Listing 121:

```
91 res = (a > currentTime);
   if (res == (b > currentTime)) return a <= b; // if both are on
   ↪ the same side
```

3.122 CVF-122

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** DataStorage.sol

Recommendation Brackets around comparisons are redundant.

Listing 122:

```
91 res = (a > currentTime);
   if (res == (b > currentTime)) return a <= b; // if both are on
      ↪ the same side
```

3.123 CVF-123

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** DataStorage.sol

Recommendation The value "65535" should be a named constant.

Listing 123:

```
 97 Timepoint[65535] storage self,

110     index = index == 0 ? 65535 - 1 : index - 1;

139 Timepoint[65535] storage self,

146 uint256 right = lastIndex >= oldestIndex ? lastIndex : lastIndex
    ↪ + 65535; // newest timepoint considering overflow

149 beforeOrAt = self[current % 65535];

156         atOrAfter = self[addmod(current, 1, 65535)];

169   beforeOrAt = self[current % 65535];

185 Timepoint[65535] storage self,

207           prevLast.blockTimestamp = self[(index - 1) % 65535].
                ↪ blockTimestamp;
              prevLast.tickCumulative = self[(index - 1) % 65535].
                ↪ tickCumulative;

253 Timepoint[65535] storage self,

276 if (self[addmod(index, 1, 65535)].initialized) {
      oldestIndex = uint16(addmod(index, 1, 65535));

301 Timepoint[65535] storage self,

309 if (self[addmod(index, 1, 65535)].initialized) {
310   oldestIndex = uint16(addmod(index, 1, 65535));

332 Timepoint[65535] storage self,

351 Timepoint[65535] storage self,

365 indexUpdated = uint16(addmod(index, 1, 65535));

378     prevLast.blockTimestamp = self[(index - 1) % 65535].
          ↪ blockTimestamp;
        prevLast.tickCumulative = self[(index - 1) % 65535].
          ↪ tickCumulative;
```

3.124 CVF-124

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** DataStorage.sol

Description The expression "time - WINDOW" is calculated twice.
Recommendation Consider calculating once and reusing.
Client Comment It looks like this change causes more gas usage due to additional stack operations than it saves.

Listing 124:

```
108 if (lteConsideringOverflow(oldestTimestamp, time - WINDOW, time)
    ↪ ) {
    if (lteConsideringOverflow(lastTimestamp, time - WINDOW, time)
      ↪ ) {

120     avgTick = int24((lastTickCumulative - startOfWindow.
          ↪ tickCumulative) / (lastTimestamp - time + WINDOW));
```

3.125 CVF-125

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** DataStorage.sol

Recommendation This could be simplified as; index += 65534;

Listing 125:

```
110 index = index == 0 ? 65535 - 1 : index - 1;
```

3.126 CVF-126

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** DataStorage.sol

Description The expression "self[index]" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 126:

```
112 ? int24((lastTickCumulative - self[index].tickCumulative) / (
    ↪ lastTimestamp - self[index].blockTimestamp))
```

3.127 CVF-127

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** DataStorage.sol

Description In case "lastTimestamp" is only marginally higher than "time - WINDOW", this formula will calculate an average tick for a very short interval. Probably, not an issue.
Client Comment Noted.

Listing 127:

```
120 avgTick = int24((lastTickCumulative - startOfWindow.
    ↪ tickCumulative) / (lastTimestamp - time + WINDOW));
```

3.128 CVF-128

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** DataStorage.sol

Recommendation This could be done in one place: at the beginning of the loop. No need to do in two places.

Listing 128:

```
149 beforeOrAt = self[current % 65535];

169   beforeOrAt = self[current % 65535];
```

3.129 CVF-129

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** DataStorage.sol

Description This assignment has no effect, as the assigned value is guaranteed to be overwritten.
Recommendation Consider removing the assignment.
Client Comment Fixed, was used to suppress compiler error due to potentially undefined storage pointer.

Listing 129:

```
150 atOrAfter = beforeOrAt; // will override in cycle
```

3.130 CVF-130

- **Severity** Critical
- **Status** Fixed
- **Category** Flaw
- **Source** DataStorage.sol

Description Here "atOrAfter" may point to an uninitialised slot.
Recommendation Consider, for example, a situation when lastIndex == oldestIndex.
Client Comment This situation is not possible due to the way contracts work. However, additional handling of the situation has been added.

Listing 130:

```
156 atOrAfter = self[addmod(current, 1, 65535)];
```

3.131 CVF-131

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** DataStorage.sol

Recommendation The argument "self" is documented twice.

Listing 131:

```
177 /// @param self The stored dataStorage array
    /// @param self The oldest timepoint
```

3.132 CVF-132

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** DataStorage.sol

Recommendation This argument is not documented.

