Problem Solving Techniques and Data-Structures:

1) Performance Measures:
    1) Timeliness
    2) Quality of Work
    3) Customer Orientation
    4) Optimal Solution
    5) Team Satisfaction
2) Problem:
    It is a sort of a puzzle that requires logical or mathematical solution
3) Problem Solving:
    Act of solving the problem by analysing it
4) Problem-solving-steps:
    1) Analyze and understand
    2) Select a method
    3) Design the solution
    4) Code the solution
    5) Test the solution
5) Classification:
    1) Concurrent: operations may overlap
    2) Sequential: operation are performed one-by-one
    3) Distributed: operations are performed at different locations
    4) Event-Based: input-driven
6) Problem-Solving Methods:
    1) Brute-Force: simple but costly(Hight time and space complexity)
    2) Greedy-Approach: selects the more optimal(time & space-efficient) solution out of the available set of solutions, works for optimisation of the problem
    3) Divide-And-Conquer:
        1) dividing it into smaller sub-problems and solving each sub-problem and at the end combining them to reach a solution
        2) Sub-problems: Disjoint{independent of each other}
    4) Dynamic-Programming: overlapping sub-problems
7) Computer Based Problem Solving Techniques:
    1) Modeling:
        1) Flow-Charts:
            1) Diagrammatic representation of an algo
            2) Memorize the symbols
        2) Entity-Relationship Diagrams
        3) UML(Unified Modeling Language)
8) Algorithm:
    way to solve a problem, it cannot be ambiguous in nature, it should work for all the desired inputs and return desired outputs in finite time & it is independent of the language

1) Characteristics:
    1) Finiteness
    2) Definiteness
    3) Input should be clearly specified
    4) Output format should be clearly specified
    5) Effectiveness-> steps should be basic and self-explanatory
2) Steps:
    1) Identify inputs and outputs
    2) Identify any other data and constants required to solve the problem
    3) Identify what needs to be computed
3) Different patterns in algorithm:
    1) Sequential: executes int the order in which they appear
    2) Selectional(Conditional): if-else if-else
    3) Iterational(Loop): Looping

9) Pseudo-Code:
    1) Fills the gap between an Alto and a program. It is a way to represent step-by-step solution to write a program.
    2) Syntax:

if <condition> then
— — — — — — — —
— — — — — — — —
else
— — — — — — — — —
— — — — — — — — —
end if

for i # 0 to 10
— — — — — — — — —
— — — — — — — — —
end for;

10) Algorithm Design:
    1) Searching Algorithms:
        1) Linear Search
        2) Binary Search:
            1) It applies to:
                1) Sorted Array
                2) BST(Binary Search Tree)(L<N<R):
                Because in-order traversal of BST results in a sorted array
    2) Sorting Techniques:
        1) Bubble Sort: Pairwise swap, move the largest element to the end.
        Time-Complexity -> O(n^2)

Pseudo Code:

```
begin bubbleSort(list)

  n : list.size


 for i # 0 to n-1:
    swapped = false
     for j # 0 to n-i-1:
        if (list[j]>list[j+1]) then
             swap(list[l],list[j+1])
             swapped=true
        End if
    End for;
    if not swapped then
       break
    End if
 End For;
```

2) Selection Sort: find the smallest element and swap it with the first element, stop when in the unsorted side only element is present, Time-Complexity -> O(n^2)
3) Insertion Sort
4) Merge Sort: Best & Worst case time Complexity -> O(nlogn)
5) Quick Sort:
    1) Pivot: it is the element based on which we partition our array
        1) Values -> a[0], a[n-1], a[mid], any random element
    2) Partition-Exchange Sort:
        1) It divides an array into 3 parts:
            1) Elements < pivot
            2) Pivot
            3) Elements > pivot
    3) Quick-Sort Tree:
        1) The Leaf-nodes are sub-sequences of size 0 or 1

6) Heap Sort:
    1) Satisfy following Conditions:
        1) Always be a complete binary tree -> it will have both the leaf nodes

3) Complexity of sorting all:
    1) Length of time a programmer spends on programming
    2) Time Complexity

3) Space Complexity

11) Data-Structures:
    1) Representation and manipulation of data
    2) Their study is the basis of programming
    3) Types:
        1) Linear: Arrays, ArrayList, LinkedList, Queues, Stacks etc.
        2) Non-Linear: Tree, Graphs, heaps etc.
    4) Size:
        1) Static: Fixed Size -> Arrays
        2) Dynamic: Variable Size, implemented using Links -> ArrayList, Vectors , Linked List, etc.
    5) Examples:
        1) Arrays:
            1) Static in nature: Fixed Size
            2) Indexing starts from 0 -> a.length-1
        2) Linked List:
            1) Order is not given by their physical placement in memory
            2) Elements can be added in front, middle or rear
            3) Can be used for implementing dynamic structures like Stacks and Queues
        3) Stack:
            1) Follows LIFO(Last-in-First-Out)
            2) Elements are always added to the top and removed from the top
            3) Major Operations:
                1) empty: return true if the stack is empty else return false
                2) full: returns true if the stack is completely filled, else false
                3) top: returns the value of the topmost element in the stack
                4) push: add element to the top of the stack
                5) pop: removes the element from the top of the stack
                6) displayStack: prints all the data in the stack
            4) Attributes:
                1) maxTop: the max size of the stack
                2) top: the current topmost element of the stack
        4) Queues:
            1) Follows FIFO(First-In-First-Out)
            2) Elements are inserted in rear and deleted from the front
            3) Attributes:
                1) Front-index
                2) Rear-index
                3) Counter: number of elements in the queue
                4) maxSize: size of the queue
            4) Major Operations:
                1) isEmpty: returns true if the queue is empty otherwise false

2) isFull: returns true if the queue is full otherwise false
3) Enqueue: adds data to the rear of the queue
4) Dequeue: removes the data from the front of the queue
5) displayQueue: print the data inside the queue

5) Trees(Non-Linear, do not have any internal loops):
   1) Root Node: Top Node/ Starting Node
   2) Leaf Nodes: Nodes that do not have any children
   3) Internal Nodes: non-root & non-leaf nodes
   4) Traversals:
      1) DFS(Depth-First Search): Pre-order(NLR), In-order(LNR), Post-Order(LRN)
      2) BFS(Breadth-First Search): Level-Order Traversal
6) Graphs(Non-Liner, they can have internal loops):
   1) No root-nodes => no fixed entry-point
   2) Made up of vertices and edges

Assignment:

1) **Write pseudo-codes for:**
   1) **Linear Search**
   2) **Binary-Search**
   3) **Selection Sort**