Mehwish Khan
1032233881
K2

Page No.
Date / /

Practice Questions

Data Structures

Q1 Sort using bubble sort: 64, 34, 25, 12, 22, 11, 90. Show step by step execution for all pases highlighting "swap" and "No Swap" situations. How many passes required to sort an array of N elements using bubble sort?

Initial Array: 64, 34, 25, 12, 22, 11, 90
Working - Repeatedly swapping adj elements if they are in wrong order.

⇒ Pass 1:

Compare 64 and 34 → Swap (64 > ~~64~~ 34)
Array: 34, 64, 25, 12, 22, 11, 90

Similarly; (64 > 25) → 34, 25, 64, 12, 22, 11, 90
Swap

(64 > 12) → 34, 25, 12, 64, 22, 11, 90
Swap

(64 > 22) → 34, 25, 12, 22, 64, 11, 90
Swap

(64 > 11) → 34, 25, 12, 22, 11, 64, 90
Swap

(64 < 90) → 34, 25, 12, 22, 11, 64, 90
No swap

⇒ Pass 2:

(34 > 25) → 25, 34, 12, 22, 11, 64, 90
Swap

(34 > 12) → 25, 12, 34, 22, 11, 64, 90
Swap

(34 > 22) → 25, 12, 22, 34, 11, 64, 90
swap

(34 > 11) → 25, 12, 22, 11, 34, 64, 90
Swap

(34 < 64) → 25, 12, 22, 11, 34, 64, 90
No swap

⇒ Pass 3:

(25 > 12) → 12, 25, 22, 11, 34, 64, 90
Swap

(25 > 22) → 12, 22, 25, 11, 34, 64, 90
Swap

(25 > 11) → 12, 22, 11, 25, 34, 64, 90
Swap

(25 < 34) → 12, 22, 11, 25, 34, 64, 90
No swap

⇒ Pass 4:

(12 < 22) → 12, 22, 11, 25, 34, 64, 90
No swap

(22 > 11) → 12, 11, 22, 25, 34, 64, 90
Swap

(22 < 25) → 12, 11, 22, 25, 34, 64, 90
No swap

⇒ Pass 5:

(12 > 11) → 11, 12, 22, 25, 34, 64, 90
Swap

(12 < 22) → 11, 12, 22, 25, 34, 64, 90
No swap

Final sorted array: 11, 12, 22, 25, 34, 64, 90

⇒ For an array of N elements, worst-case no. of passes is N-1. Because in each pass, the largest unsorted element moves to its correct position.

Page No.
Date

Q2) Sort using selection sort: 50, 23, 03, 18, 9, 01, 70, 21, 20, 6, 40, 04. What is best & worst case time complexity of selection sort?

50, 23, 03, 18, 9, 01, 70, 21, 20, 6, 40, 4

Working: Repeatedly finding min element from the unsorted portion & swapping with first unsorted element.

Pass-1: min element = 01
Swap with first element (50)
01, 23, 03, 18, 09, 50, 70, 21, 20, 06, 40, 04

Pass 2: min element = 03
Swap with 23
01, 03, 23, 18, 09, 50, 70, 21, 20, 06, 40, 04

Pass 3: min ele = 04
Swap with 23
01, 03, 04, 18, 09, 50, 70, 21, 20, 06, 40, 23

Pass 4: min ele = 06
Swap with 18
01, 03, 04, 06, 09, 50, 70, 21, 20, 18, 40, 23

Pass 5: min ele = 09
No swap needed

Pass 6: min ele = 18
Swap with 50
01, 03, 04, 06, 09, 18, 70, 21, 20, 50, 40, 23

Page No.
Date

Pass 7: min ele = 20
Swap with 70
01, 03, 04, 06, 09, 18, 20, 21, 70, 50, 40, 23

Pass 8: min ele = 21
No swap needed

Pass 9: min ele = 23
Swap with 70
01, 03, 04, 06, 09, 18, 20, 21, 23, 50, 40, 70

Pass 10: min ele = 40
Swap with 50
01, 03, 04, 06, 09, 18, 20, 21, 23, 40, 50, 70

Required sorted array

Pass 11: min ele = 50
No swap

Time Complexity:

- Best case : $O(N^2)$
- Worst case : $O(N^2)$

* For N elements, always the pass required = (N-1)

Page No.
Date / /

3Q. Sort using insertion sort: 55, 85, 45, 11, 34, 5, 89, 99, 67

⇒ ~~Soln:~~ Working: Picks an element and places it in the correct position relative to the elements that are already sorted.