Listing 132:

```
190 uint16 oldestIndex,
```

3.133 CVF-133

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** DataStorage.sol

Recommendation The returned value is not documented.

Listing 133:

```
192 ) internal view returns (Timepoint memory) {
```

3.134 CVF-134

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** DataStorage.sol

Description The expression "self[(index - 1) % 65535]" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 134:

```
207 prevLast.blockTimestamp = self[(index - 1) % 65535].
    ↪ blockTimestamp;
    prevLast.tickCumulative = self[(index - 1) % 65535].
    ↪ tickCumulative;
```

3.135 CVF-135

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** DataStorage.sol

Recommendation Should be "else if" for readability.
Client Comment Noted.

Listing 135:

```
223 if (target != beforeOrAt.blockTimestamp) {
```

3.136 CVF-136

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** DataStorage.sol

Description Writing the intermediate calculated time point into the "beforeOrAt" variable changes the meaning of the "beforeOrAt" variable and the new meaning is inconsistent with the variable name.
Recommendation Consider using a separate variable for the intermediary time point.
Client Comment Comment added.

Listing 136:

```
228 beforeOrAt.tickCumulative += ((atOrAfter.tickCumulative -
    ↪ beforeOrAt.tickCumulative) / timepointTimeDelta) *
    ↪ targetDelta;
beforeOrAt.secondsPerLiquidityCumulative += uint160(

232 beforeOrAt.volatilityCumulative += ((atOrAfter.
    ↪ volatilityCumulative - beforeOrAt.volatilityCumulative) /
    ↪ timepointTimeDelta) * targetDelta;
beforeOrAt.volumePerLiquidityCumulative +=
```

3.137 CVF-137

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** DataStorage.sol

Recommendation Should be "else return" for readability.
Client Comment Noted.

Listing 137:

```
239 return beforeOrAt;
```

3.138 CVF-138

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** DataStorage.sol

Recommendation It would be more efficient to return a single array of structs with four fields, rather than four parallel arrays.

Listing 138:

```
263 int56[] memory tickCumulatives,
    uint160[] memory secondsPerLiquidityCumulatives,
    uint112[] memory volatilityCumulatives,
    uint256[] memory volumePerAvgLiquiditys
```

3.139 CVF-139

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** DataStorage.sol

Description These returned values are not documented.

Listing 139:

```
265 uint112[] memory volatilityCumulatives,
    uint256[] memory volumePerAvgLiquiditys
```

3.140 CVF-140

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** DataStorage.sol

Description The expression "addmod(index, 1, 65535)" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 140:

```
276 if (self[addmod(index, 1, 65535)].initialized) {
      oldestIndex = uint16(addmod(index, 1, 65535));
```

3.141 CVF-141

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** DataStorage.sol

Recommendation Here "@return" is missing.

Listing 141:

```
299 /// volumePerLiqAverage The average volume per liquidity in the
    ↪ recent range
```

3.142 CVF-142

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** DataStorage.sol

Description The expression "addmod(index, 1, 65536)" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 142:

```
309 if (self[addmod(index, 1, 65535)].initialized) {
310   oldestIndex = uint16(addmod(index, 1, 65535));
```

3.143 CVF-143

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** DataStorage.sol

Description There is no check that the table is not yet initialised.
Recommendation Consider adding such explicit check.

Listing 143:

```
336 self[0].initialized = true;
```

3.144 CVF-144

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** DataStorage.sol

Description The expression “self[index]” is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 144:

```
359 if (self[index].blockTimestamp == blockTimestamp) {

362 Timepoint memory last = self[index];
```

3.145 CVF-145

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Constants.sol

Description The nunber format of this value is unclear.
Recommendation Consider documenting.

Listing 145:

```
8 uint16 internal constant BASE_FEE = 100;
```

3.146 CVF-146

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** AdaptiveFee.sol

Recommendation Should be "^0.7.0" unless there is something special about this particular version.
Client Comment This library is not supposed to be compiled in another version.

Listing 146:

```
2 pragma solidity =0.7.6;
```

3.147 CVF-147

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** AdaptiveFee.sol

Description The semantics and the number format of these fields are unclear.
Recommendation Consider documenting.
Client Comment Comments have been added.

Listing 147:

```
8  uint32 alpha1;
   uint32 alpha2;
10 uint32 beta1;
   uint32 beta2;
   uint16 gamma1;
   uint16 gamma2;
   uint32 volumeBeta;
   uint32 volumeGamma;
   uint16 baseFee;
```

3.148 CVF-148

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** AdaptiveFee.sol

Description The number format of these arguments in unclear.
Recommendation Consider documenting.

