# Data Structures - Greedy Algorithms

An algorithm is designed to achieve an optimum solution for a given problem. In the greedy algorithm approach, decisions are made from the given solution domain. As being greedy, the closest solution that seems to provide an optimum solution is chosen.

Greedy algorithms try to find a localized optimum solution, which may eventually lead to globally optimized solutions. However, generally greedy algorithms do not provide globally optimized solutions.

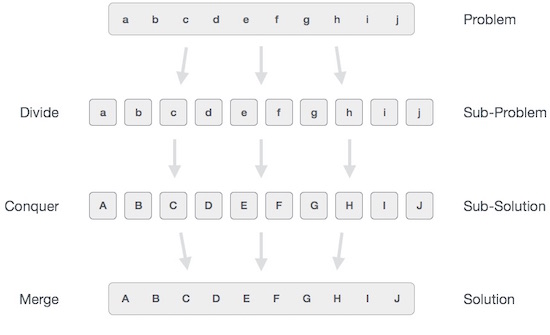
Example: When we are playing Cricket. At each ball, we would want to Hit a six to maximize your score.

### Examples

Most networking algorithms use the greedy approach. Here is a list of a few of them −

* Travelling Salesman Problem
* Prim's Minimal Spanning Tree Algorithm
* Kruskal's Minimal Spanning Tree Algorithm
* Dijkstra's Minimal Spanning Tree Algorithm
* Graph - Map Coloring
* Graph - Vertex Cover
* Knapsack Problem
* Job Scheduling Problem

There are lots of similar problems that use the greedy approach to find an optimum solution.



# Data Structures - Divide and Conquer

In divide and conquer approach, the problem in hand, is divided into smaller sub-problems and then each problem is solved independently. When we keep on dividing the subproblems into even smaller sub-problems, we may eventually reach a stage where no more division is possible. Those "atomic" smallest possible sub-problem (fractions) are solved. The solution of all sub-problems is finally merged in order to obtain the solution of an original problem.

Broadly, we can understand the divide-and-conquer approach in a three-step process.

## Divide/Break: This step involves breaking the problem into smaller sub-problems. At this stage, sub-problems become atomic in nature but still represent some part of the actual problem.

## Conquer/Solve: This step receives a lot of smaller sub-problems to be solved. Generally, at this level, the problems are considered 'solved' on their own.

## Merge/Combine: When the smaller sub-problems are solved, this stage recursively combines them until they formulate a solution to the original problem. This algorithmic approach works recursively and conquer & merge steps work so close that they appear as one.

Example: When we want to eat a bar of big fat chocolate. We break it into smaller pieces and have it one at a time.

### Examples

The following computer algorithms are based on the divide-and-conquer programming approach −

* Merge Sort
* Quick Sort
* Binary Search
* Strassen's Matrix Multiplication
* Closest pair (points)

There are various ways available to solve any computer problem, but the mentioned are a good example of the divide and conquer approach.

# Data Structures - Dynamic Programming

The dynamic programming approach is similar to divide and conquer in breaking down the problem into smaller and yet smaller possible sub-problems. But unlike, divide and conquer, these sub-problems are not solved independently. Rather, results of these smaller sub-problems are remembered and used for similar or overlapping sub-problems.

Dynamic programming is used where we have problems, which can be divided into similar sub-problems so that their results can be re-used. Mostly, these algorithms are used for optimization. Before solving the in-hand sub-problem, the dynamic algorithm will try to examine the results of the previously solved sub-problems. The solutions of sub-problems are combined in order to achieve the best solution.

So we can say that −

* The problem should be able to be divided into smaller overlapping sub-problem.
* An optimum solution can be achieved by using an optimum solution of smaller sub-problems.
* Dynamic algorithms use Memoization.

Example: Chess!... We can't just choose the best current move. We need to think about future possibilities and scenarios.

## Comparison

In contrast to greedy algorithms, where local optimization is addressed, dynamic algorithms are motivated for the overall optimization of the problem.

In contrast, to divide and conquer algorithms, where solutions are combined to achieve an overall solution, dynamic algorithms use the output of a smaller sub-problem and then try to optimize a bigger sub-problem. Dynamic algorithms use Memoization to remember the output of already solved sub-problems.

### Example

The following computer problems can be solved using the dynamic programming approach −

* Fibonacci number series
* Knapsack problem
* Tower of Hanoi
* All pair shortest path by Floyd-Warshall
* Shortest path by Dijkstra
* Project scheduling

Dynamic programming can be used in both top-down and bottom-up manner. And of course, most of the time, referring to the previous solution output is cheaper than recomputing in terms of CPU cycles.

Backtracking: Backtracking is an algorithmic technique that solves the problem recursively and removes the solution if it does not satisfy the constraints of a problem.

Branch and Bound Algorithm: The branch and bound algorithm can be applied to only integer programming problems. This approach divides all the sets of feasible solutions into smaller subsets. These subsets are further evaluated to find the best solution.

Randomized Algorithm: As we have seen in a regular algorithm, we have predefined input and required output. Those algorithms that have some defined set of inputs 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 algorithms are simpler and efficient than the deterministic algorithm.

Brute force algorithm: The general logic structure is applied to design an algorithm. It is also known as an 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.