Pass 1: (Key = 85),
compare with 55 (85 > 55)
No swap

Pass 2: (Key = 45),
compare with 85 & 55, (45 < 85), (45 < 55)
swap swap

45, 55, 85, 11, 34, 5, 89, 99, 67

Pass 3: Key = 11
(11 < 85), (11 < 55), (11 < 45)
swap swap swap

11, 45, 55, 85, 34, 5, 89, 99, 67

Pass 4: Key = 34
(34 < 85), (34 < 55), (34 < 45), (34 > 11)
swap swap swap No swap

11, 34, 45, 55, 85, 5, 89, 99, 67

Pass 5: Key = 5
(5 < 85), (5 < 55), (5 < 45), (5 < 34), (5 < 11)
Swap

5, 11, 34, 45, 55, 85, 89, 99, 67

Pass 6: Key = 89
(89 > 85)
No swap

Page No.
Date

Pass 7: Key = 99

(99 > 89)

No Swap

Pass 8: Key = 67

(67 < 99), (67 < 89), (67 < 85), (67 > 55)

Swap Swap Swap No Swap

5, 11, 34, 45, 55, 67, 85, 89, 99

Time Complexity:

- Best Case = $O(N)$
- Worst Case - $O(N^2)$

Q4) Write rules to convert given infix to postfix & infix to prefix
(P * Q - (L + M * N) ^ (X * Y / Z).

Rules: For Postfix

1) Operands (letters / numbers) : Add directly to postfix

2) Left Parenthesis '(' : push it into stack.

3) Right Parenthesis ')' : pop operators from the stack & ~~ado~~ add them to postfix expression, until left parenthesis comes. Dicard both parethesis.

4) Operators (+, -, *, /, ^):

- pop operators from stack with greater or equal precedence and add them to postfix expression.
- push current operator onto the stack.

5) End expression: pop all ~~operati~~ operators from the stack and add them to the postfix expression.

Page No.
Date

Operator precedence : 1) ^ : Highest precedence (right - associative)
2) *, / : } left associative
3) +, - : }

(P * Q - (L+M * N) ^ (X * Y / Z))

	Postfix	Stack
(		(
P	P	(
*	P	(, *
Q	P, Q	(, *
-	P, Q, *	(, -
(	P, Q, *	(, -, (
L	P, Q, *, L	(, -, (
+	P, Q, *, L	(, -, (, +
M	P, Q, *, L, M	(, -, (, +
*	P Q * L M	(- (+ *
N	P Q * L M N	(- (+ *
)	P Q * L M N * +	(-
^	P Q * L M N * +	(- ^
(	P Q * L M N * +	(- ^ (
X	P Q * L M N * + X	(- ^ (
*	P Q * L M N * + X	(- ^ (*
Y	P Q * L M N * + X Y	(- ^ (*
/	P Q * L M N * + X Y *	~~(- ^ (*~~ (- ^ (/
Z	P Q * L M N * + X Y * Z	(- ^ (/
)	P Q * L M N * + X Y * Z /	(- ^ (
)	P Q * L M N * + X Y * Z / ^ -	-

→ Final expression

* Time & Space Complexity → O(N)

Rules : For prefix

1) Reverse the infix expression
2) Replace every) with (and vice versa.
3) Use postfix conversion rules on reversed expression
4) Reverse the postfix expression to get prefix.

(P * Q - (L + M * N) ^ (X * Y / Z))

Reversed : ~~( ) Z / Y * X~~

~~) Z / Y * X (^) N * M + L (- Q * P (~~

((Z / Y * X) ^ (N * M + L) - Q * P)

	Postfix	Stack
(	-	((
(	-	((
Z	Z	(((
/	~~Z Y~~ Z	((/
Y	Z Y	((/
*	Z Y /	((*
X	Z Y / X	((*
)	Z Y / X *	~~(~~ (
^	Z Y / X *	(^
(	Z Y / X *	(^(
~~BA~~ N	Z Y / X * N	(^(
*	Z Y / X * N	(^(*
M	Z Y / X * N M	(^(*
+	Z Y / X * N M *	(^(+
L	Z Y / X * N M * L	(^(+
)	Z Y / X * N M * L +	(^
-	Z Y / X * N M * L +	(^ -
Q	Z Y / X * N M * L + Q	(^ -
*	Z Y / X * N M * L + Q	(^ - *
P	Z Y / X * N M * L + Q P	(^ - *
)	Z Y / X * N M * L + Q P * - ^	-

Page No
Date / /

Final postfix expression:

ZY/X*NM*L+QP*-^

prefix expression:

^-*PQ+L*MN*X/YZ

Qs) Draw & explain circular queue using array. Write pseudocode for Add, Remove, Is full, Is Empty.