Listing 148:

```
20 uint88 volatility,
   uint256 volumePerLiquidity,
```

3.149 CVF-149

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AdaptiveFee.sol

Description The conversion to "uint256" is redundant, as compiler will do it anyway.
Recommendation Consider removing the conversion.

Listing 149:

```
27 fee = uint256(config.baseFee) + sigmoid(volumePerLiquidity,
     ↪ config.volumeGamma, sigm1 + sigm2, config.volumeBeta);
```

3.150 CVF-150

- **Severity** Moderate
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** AdaptiveFee.sol

Description Overflow is possible here.
Recommendation Consider using SafeMath,
Client Comment Added checks, comments and fuzzy tests.

Listing 150:

```
27 fee = uint256(config.baseFee) + sigmoid(volumePerLiquidity,
     ↪ config.volumeGamma, sigm1 + sigm2, config.volumeBeta);

38   if (x >= 6 * g) return alpha;

40   res = ((10 * alpha * (ex)) / (g**7 + ex)) / 10;

43   if (x >= 6 * g) return 0;
     uint256 ex = g**7 + exp(x, g);
     res = ((10 * alpha * g**7) / (ex)) / 10;

50 return g**7 + x * g**6 + (x**2 * g**5) / 2 + (x**3 * g**4) / 6 +
     ↪  (x**4 * g**3) / 24 + (x**5 * g**2) / 120 + (x**6 * g) /
     ↪ 720 + x**7 / (720 * 7);
```

3.151 CVF-151

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** AdaptiveFee.sol

Recommendation This function could be calculated in a straightforward way using a fixed-point or floating-point library, such as ABDKMath64x64, or using a custom implementation of fixed-point or floating-point math.

Listing 151:

```
30 function sigmoid(
```

3.152 CVF-152

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AdaptiveFee.sol

Description The expression "g**7" is calculated several times.
Recommendation Consider calculating once and reusing.

Listing 152:

```
40    res = ((10 * alpha * (ex)) / (g**7 + ex)) / 10;

45    res = ((10 * alpha * g**7) / (ex)) / 10;

50  return g**7 + x * g**6 + (x**2 * g**5) / 2 + (x**3 * g**4) / 6 +
      ↪  (x**4 * g**3) / 24 + (x**5 * g**2) / 120 + (x**6 * g) /
      ↪ 720 + x**7 / (720 * 7);
```

3.153 CVF-153

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AdaptiveFee.sol

Description Various powers of "x" and "g" are calculated independently.
Recommendation Consider calculating incrementally like this: uint256 x2 = x*x; uint256 x3 = x2 * x; uint256 x4 = x3 * x; uint256 x5 = x4 * x; uint256 x6 = x5 * x; uint256 x7 = x6 * x; uint256 g2 = g*g; uint256 g3 = g2 * g; uint256 g4 = g3 * g; uint256 g5 = g4 * g; uint256 g6 = g5 * g; uint256 g7 = g6 * g; return g7 + x * g6 + x2 * g5 / 2 + x3 * g4 / 6 + x4 * g3 / 24 + x5 * g2 / 120 + x6 * g / 720 + x / 5040;
Client Comment Refactored incrementally.

Listing 153:

```
40    res = ((10 * alpha * (ex)) / (g**7 + ex)) / 10;

45    res = ((10 * alpha * g**7) / (ex)) / 10;

50  return g**7 + x * g**6 + (x**2 * g**5) / 2 + (x**3 * g**4) / 6 +
      ↪  (x**4 * g**3) / 24 + (x**5 * g**2) / 120 + (x**6 * g) /
      ↪ 720 + x**7 / (720 * 7);
```

3.154 CVF-154

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraPoolState.sol

Recommendation Here "@return" is missing.
Client Comment Fixed where possible. This version of the compiler does not work well with multiple "@return" tags for state variables.

Listing 154:

```
13 * timepointIndex The index of the last written timepoint
   * timepointIndexSwap The index of the last written (on swap)
     ↪ timepoint
   * communityFeeToken0, communityFeeToken1 The community fee for
     ↪ both tokens of the pool.
   * unlocked Whether the pool is currently locked to reentrancy

56 * outerFeeGrowth0Token the fee growth on the other side of the
     ↪ tick from the current tick in token0,
   * outerFeeGrowth1Token the fee growth on the other side of the
     ↪ tick from the current tick in token1,
   * outerTickCumulative the cumulative tick value on the other
     ↪ side of the tick from the current tick
   * outerSecondsPerLiquidity the seconds spent per liquidity on
     ↪ the other side of the tick from the current tick,
60 * outerSecondsSpent the seconds spent on the other side of the
     ↪ tick from the current tick,
   * initialized Set to true if the tick is initialized, i.e.
     ↪ liquidityTotal is greater than 0,

87 * innerFeeGrowth0Token fee growth of token0 inside the tick
     ↪ range as of the last mint/burn/poke,
   * innerFeeGrowth1Token fee growth of token1 inside the tick
     ↪ range as of the last mint/burn/poke,
   * fees0 the computed amount of token0 owed to the position as of
     ↪  the last mint/burn/poke,
90 * fees1 the computed amount of token1 owed to the position as of
     ↪  the last mint/burn/poke
```

(... 110)

3.155 CVF-155

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraPoolState.sol

Description These returned values are not documented.
Recommendation Consider documenting them.

