***APPROACHES OF ALGORITHM***

1. **BRUTE FORCE ALGORITHM:** The general logic structure is applied to design an algorithm. It is also known as exhaustive search algorithm that searches all the possibilities to provide the required solution. Such algorithms are of two types.

**OPTIMIZING:** Finding all the solutions of a problem and then take out the best solution or if the value of the best solution is known then it will terminate if the best solution is known.

**SACRIFICING:** As soon as the best solution is found, then it will stop.

1. **DIVIDE & CONQUER:** It is a very implementation of an algorithm. It allows you to design an algorithm in a step-by-step variation. It breaks down the algorithm to solve the problem in different methods. It allows you to break down the problem into different methods, and valid output is produced for the valid input. This valid output is passed to some other function.
2. **GREEDY ALGORITHM:** It is an algorithm paradigm that makes an optimal choice on each iteration with the hope of getting the best solution. It is easy to implement and has a faster execution time, but there are very rare cases in which is provides the optimal solution.
3. **DYNAMIC PROGRAMMING:** It makes the algorithm more efficient by storing the intermediate results. It follows five different steps to find the optimal solution for the problem.
4. **BRANCH AND BOUND ALGORITHM:** The branch and bound algorithm can be applied to only integer programing problems. This approach divides all the sets of feasible solutions into smaller subsets. These subsets are further evaluated to find the best solution.
5. **RANDOMIZED ALGORITHM:** As we have seen in a regular algorithm, we have predefined input and required output, and follow some described steps are known as deterministic algorithms. What happens that when the random variable is introduced in the randomized algorithm. In a randomized algorithm, some random bits are introduced by the algorithm and added in the input to produce the output, which is random in nature. Randomized algorithm is simpler and efficient that the deterministic algorithm.
6. **BACKTRACKING:** It is an algorithm that solves the problem recursively and removes the solutions if it does not satisfy the constraints of a problem.

**ALGORITHM ANALYSIS**

1. **PRIORI ANALYSIS:** It is theoretical analysis done before implementing the algorithm. Various factors can be considered before implementing the algorithm like processor speed, which has no effect on the implementation part.
2. **POSTERIOR ANALYSIS:** It’s practical analysis, it is achieved by implementing the algorithm using any programming language. This analysis evaluates that how much running time and space taken by an algorithm.

**ALGORITHM COMPLEXITTY**

1. **TIME COMPLEXITY:** It is the amount of time required to complete the execution. **The time complexity of an algorithm is denoted by the big O notation.** Hence, big O notation is the asymptotic notation to represent the time complexity. The time complexity is mainly calculated by counting the number of steps to finish the execution.

**Example:**

**sum=0;**

**for i=1;** **i to n // calculate sum of n numbers**

**sum=sum+i;**

**return sum;**

Time complexity of the loop statement will be at least n, and if the value of n increases, then the time complexity also increases. While the complexity of the code, i.e., return sum will be constant as its value is not dependent on the value of n and will provide the result in one step only. We generally consider the worst-time complexity as it is the maximum time taken for any given input size.

1. **SPACE COMPLEXITY:** Amount of space required to solve a problem and produce an output. Similar to the time complexity, **space complexity is also expressed in big O notation.**

**AN ALGORITHM, THE SPACE IS REQUIRED FOR THE FOLLOWING PURPOSE.**

1. **TO STORE PROGRAM INSTRUCTIONS.**
2. **TO STORE CONSTANT VALUES**
3. **TO STORE VARIABLE VALUES**
4. **TO TRACK THE FUNCTION CALLS, JUMPLING STATEMENTS, ETC.**

AUXILIARY SPACE: The extra space required by the algorithm, excluding the input size, is known as auxiliary space. The space complexity considers both the spaces, i.e. auxiliary space and space used by input.

**Space complexity = Auxiliary Space + Input size.**