- Initialize an array queue of size n,
- Initialize two variables front & rear to -1.
- Enqueue: To enqueue an element x into the queue,
  - increment rear by 1.
  - If rear is n, set rear to 0.
  - If front is -1, set front to 0.
  - Set queue(rear) to x.
- Dequeue:
  - Check if the queue is empty by checking if front is -1.
  - If it is empty, return error message indicating queue is empty.
  - Set x to queue front.
  - If front = rear, set front & rear to -1
  - Otherwise, increment front by 1
  - If front = n, set front = 0
  - Return x.

Page No.
Date

Pseudo code :

⇒ Add Q : initially front = rear = 0

```
Algorithm AddCQ (elem)
{
  if (rear+1) % n == full
    print " queue full "
  else
    {
      rear = (rear+1) % n
      Q[rear] = elem
    }
}
```

⇒ Delete Q :- initially front = rear = 0

```
Algorithm DelCQ [elem]
{
  if front == rear
    print " queue empty "
  else {
      front = (front+1) % n
    elem = Q (front)
      return elem
    }
}
```

⇒ Is Empty :

```
is Empty (elem)
{ if (front == rear)
    return 1
  else {
      return 0
    }
```

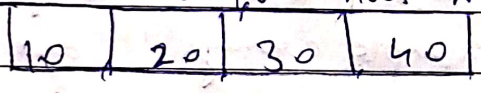
=> Is Full : isFull (elem)

```
{ if ((rear+1) % size == front)
      return 1
  else {
      return 0
  }
}
```

Q What is double ended queue? Draw diagram with labelling 4 basic operations at appropriate places. Which 2 data structures ~~& con~~ are combined in it & how?

=> A double ended queue [deque] is an abstract data type that generalizes a queue, for which elements can be added to or removed from either the front or rear.

-> Lets assume deque has max size = 5,



- Push Rear () - Inserts an element ~~before 10~~ after 40.
- Push front () - Inserts an element before 10.
- Pop Rear () - Removes 40.
- Pop Front () - Removes 10.

-> Queue & Stack are data structures involved in Deque

- Queue - FIFO functionality is used
- Stack - ability to insert & remove elements from both ends.

Page No.
Date

Q. Diff b/w Linear & circular queue.

Features:	Linear Queue	Circular Queue
Structure	Elements stored in linear manner	" circular manner
Overflow	Can experience wasted space as elements are dequeued	Utilizes space efficiently by wrapping around.
Insertion	Happens at rear end only.	Can happen at both rear & front.
Deletion	Happens at front end only	Can happen "
Implementation	Easier to implement, with simple arrays	Slightly more compl due to circular nature of pointers
Size Limitations	Fixed size, leading to overflow, even if space available	Fixed size, but can reuse space efficiently, allowing for continuous operations w/o overflow.

Q. Pseudocode of Bubble sort, Selection sort & Insertion sort. & no shell sort.

⇒ Bubble Sort:

```
Algorithm bubble()
{
  for i = 1 to n
  {
    for j=0 to n-i-1
    { if a[j] > a[j+1]
      { swap(a[j], a[j+1])
      }
    }
    display(a,n); }
```

⇒ Selection sort:

```
Void Selection sort()
{ for i=0 to n-2
  {
    min pos = i;
    for j = i+1 to n-1
    {
      if a[j] < a[min pos]
      { min pos = j;
      }
    }
      if (min pos != i)
    {
      temp = a[i]; a[i] = a[min pos]; a[min pos] = temp;
    }}
```

Page No.
Date

⇒ Insertion sort:

```
Algorithm insertion Sort (arr[], n)
{ for i=1 to i<n step 1
{ key = arr[i];
  j = i-1;
while (j >=0 ff arr[j] > key)
{ arr[j+1] = arr[j];
  j = j-1;
  }
  arr[j+1] = key;
 }
}
```