Listing 155:

```
24 uint16 fee,

28 uint8 communityFeeToken1,
```

3.156 CVF-156

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraPoolState.sol

Description The number format of these values is unclear.
Recommendation Consider documenting.

Listing 156:

```
27 uint8 communityFeeToken0,
   uint8 communityFeeToken1,
```

3.157 CVF-157

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraPoolState.sol

Description The number format of the returned value is unclear.
Recommendation Consider documenting.
Client Comment Comment extended.

Listing 157:

```
48 function liquidity() external view returns (uint128);
```

3.158 CVF-158

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IAlgebraPoolState.sol

Description The name is confusing. The function returns information about a single tick, rather than about several ones.
Recommendation Consider renaming.
Client Comment Noted. This is actually a mapping.

Listing 158:

```
66 function ticks(int24 tick)
```

3.159 CVF-159

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IAlgebraPoolState.sol

Description The name is confusing. The function returns information about a single position, rather than about several ones.
Recommendation Consider renaming.
Client Comment Noted. This is actually a mapping.

Listing 159:

```
92 function positions(bytes32 key)
```

3.160 CVF-160

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** IAlgebraPoolState.sol

Description In contrast with names of other returned values in this file, this name is prefixed with underscore ("_").
Recommendation Consider removing the prefix.

Listing 160:

```
96 uint128 _liquidity,
```

3.161 CVF-161

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraPoolState.sol

Description This returned value is not documented.
Recommendation Consider documenting.

Listing 161:

```
97 uint32 lastModificationTimestamp,
```

3.162 CVF-162

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraPoolState.sol

Description There is no returned value with such name.

Listing 162:

```
114 * volumePerAvgLiquidity Cumulative swap volume per liquidity for
    ↪ the life of the pool as of the timepoint timestamp
```

3.163 CVF-163

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraPoolState.sol

Description These returned values are not documented.

Listing 163:

```
125 int24 averageTick,
uint144 volumePerLiquidityCumulative
```

3.164 CVF-164

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraPoolState.sol

Recommendation The return type should be "IAlgebraVirtualPool".

Listing 164:

```
136 function activeIncentive() external view returns (address
    ↪ virtualPool);
```

3.165 CVF-165

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraPoolPermissionedActions.sol

Description The number format of the argument is unclear. Using only 8 bits for fees mans that value precision is quite coarse.
Recommendation Consider explaining the number format.
Client Comment Clarifying comment added.

Listing 165:

```
14 function setCommunityFee(uint8 communityFee0, uint8
   ↪ communityFee1) external;
```

3.166 CVF-166

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraPoolPermissionedActions.sol

Recommendation The argument type should be "IAlgebraVirtualPool".

Listing 166:

```
20 function setIncentive(address virtualPoolAddress) external;
```

3.167 CVF-167

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraPoolImmutables.sol

Recommendation The return type should be "IAlgebraFactory".

Listing 167:

```
18 function factory() external view returns (address);
```

3.168 CVF-168

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraPoolImmutables.sol

Recommendation The return type should be "IERC20".

Listing 168:

```
24 function token0() external view returns (address);

30 function token1() external view returns (address);
```

3.169 CVF-169

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** IAlgebraPoolImmutables.sol

Description The comment says that the value is "int24" while actually the returned value is "uint8".
Recommendation Consider changing the return value type to "int24".

Listing 169:

```
36  * This value is an int24 to avoid casting even though it is
      ↪ always positive.

39 function tickSpacing() external view returns (uint8);
```

3.170 CVF-170

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IAlgebraPoolEvents.sol

Description The "tick" parameter is redundant as it could be derived from the "price" parameter.

Listing 170:

```
12 event Initialize(uint160 price, int24 tick);
```

3.171 CVF-171

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** IAlgebraPoolEvents.sol

Description The parameter name "amount" is too generic and there are other amount parameters in the same event.
Recommendation Consider renaming to "liquidityAmout".

