# Module 5 - Decrease and Conquer Algorithm

## 1. Understanding Decrease and Conquer Approach

### Concepts:

**Definition: A problem-solving approach that exploits the relationship between a solution to a given instance of a problem and a solution to its smaller instance.**

**Process:**
**Decrease/Reduce: Reduce the problem instance to a smaller instance of the same problem.**

**Conquer/Solve: Solve the smaller instance of the problem.**

**Extend/Combine: Extend the solution of the smaller instance to obtain the solution to the original problem.**

### Comparison with Divide and Conquer:

**Divide and Conquer: Divides the problem into multiple subproblems, solves each subproblem recursively, and combines their solutions.**

**Decrease and Conquer: Reduces the problem to a single smaller subproblem and extends its solution.**

### Examples:

**Insertion Sort**
**Depth-First Search (DFS)**
**Breadth-First Search (BFS)**

## 2. Variations of Decrease and Conquer

**Variations:**

**Decrease by a Constant: The size of the instance is reduced by the same constant on each iteration.**

**Examples: Insertion Sort, DFS, BFS, Topological Sorting, Permutations/Subsets Generation.**

**Decrease by a Constant Factor: The size of the instance is reduced by the same constant factor on each iteration.**

**Examples: Binary Search, Fake-Coin Problems, Russian Peasant Multiplication.**

**Variable Size Decrease: The size-reduction pattern varies from one iteration to another.**

**Examples: Computing Median, Interpolation Search, Euclid's Algorithm.**

**3. Insertion Sort**

**Concepts:**

**Description: A sorting technique where the sorted array is built one item at a time by inserting elements in their correct position.**

**Steps:**

**Start with the second element and compare it with the previous elements.**

**Insert the element in its correct position by shifting other elements if necessary.**

**Repeat the process for all elements.**

```
void insertionSort(int arr[], int n) {
   for (int i = 1; i < n; ++i) {
      int key = arr[i];
      int j = i - 1;

      while (j >= 0 && arr[j] > key) {
         arr[j + 1] = arr[j];
         j--;
      }
      arr[j + 1] = key;
   }
}
```

**Analogy: Imagine arranging a deck of cards. You pick one card at a time and place it in the correct position among the already sorted cards.**

## Depth-First Search (DFS) (STACK)

### Concepts:

**Description: A graph traversal algorithm that explores as far as possible along each branch before backtracking.**

**Steps:**

**Visit an adjacent unvisited vertex, mark it as visited, display it, and push it onto a stack.**

**If no adjacent vertex is found, pop a vertex from the stack.**

**Repeat until the stack is empty.**

### Complexity Analysis:

**Time Complexity: O(V + E), where V is the number of vertices and E is the number of edges.**

**Space Complexity: O(V), due to the extra visited array needed.**

**Rules:**

**Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.**

**If no adjacent vertex is found, pop up a vertex from the stack.**

**Repeat Rule 1 and Rule 2 until the stack is empty.**

## 5. Breadth-First Search (BFS) (QUEUE)

**Concepts:**

**Description: A graph traversal algorithm that explores the neighbor nodes first before moving to the next level neighbors.**

**Steps:**

**Visit an adjacent unvisited vertex, mark it as visited, display it, and insert it into a queue.**

**If no adjacent vertex is found, remove the first vertex from the queue.**

**Repeat until the queue is empty.**

**Complexity Analysis:**

**Time Complexity: O(V + E), where V is the number of vertices and E is the number of edges.**

**Rules:**

**Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.**

**If no adjacent vertex is found, remove the first vertex from the queue.**

**Repeat Rule 1 and Rule 2 until the queue is empty.**