⇒ Shell sort:

```
void shell-sort (int A[], int n)
{  gap = n/2
  do
{  do
   {  swapped = 0
   for (i=0 ; i<n-gap ; i++)
   if (A[i] > A[i+gap])
   {
      swap();
      swapped = 1;
   }
}
   while (swapped == 1);
} while ((gap = gap/2) >=1);
}
```

Page No.
Date

8. write pseudocode for basic stack operations:

- isFull():

```
Algorithm isFull()
{
    if (top == MAXSIZE-1)
        return true;
    else
        return false;
}
```

- isEmpty():

```
{
    if (top == -1)
        return true;
    else
        return false;
}
```

- push():

```
{
    if (! isFull())
    {
        top = top+1
        stack [top] = item;
    }
}
```

- pop():

```
pop():
    {
     if (! isEmpty())
     {
      temp = stack[top];
      top = top - 1;
      return (temp);
     }
    }
```

Q. What is time complexity? Why asymptotic notations are imp? Find time complexity of following code using step count method.

```
int x = 0;
for (i=0; i<N; i++)
{
  for (j=N; j>i; j++)
  {   x = x+i+j;
  }
}
```

→ Time Complexity : Amount of time an algorithm takes to run as a function of the size of its input.

- It helps in finding how the runtime of an algorithm increases as the size of input increases, enabling to choose the most efficient algorithm for given problem.

- Asymptotic notations are useful as,

① They simplify analysis of algorithm.
② Compares efficiency of diff algorithm.
③ Predicts how an algorithm's performance will change as

the size of input grows.

④ Guide developers in selecting right algorithm.

Q. ~~Explain~~ Differentiate Linear & Non linear DS and Static & Dynamic DS:

Features	Linear	Non-Linear
Definition	Data elements are arranged sequentially.	" arranged hierarchially
Traversal	Traversed in single level from start to end.	Traversal can be done in multiple ways.
Memory usage	Memory is utilized efficiently due to contiguous arrangement	Memory utilization less efficient, as elements stored in non-contiguous locations
Ex:	Arrays, Linked list, Stacks, Queues	Ex: Trees, Graphs.

Features:	Static Data Structure	Dynamic Data Structure
Definition	Size of DS is fixed & cant be changed during execution	Size of DS can grow or shrink during execution.
Memory Allocation	Memory allocated at compile time.	Memory allocated at runtime.
Flexibility	Less flexible as size is pre-determined	Highly flexible as the size can adjust based on requirement
Usage:	Useful when the size of data is known in advance	Useful when the size of data is unpredictable or varies with time
Ex:	Arrays	Linked lists, Dynamic Arrays (like malloc in C)

Q What is need to convert infix exp to postfix?

Infix exp (A+B) is converted to postfix (AB+) as, postfix notation is easier for computers to evaluate w/o the need for parentheses to dictate precedence.

Q: What is row major & column major representation? with eg. Write formula to find any element A[i][j] in row major & column major representation of A.

- Row major - Stores all elements of a row contiguously in memory before moving to the next row.
- Column major - Stores all elements of column contiguously in memory before moving to next column.

A[i][j]

→ Row major representation:

Address of A[i][j] = Base address of A + (i x n + j) x size of element

n = no. of column

i = Row index

j = Column index

→ Column major representation:

Address of A[i][j] = Base Address of A + (j x m + i) x size of element

m = no. of rows in array

Ex: 2D array: A[3][4]

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

For row major

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

→ Calculating memory location of A[2][3]
(i, j)

A[2][3] = Base address of A + (2×4 + 3) × size of element

→ For column major

{1, 5, 9, 2, 6, 10, 3, 7, 11, 4, 8, 12}

A[2][3] = Base address of A + (3×3 + 2) × size of element.

Q. What is sparse matrix? List applications. Represent it with suitable DS. Also write algorithm for compact form

→ Any m*n matrix which contains large no. of zeroes is called sparse matrix.

- We can characterize uniquely any element within a matrix by using an array of triples to represent a sparse matrix.
- An array of triples → also called compact form.