Listing 171:

```
29    uint128 amount,

55  event Burn(address indexed owner, int24 indexed bottomTick,
      ↪ int24 indexed topTick, uint128 amount, uint256 amount0,
      ↪ uint256 amount1);
```

3.172 CVF-172

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraPoolEvents.sol

Description The "recipient" parameter is not documented.

Listing 172:

```
43  event Collect(address indexed owner, address recipient, int24
      ↪ indexed bottomTick, int24 indexed topTick, uint128 amount0
      ↪ , uint128 amount1);
```

3.173 CVF-173

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** IAlgebraPoolEvents.sol

Description The old fee parameters are redundant as their values could be derived from the previous events.

Listing 173:

```
87  event SetCommunityFee(uint8 communityFee0Old, uint8
      ↪ communityFee1Old, uint8 communityFee0New, uint8
      ↪ communityFee1New);
```

3.174 CVF-174

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** IAlgebraPoolEvents.sol

Recommendation Events are usually named via nouns.

Listing 174:

```
87 event SetCommunityFee(uint8 communityFee0Old, uint8
     ↪ communityFee1Old, uint8 communityFee0New, uint8
     ↪ communityFee1New);

96 event CollectCommunityFee(address indexed sender, address
     ↪ indexed recipient, uint128 amount0, uint128 amount1);

102 event IncentiveSet(address virtualPoolAddress);

107 event ChangeFee(uint16 Fee);
```

3.175 CVF-175

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** IAlgebraPoolEvents.sol

Recommendation The parameter should be indexed.

Listing 175:

```
102 event IncentiveSet(address virtualPoolAddress);
```

3.176 CVF-176

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IAlgebraPoolDerivedState.sol

Description The number formats of these values are unclear.
Recommendation Consider documenting.

Listing 176:

```
18 * @return tickCumulatives Cumulative tick values as of each `
     ↪ secondsAgos` from the current block timestamp
   * @return secondsPerLiquidityCumulatives Cumulative seconds per
     ↪ liquidity-in-range value as of each `secondsAgos`

21 * @return volatilityCumulatives Cumulative standard deviation as
     ↪  of each `secondsAgos`
   * @return volumePerAvgLiquiditys Cumulative swap volume per
     ↪ liquidity as of each `secondsAgos`

49     int56 innerTickCumulative,
50     uint160 innerSecondsSpentPerLiquidity,
       uint32 innerSecondsSpent
```

3.177 CVF-177

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IAlgebraPoolDerivedState.sol

Recommendation It would be more efficient to return a single array of structs with four fields, rather than four parallel arrays.

Listing 177:

```
28 int56[] memory tickCumulatives,
   uint160[] memory secondsPerLiquidityCumulatives,
30 uint112[] memory volatilityCumulatives,
   uint256[] memory volumePerAvgLiquiditys
```

3.178 CVF-178

- **Severity** Major
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraPoolDerivedState.sol

Description The descriptions of these returned values are exactly the same. Most probably one of them is incorrect.
Recommendation Consider fixing the incorrect description.

Listing 178:

```
42 * @return innerSecondsSpentPerLiquidity The snapshot of seconds
     ↪ per liquidity for the range
   * @return innerSecondsSpent The snapshot of seconds per
     ↪ liquidity for the range
```

3.179 CVF-179

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraPoolActions.sol

Description This argument is not documented.
Recommendation Consider documenting.

Listing 179:

```
27 address sender,
```

3.180 CVF-180

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraPoolActions.sol

Description This returned value is not documented.
Recommendation Consider documenting.

Listing 180:

```
38 uint256 liquidityAmount
```

3.181 CVF-181

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraPoolActions.sol

Recommendation Missing "@return".

Listing 181:

```
53 * amount1 The amount of fees collected in token1

71 * amount1 The amount of token1 sent to the recipient

90 * amount1 The delta of the balance of token1 of the pool, exact
   ↪ when negative, minimum when positive

112 * amount1 The delta of the balance of token1 of the pool, exact
   ↪ when negative, minimum when positive
```

3.182 CVF-182

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** IAlgebraPoolActions.sol

Recommendation The name "zeroToOne" would be more clear. Zero for one could be understood as "sell token zero for token one" or "buy token zero for token one".

Listing 182:

```
83 * @param zeroForOne The direction of the swap, true for token0
   ↪ to token1, false for token1 to token0

105 * @param zeroForOne The direction of the swap, true for token0
   ↪ to token1, false for token1 to token0
```

3.183 CVF-183

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** IAlgebraPoolActions.sol

Description The price after a swap is not what a caller most probably care of. What he care is the other token amount.
Recommendation Consider replacing the "limitSqrtPrice" argument with "limitOtherAmount" argument that specifies the minimum other output amount, in case "amountSpecified" is positive, or the maximum input amount, in case "amountSpecified" is negative. Those users who actually do care about the price after a swap could check the price after the swap themselves and revert the transaction in case the price is not what they expected.

