ALGORITHM DESIGN TECHNIQUES

What is an algorithm?
An Algorithm is a procedure to solve a particular problem in a finite number of steps for a finite-sized input.

Classification of various algorithm design techniques

The algorithms can be classified in various ways. They are:

i) Implementation Method
ii) Design Method
iii) Design Approaches
iv) Other Classifications

i) Classification by Implementation Method:

There are primarily three main categories into which an algorithm can be named in this type of classification. They are:

1. **Recursion or Iteration:** A recursive algorithm is an algorithm which calls itself again and again until a base condition is achieved whereas iterative algorithms use loops and/or data structures like stacks, queues to solve any problem. Every recursive solution can be implemented as an iterative solution and vice versa.

   Example: The Tower of Hanoi is implemented in a recursive fashion while Stock Span problem is implemented iteratively.

2. **Exact or Approximate:** Algorithms that are capable of finding an optimal solution for any problem are known as the exact algorithm. For all those problems, where it is not possible to find the most optimized solution, an approximation algorithm is used. Approximate algorithms are the type of algorithms that find the result as an average outcome of sub outcomes to a problem.

   Example: For NP-Hard Problems, approximation algorithms are used. Sorting algorithms are the exact algorithms.

3. **Serial or Parallel or Distributed Algorithms:** In serial algorithms, one instruction is executed at a time while parallel algorithms are those in which

we divide the problem into sub-problems and execute them on different processors. If parallel algorithms are distributed on different machines, then they are known as distributed algorithms.

ii) Classification by Design Method:

There are primarily three main categories into which an algorithm can be named in this type of classification. They are:

1. **Greedy Method:** In the greedy method, at each step, a decision is made to choose the *local optimum*, without thinking about the future consequences.

   Example: Fractional Knapsack, Activity Selection.

2. **Divide and Conquer:** The Divide and Conquer strategy involves dividing the problem into sub-problem, recursively solving them, and then recombining them for the final answer.

   Example: Merge sort, Quicksort.

3. **Dynamic Programming:** The approach of Dynamic programming is similar to divide and conquer. The difference is that whenever we have recursive function calls with the same result, instead of calling them again we try to store the result in a data structure in the form of a table and retrieve the results from the table. Thus, the overall time complexity is reduced. "Dynamic" means we dynamically decide, whether to call a function or retrieve values from the table.

   Example: 0-1 Knapsack, subset-sum problem.

4. **Linear Programming:** In Linear Programming, there are inequalities in terms of inputs and maximizing or minimizing some linear functions of inputs.

   Example: Maximum flow of Directed Graph

5. **Reduction (Transform and Conquer):** In this method, we solve a difficult problem by transforming it into a known problem for which we have an optimal solution. Basically, the goal is to find a reducing algorithm whose complexity is not dominated by the resulting reduced algorithms.

Example: Selection algorithm for finding the median in a list involves first sorting the list and then finding out the middle element in the sorted list. These techniques are also called *transform and conquer.*

6. **Backtracking:** This technique is very useful in solving combinatorial problems that have *a single unique solution*. Where we have to find the correct combination of steps that lead to fulfillment of the task. Such problems have multiple stages and there are multiple options at each stage. This approach is based on exploring each available option at every stage one-by-one. While exploring an option if a point is reached that doesn't seem to lead to the solution, the program control backtracks one step, and starts exploring the next option. In this way, the program explores all possible course of actions and finds the route that leads to the solution.

   Example: N-queen problem, maize problem.

7. **Branch and Bound:** This technique is very useful in solving combinatorial optimization problem that have *multiple solutions* and we are interested in find the most optimum solution. In this approach, the entire solution space is represented in the form of a state space tree. As the program progresses each state combination is explored, and the previous solution is replaced by new one if it is not the optimal than the current solution.

   Example: Job sequencing, Travelling salesman problem.

iii) Classification by Design Approaches:

There are two approaches for designing an algorithm. These approaches include**:**

1. **Top-Down Approach :**
2. **Bottom-up approach**

- **Top-Down Approach:** In the top-down approach, a large problem is divided into small sub-problem and keep repeating the process of decomposing problems until the complex problem is solved.

- **Bottom-up approach:** The bottom-up approach is also known as the reverse of top-down approaches.
  In approach different, part of a complex program is solved using a programming language and then this is combined into a complete program.

iv) Other Classifications:

Apart from classifying the algorithms into the above broad categories, the algorithm can be classified into other broad categories like:

1. **Randomized Algorithms:** Algorithms that make random choices for faster solutions are known as randomized algorithms.
   Example: Randomized Quicksort Algorithm.

2. **Classification by complexity:** Algorithms that are classified on the basis of time taken to get a solution to any problem for input size. This analysis is known as time complexity analysis.
   Example: Some algorithms take O(n), while some take exponential time.

3. **Classification by Research Area:** In CS each field has its own problems and needs efficient algorithms.
   Example: Sorting Algorithm, Searching Algorithm, Machine Learning etc.

4. **Branch and Bound Enumeration and Backtracking:** These are mostly used in Artificial Intelligence.