Ex:

sparse matrix

Row	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5
Row 0	15	0	0	22	0	-15
A[1]	0	11	3	0	0	0
A[2]	0	0	0	-6	0	0
A[3]	0	0	0	0	0	0
A[4]	91	0	0	0	0	0
A[5]	0	0	28	0	0	0

Non zero elements = 8

↓ Compact form

	Col 0	Col 1	Col 2
Row[0]	6 rows	6 columns	8 → Non zero elements
B[1]	0	0	15
B[2]	0	3	22
B[3]	0	5	-15
B[4]	1	1	11
B[5]	1	2	3
B[6]	2	3	-6
B[7]	4	0	91
B[8]	5	2	28

* Rows in compact form = (non zero element in sparse matrix) + 1

* Columns = 3

→ Applications:

① Computational fluid dynamics → Scientific computing

② Graph theory – adjacency matrices of graphs are sparse

③ Machine Learning – Natural language processing.

④ Image compression – Algorithms like JPEG compression.

⑤ Signal Processing – Communication & Audio processing.

```
Algorithm compact (A, m, n, B)

    B(0,0) = m;
    B(0,1) = n;
    K = 1;
    for i=0 to m
    {
      for j=0 to n
      {
        if (A(i,j) != 0)
        {   B(k,0) = i;
            B(k,1) = j;
            B(k,2) = A(i,j);
            K++;
        }
      }
    }
    B(0,2) = K-1;
}
```

Q. Explain Simple & Fast Transpose with Ex. Write its algorithm.

- Transpose → Swapping rows & columns.
  A[i][j] → A[j][i]

→ Simple Transpose: Directly swap rows & columns indices of each non-zero element. Time Complexity: O(n*t)

Ex:

$$A = \begin{bmatrix} 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 6 \\ 0 & 9 & 0 & 0 \end{bmatrix} = \begin{array}{c} \\ A_0 \\ A_1 \\ A_2 \\ A_3 \end{array} \begin{array}{c} \begin{matrix} C_0 & C_1 & C_2 & C_3 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 6 \\ 0 & 9 & 0 & 0 \end{bmatrix} \end{array}$$

n → columns, t → elements

	C0	C1	C2
ROW 0	4	4	4
1	0	2	3
2	2	0	5
3	2	3	6
4	3	1	9
5			

	C0	C1	C2
B0	4	4	4
→ B1	0	2	5
B2	1	3	9
B3	2	0	3
B4	3	2	6

Algorithm Transpose (A, B)

```
1)  (m, n, t) := (A(0,0), A(0,1), A(0,2))
2)  (B(0,0), B(0,1), B(0,2)) := (n, m, t)
3)  if t <= 0 then return
4)    q := 1
5)    for col := 0 to n do
6)    for p := 1 to t do
7)    if A(p,1) = col then
8)  {(B(q,0), B(q,1), B(q,2) := (A(p,0), (A(p,2))
9)    q := q+1 }
10)   end
11)  end
12)  end Transpose
```

→ Fast Transpose :

Time complexity : O(n+t)

	C0	C1	C2	C3
A0	0	0	3	0

→

	C0	C1	C2
B0	4	4	4
B1	0	2	5

4	3	1	9
5			

B4	3	2	6

Algorithm Transpose (A, B)

```
1)  (m, n, t) := (A(0,0), A(0,1), A(0,2))
2)  (B(0,0), B(0,1), B(0,2)) := (n, m, t)
3)  if t <= 0 then return
4)    q := 1
5)    for col := 0 to n do
6)    for p := 1 to t do
7)    if A(p,1) = col then
8)  {(B(q,0), B(q,1), B(q,2) := (A(p,0), (A(p,2))
9)    q := q+1 }
10)   end
11)  end
12)  end Transpose
```

Page No.
Date / /

Fast Transpose

Ex:

	0	1	2	3
A0	0	0	3	0
1	0	0	0	0
2	5	0	0	6
3	0	9	0	0

→

	0	1	2
0	4	4	4
1	0	2	3
2	2	0	5
3	2	3	6
4	3	1	9

r term

0	1	2	3
1	1	1	1

r pos

0	1	2	3	4
X	2	3	4	5
2	3	4	5	

	0	1	2
0	4	4	4
1	0	2	5
2	1	3	9
3	2	0	3
4	3	2	6

B_3	0	2	3
B_4	2	3	6

Index =

0	1
1	2
2	3 4
3	4
4	5

Algorithm Fast_Transpose (A, B)
declare S(n), T(n);
(m, n, t) := (A(0,0), A(0,1), A(0,2))
(B(0,0), B(0,1), B(0,2), := (n, m, t)
if t ≤ 0 then return
for i := 0 to n do S(i) := 0 end
for ~~i := 0~~ i := 1 to t do
S(A(i,1)) := S(A(i,1)) + 1
end
T(0) := 1
for i := 1 to n do
T(i) = T(i-1) + S(i-1)
end
for i := 1 to t do
j := A(i, 1)
(B(T(j), 0), B(T(j), 1), B(T(j), 2)) := (A(i,1), A(i,0), A(i,2)
T(j) = T(j) + 1
end
end Fast_Transpose