Listing 183:

```
85  * @param limitSqrtPrice The Q64.96 sqrt price limit. If zero for
        ↪ one, the price cannot be less than this
    * value after the swap. If one for zero, the price cannot be
        ↪ greater than this value after the swap

107 * @param limitSqrtPrice The Q64.96 sqrt price limit. If zero for
        ↪ one, the price cannot be less than this
    * value after the swap. If one for zero, the price cannot be
        ↪ greater than this value after the swap
```

3.184 CVF-184

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraPoolActions.sol

Description Usually token amounts are represented by "uint256". Using "int256" limits the possible amounts range.
Recommendation Consider using "uint256" for amounts and a separate argument for the swap direction.
Client Comment Considered.

Listing 184:

```
95  int256 amountSpecified,

118 int256 amountSpecified,
```

3.185 CVF-185

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraPoolActions.sol

Description This use case is unclear.
Recommendation Consider elaborating a bit more on it or just remove this comment.
Client Comment Comment enhanced.

Listing 185:

```
126 * @dev Can be used to donate underlying tokens pro-rata to
      ↪ currently in-range liquidity providers by calling
    * with 0 amount{0,1} and sending the donation amount(s) from the
      ↪  callback
```

3.186 CVF-186

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IDataStorageOperator.sol

Recommendation Here "@return" is missing.
Client Comment As intended.

Listing 186:

```
 13 * blockTimestamp The timestamp of the observation,
    * tickCumulative the tick multiplied by seconds elapsed for the
      ↪ life of the pool as of the timepoint timestamp,
    * secondsPerLiquidityCumulative the seconds per in range
      ↪ liquidity for the life of the pool as of the timepoint
      ↪ timestamp,
    * volatilityCumulative Cumulative standard deviation for the
      ↪ life of the pool as of the timepoint timestamp
    * volumePerAvgLiquidity Cumulative swap volume per liquidity for
      ↪  the life of the pool as of the timepoint timestamp
```

3.187 CVF-187

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IDataStorageOperator.sol

Description There is not argument with such name.

Listing 187:

```
17 * volumePerAvgLiquidity Cumulative swap volume per liquidity for
     ↪ the life of the pool as of the timepoint timestamp
```

3.188 CVF-188

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IDataStorageOperator.sol

Description The number format of these returned values is unclear.
Recommendation Consider documenting.

Listing 188:

```
25 int56 tickCumulative,
uint160 secondsPerLiquidityCumulative,
uint88 volatilityCumulative,
int24 averageTick,
uint144 volumePerLiquidityCumulative
```

3.189 CVF-189

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IDataStorageOperator.sol

Description These arguments are not documented.
Recommendation Consider documenting.

Listing 189:

```
28 int24 averageTick,
uint144 volumePerLiquidityCumulative
```

3.190 CVF-190

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IDataStorageOperator.sol

Description The semantics of these functions, their arguments and returned values are unclear.
Recommendation Consider documenting.
Client Comment Descriptions have been added.

Listing 190:

```
34 function getSingleTimepoint(

66 function getAverages(

91 function getFee(
```

3.191 CVF-191

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IDataStorageOperator.sol

Recommendation It would be more efficient to return a single array of structs with four fields, rather than four parallel arrays.

Listing 191:

```
60 int56[] memory tickCumulatives,
   uint160[] memory secondsPerLiquidityCumulatives,
   uint112[] memory volatilityCumulatives,
   uint256[] memory volumePerAvgLiquiditys
```

3.192 CVF-192

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IDataStorageOperator.sol

Description The number format of this argument is unclear.
Recommendation Consider documenting.
Client Comment Value capped by 10000 « 64.

Listing 192:

```
78 uint128 volumePerLiquidity
```

3.193 CVF-193

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** IDataStorageOperator.sol

Description Here argument names have the underscore ("_") prefix, unlike names of other arguments in this file.
Recommendation Consider removing the prefix for consistency.

Listing 193:

```
81 function changeFeeConfiguration(AdaptiveFee.Configuration
     ↪ calldata _feeConfig) external;

92   uint32 _time,
     int24 _tick,
     uint16 _index,
     uint128 _liquidity
```

3.194 CVF-194

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IDataStorageOperator.sol

Description The number format of this argument is unclear.
Recommendation Consider documenting.
Client Comment Pool guarantees that liquidity cannot overflow uint128.

Listing 194:

```
84 uint128 liquidity,
```

3.195 CVF-195

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IDataStorageOperator.sol

Description The number format of the returned value is unclear.
Recommendation Consider documenting.
Client Comment Added mention of the maximum value.