Page No.
Date

Q. Explain addition of two sparse matrix with Ex. & write its algorithm.

→ 1) Obtain the triplet form of both sparse matrix.
2) Create new triplet to store result as result matrix
3) Copy no. of rows & columns from any sparse matrix to result matrix.
4) Let i, j, k be the indices of sparse matrices 1, 2, 3
5) Initialize i, j, k to 1.
6) Traverse both matrices from 2nd row.

Ex: ~~Assu~~ Let there be two sparse matrix in triplet form.

	0	1	2	
0	2	3	4	
1	0	0	5	i
2	0	1	6	i +
3	1	0	8	i
4	1	1	5	i

end

	0	1	2	
0	2	3	4	
1	0	1	4	j
2	0	2	3	j
3	1	1	3	j
4	1	2	6	j

Smat 1

i = 1 2 3 4 5

Smat 2

j = 1 2 3 4 5

k = 1 2 3 4 5 6 7

Resultant Mat =

	Row	col	val	
0	2	3	6	→ k-1
1	0	0	5	
2	0	1	10	
3	0	2	3	
4	1	0	8	
5	1	1	8	
	1	2	6	

Algorithm Addition (a, b, c)

if a(0,0) = b(0,0) and a(0,1) = b(0,1)
if t1=0 and t2 = 0 then stop
~~if t1 = 0 and t2 = 0~~
c(0,0) = a(0,0) ; c(0,1) = a(0,1) ;
End
i = j = k = 1
while (i <= t1 and j <= t2)
case A: a(i,0) = b(j,0) // row position
case 1: a(i,1) = b(j,1) // col pos
temp = a(i,2) + b(j,2)
if temp != 0
c(k,0) = a(i,0)
c(k,1) = a(i,1)
c(k,2) = temp
k = k+1
end if
i = i+1 ; j = j+1 ;
case 2 : if a(i,1) < b(j,1)
c(k,0) = a(i,0)
c(k,1) = a(i,1)
c(k,2) = a(i,2)
k = k+1
i = i+1
case 3: if a(i,1) > b(j,1)
c(k,0) = b(j,0)
c(k,1) = b(j,1)
c(k,2) = b(j,2)
k = k+1
j = j+1

Page No.
Date

Case B: if a(i,0) < b(j,0)

c(k,0) = a(i,0)

c(k,1) = a(i,1)

c(k,2) = a(i,2)

k = k+1

i = i+1

Case C: if a(i,0) > b(j,0)

c(k,0) = b(j,0)

c(k,1) = b(j,1)

c(k,2) = b(j,2)

k = k+1

j = j+1

end while

while (i ≤ t1)

{

c := a

i++ ; k++

}

while (j ≤ t2)

{

c := b

j++ ; k++ ;

}

c(0,2) = k-1

End Addition.

Q. Explain concept of ordered List, Show its representation using array.

- Collection of elements arranged in specific sequence
- Each element has defined positions based on some ordering criteria.
- ~~Order~~ The List allows duplicates, or enforce unique elements
- Insertion or deletion migh require, reordering the list.

Adv :

1) Efficient access : O(1) time complexity
2) Binary search : Fast searching algorithms can be applied
3) Simple Implementation : use of array.

Dis Adv :

1) Insertion & Deletion of an element can take O(n) time
2) If array of fixed size, it may run out of size, space requiring resizing or reallocating memory.
3) Contiguous Memory allocation can be ineffecient for very large ordered lists.

→ Ordering can be :

- Ascending order, descending order
- Lexicographical order : For strings (alphabetically)

Representation using array :

= [2, 5, 8, 12, 16]

Q. Explain with Ex how single variable polynomial can be represented using 1-D array? What is adv & dis-adv of this representation?

Polynomial is one of the classic eg: of ordered List.

→ If polynomial has m non-zero terms then the poly can be represented as ordered list of 3m+1 terms.

m, power of x, power of y, coeff, power of x, power of y, coeff

1st term 2nd term

Ex: $x^2 + 5xy + y^2 - y - x$

$(5, (2,0,1), (1,1,5), (0,2,1), (0,1,-1), (1,0,-1))$

OR

5	2	0	1	1	1	5	0	2	1	0	1	-1	1	0	-1

array representation.

Similarly for (x, y, z)

Ex: $-8x^3y^3z^3 + 10x^3y^2z + 5xyz^2 + 10$

4	3	3	3	-8	3	2	1	10	1	1	2	5	0	0	0	10

Adv of this representation:

- Simple & Easy to implement
- Easily access any coeff based on power of x.
- Compact & Efficient representation.
- Adding, Subtracting, multiplying poly is easy when represented as arrays.

Page No.
Date

Disadvantage:

- Wasted space for sparse poly
- Fixed size
- Inefficient for poly with large gap b/w non-zero terms.
- Complexity in Handling large degree poly.

Q. Write algorithm for polynomial addition & multiplicat

```
Algorithm polyadd (p1, p2, max1, max2)
{   i = j = k = 0 ;
{   while (i < max1 && j < max2)
{
   if  p1 [i] . exp > p2 [j] . exp
{   p3 [k] = p1 [i] ;
     k++ ; i++
}
   else if ( p1 [i] . exp < p2 [j] . exp)
{
     p3 [k] = p2 [j] ;
     k++ ;  j++ ;
}
   else // same components
{   temp = p1 [i] . coeff + p2 [j] . coeff
   if (temp ! = 0)
{   p3 [k] . exp = p1 [i] . exp ;
     p3 [k] . coeff = temp ;
     i++ ;  j++ ;  k++ ;
}
} ,
```

```
while (i < max1)
{
    p3[k] = p1[i];
    k++;
    i++;
}
while (j < max2)
{
    p3[k] = p2[j];
    k++;
    j++;
}
```

=> Algorithm polymultiplication (p1, p2, max1, max2)

```
{   i = j = k = 0;
    while (i < max1) {
    j = 0;
    while (j < max2)
    {   temp = p1[i].coff * p2[j].coff;
        if (temp != 0)
    {   flag = 0;
        exp = p1[i].exp + p2[j].exp;
    for (x = 0; x < k; x++) {
    if (exp == p3[x].exp) {
    flag = 1;
    break; }
    }
    if (flag == 1) {
    p3[x].coff = p3[x].coff + temp;
    j++;
    }
```

```
else {
    p3 [k]. exp = exp;
    p3 [k]. coeff = temp;
      j++ ; k++;
  }
 } // end if
} // end while
  i++;
} // end while.
```

Q. State applications of Queues & Stacks.

Queues:

1) Job scheduling - manage job execution such as printers where jobs are executed in the order they are received.
2) Data streaming - Used in networks to manage incoming & outgoing data packets.
3) • while streaming video or audio, queues buffer data to ensure continuous playback.
4) Task scheduling - Useful in systems that need to handle real-time task with diff priorities.
5) Paging in OS - Used in page replacement algorithms

Stacks:

1) Backtracking Algorithms — Depth first search [DFS] in graphs & mazes. When you hit a dead end, you pop from the stack & backtrack to the previous decision point.

2) Expression Evaluation - Infix to postfix & prefix conversion.

3) Function Call Management - Each function call is pushed onto the stack, & when the function returns, it is popped from the stack.

4) String reversal - Reverse a string by pushing each character onto stack & then popping them.

5) Browser History Management : Browser use stacks to manage the history of web pages.

Q. Each element of an array Data [20][50] requires 4 bytes of storage. Base Add = 2000, Find location of Data [10][10] when array is stored as
i) row major ii) column major.

i) Row - Major Order :

Address (A[10][10]) = 2000 + [(10 x 50) + 10] x 4
= 2000 + [510 x 4]
= 4040

ii) Column - Major

= 2000 + [(10 x 20) + 10] x 4
= 2840

Q. Consider linear array A[5:50]. BA = 300 & no. of words per memory cell = 4 bytes. Find addr of A[15]

→ $A[15] = 300 + (15 - 5) \times 4 = \underline{340}$

(5 → lower bound of array considered)

Q. Consider int arr[5][4]. BA = 510. Find address of arr[3][2] with row & column method.

→ Row Major:

$A[3][2] = 510 + [(3 \times 4) + 2] \times 4$

$= 566$

→ Column Major:

$A[3][2] = 510 + [(2 \times 5) + 3] \times 4$

$= 562$