Listing 195:

```
87 ) external pure returns (uint128 volumePerLiquidity);
```

3.196 CVF-196

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** IDataStorageOperator.sol

Description Naming functions IN_UPPER_CASE is uncommon.
Recommendation Consider naming inCamelCase.

Listing 196:

```
89 function WINDOW() external view returns (uint32);
```

3.197 CVF-197

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IDataStorageOperator.sol

Description The semantics of the returned value is unclear.
Recommendation Consider documenting.

Listing 197:

```
89 function WINDOW() external view returns (uint32);
```

3.198 CVF-198

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** IAlgebraVirtualPool.sol

Description This version requirement is inconsistent with version requirements in other interfaces located in the same project.
Recommendation Consider using consistent version requirements across a project.

Listing 198:

```
2 pragma solidity =0.7.6;
```

3.199 CVF-199

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** IAlgebraVirtualPool.sol

Recommendation The name "zeroToOne" would be more clear. Zero for one could be understood as "sell token zero for token one" or "buy token zero for token one".

Listing 199:

```
17 function cross(int24 nextTick, bool zeroForOne) external;
```

3.200 CVF-200

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraVirtualPool.sol

Description The role of this function is unclear and the comment doesn't help.
Recommendation Consider explaining.

Listing 200:

```
19 // This function updates the #prevLiquidity
20 function processSwap() external;
```

3.201 CVF-201

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraVirtualPool.sol

Description The returned value is not documented.
Recommendation Consider documenting.

Listing 201:

```
27 function increaseCumulative(uint32 currentTimestamp) external
   ↪ returns (Status);
```

3.202 CVF-202

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** IAlgebraPoolDeployer.sol

Recommendation Events are usually named via nouns, such as "Factory".

Listing 202:

```
16 event FactoryChanged(address indexed factory, address indexed
   ↪ _factory);
```

3.203 CVF-203

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraPoolDeployer.sol

Recommendation The parameters type should be "IAlgebraFactory".

Listing 203:

```
16 event FactoryChanged(address indexed factory, address indexed
   ↪ _factory);
```

3.204 CVF-204

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** IAlgebraPoolDeployer.sol

Description The old value parameter is redundant as its value could be derived from the previous events.
Recommendation Consider removing it.

Listing 204:

```
16 event FactoryChanged(address indexed factory, address indexed
   ↪ _factory);
```

3.205 CVF-205

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** IAlgebraPoolDeployer.sol

Description The parameter names are very confusing as the differ only in underscore ("_") prefix.
Recommendation Consider renaming to "oldFactory" and "newFactory".

Listing 205:

```
16 event FactoryChanged(address indexed factory, address indexed
   ↪ _factory);
```

3.206 CVF-206

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraPoolDeployer.sol

Recommendation The type of this returned value should be "IAlgebraFactory".

Listing 206:

```
31 address factory,
```

3.207 CVF-207

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraPoolDeployer.sol

Recommendation The type of these returned values should be "IERC20".

Listing 207:

```
32 address token0,
   address token1
```

3.208 CVF-208

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraPoolDeployer.sol

Recommendation The type of this argument should be "IAlgebraFactory".

Listing 208:

```
47 address factory,
```

3.209 CVF-209

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraPoolDeployer.sol

Recommendation The type of these arguments should be "IERC20".

Listing 209:

```
48 address token0,
   address token1
```

3.210 CVF-210

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraPoolDeployer.sol

Recommendation The return type should be "IAlgebraPool".

Listing 210:

```
50 ) external returns (address pool);
```

3.211 CVF-211

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraPoolDeployer.sol

Recommendation The argument type should be "IAlgebraFactory".

Listing 211:

```
56 function setFactory(address factory) external;
```

3.212 CVF-212

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraPool.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 212:

```
22 {

   }
```

3.213 CVF-213

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** IAlgebraFactory.sol

Recommendation Events are usually named via nouns, such as "Owner", "VaultAddress", "Pool" etc.

Listing 213:

```
13 event OwnerChanged(address indexed oldOwner, address indexed
   ↪ newOwner);

20 event VaultAddressChanged(address indexed vaultAddress, address
   ↪ indexed _vaultAddress);

28 event PoolCreated(address indexed token0, address indexed token1
   ↪ , address pool);

35 event FarmingAddressChanged(address indexed farmingAddress,
   ↪ address indexed _farmingAddress);
```

3.214 CVF-214

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** IAlgebraFactory.sol

Description The old value parameters are redundant, as their values could be derived from the previous events.

Listing 214:

```
13 event OwnerChanged(address indexed oldOwner, address indexed
   ↪ newOwner);

20 event VaultAddressChanged(address indexed vaultAddress, address
   ↪ indexed _vaultAddress);

35 event FarmingAddressChanged(address indexed farmingAddress,
   ↪ address indexed _farmingAddress);
```

3.215 CVF-215

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** IAlgebraFactory.sol

Description The parameter names are very confusing, as they differ only in underscore ("_") prefix.
Recommendation Consider renaming to "oldVaultAddress"/"newVaultAddress" and "oldFarmingAddress"/"newFarmingAddress" for consistency with the parameter names in the "OwnerChange" event.

Listing 215:

```
20 event VaultAddressChanged(address indexed vaultAddress, address
   ↪ indexed _vaultAddress);

35 event FarmingAddressChanged(address indexed farmingAddress,
   ↪ address indexed _farmingAddress);
```

3.216 CVF-216

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraFactory.sol

Recommendation The type of the "pool" parameter should be "IAlgebraPool".

Listing 216:

```
28 event PoolCreated(address indexed token0, address indexed token1
   ↪ , address pool);
```

3.217 CVF-217

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraFactory.sol

Recommendation The type of the "token0" and "token1" parameters should be "IERC20".

Listing 217:

```
28 event PoolCreated(address indexed token0, address indexed token1
   ↪ , address pool);
```

3.218 CVF-218

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraFactory.sol

Recommendation The arguments type should be "IERC20".

Listing 218:

```
66 function poolByPair(address tokenA, address tokenB) external
   ↪ view returns (address pool);
```

3.219 CVF-219

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraFactory.sol

Recommendation The return type should be "IAlgebraPool".

Listing 219:

```
66 function poolByPair(address tokenA, address tokenB) external
   ↪ view returns (address pool);
```

3.220 CVF-220

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraFactory.sol

Recommendation The arguments type should be "IERC20".

Listing 220:

```
77 function createPool(address tokenA, address tokenB) external
   ↪ returns (address pool);
```

3.221 CVF-221

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraFactory.sol

Recommendation The return type should be "IAlgebraPool".

Listing 221:

```
77 function createPool(address tokenA, address tokenB) external
   ↪ returns (address pool);
```

3.222 CVF-222

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraFactory.sol

Description The semantics and the number format of the arguments is unclear.
Recommendation Consider documenting.
Client Comment Description added.

Listing 222:

```
99  uint32 alpha1,
100 uint32 alpha2,
    uint32 beta1,
    uint32 beta2,
    uint16 gamma1,
    uint16 gamma2,
    uint32 volumeBeta,
    uint32 volumeGamma,
    uint16 baseFee
```

3.223 CVF-223

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** IAlgebraSwapCallback.sol

Description The order of arguments in a documentation comment differs from the order of the same arguments in the corresponding function declaration.
Recommendation Consider using the same order of arguments.
Client Comment Argument feeAmount removed.

Listing 223:

```
15 /// @param data Any data passed through by the caller via the
      ↪ IAlgebraPoolActions#swap call
   /// @param feeAmount fee amount

20    uint256 feeAmount,
      bytes calldata data
```

3.224 CVF-224

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IAlgebraSwapCallback.sol

Description It is unclear whether what token this amount is denominated in and whether this amount ought to be sent to the pool inside the callback.
Recommendation Consider clarifying.
Client Comment Argument feeAmount removed.

Listing 224:

```
16 /// @param feeAmount fee amount
```

3.225 CVF-225

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** IAlgebraSwapCallback.sol

Recommendation Function names usually start with lowercase letters.

Listing 225:

```
17 function AlgebraSwapCallback(
```

3.226 CVF-226

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IAlgebraSwapCallback.sol

Description Usually token amounts are represented by "uint256". Using "int256" limits the possible amounts range.
Recommendation Consider using "uint256" for amounts and a separate argument for the swap direction.
Client Comment Considered.

Listing 226:

```
18 int256 amount0Delta,
   int256 amount1Delta,
```

3.227 CVF-227

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** IAlgebraMintCallback.sol

Recommendation Function names usually start with lowercase letters.

Listing 227:

```
13 function AlgebraMintCallback(
```

3.228 CVF-228

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** IAlgebraFlashCallback.sol

Description There is not easy way to know the exact token amounts ought to be repayed.
Recommendation Consider passing them as arguments.
Client Comment Considered for the future.

Listing 228:

```
11 * @dev In the implementation you must repay the pool the tokens
   ↪ sent by flash plus the computed fee amounts.
```

3.229 CVF-229

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** IAlgebraFlashCallback.sol

Recommendation Function names usually start with lowercase letters.

Listing 229:

```
17 function AlgebraFlashCallback(
```