# University of Sargodha

## B.S 3rd Term Examination 2017

**Subject: I. T & S.E** **Paper: Data Structure & Algorithms (CMP:3112)**

**Time Allowed: 2:30 Hours** **Maximum Mar**

### Objective Part (Compulsory)

**Q.No.1. Write short answers of the following questions in 2-4 lines only. (16x2=32)**

1. What is data-structure?
2. What are various data-structures available?
3. What is algorithm?
4. Why we need to do algorithm analysis?
5. What are the criteria of algorithm analysis?
6. What are asymptotic notations?
7. Give some examples greedy algorithms.
8. Why do we use stacks?
9. What is shell sort?
10. How breadth first traversal works?
11. What is an AVL Tree?
12. What is a minimum spanning tree (MST) ?
13. What is interpolation search technique?
14. What is recursion?
15. What is adjacency list?
16. Define full binary tree?

### Subjective Part

**Note: Attempt any four questions.** **(4x12=48)**

**Q.No.2.** Give an algorithm in *O(nlogn)* that takes as input a sequence A of *n* integers, and gives back as output a sequence in which every integer from A occurs exactly once.

**Q.No.3.** Depict binary trees with heights 2, 3, 4, 5, and 6. All should be depicted with the following seven keys 1,4,5,10,16,17,21.

**Q.No.4.** What is the time complexity? Discuss time complexity of any two sorting algorithms.

**Q.No.5.** "Coffee Can Problem". In a coffee can C there are white and black beans, initially at least two beans. Moreover, there is a large pile of extra black beans. Show that the following process terminates. Denote the number of beans in the can by N(C) and can one know the color of the last bean in advance?

```
Input: Coffee can C with white and black beans.
Initially N(C) >= 2.
Output: Last bean in the can.

BEANS(C)
1.    while N(C) > 1
2.        select randomly two beans from the can C
3.            if beans are of same color
4.                throw both beans out, insert black bean in C
5.            else
6.                return the white bean to C and throw out the black bean
7.    return the last bean in the can
```

**Q.No.6.** Given an array A={12, 11, 13, 5, 6}. Sort it out using a technique illustrated in insertion sort. You have to discuss only the passes in detail and there is no need to write an algorithm of insertion sort.

**Q.No.7.** Write a program to insert a value at specified location in a link list.

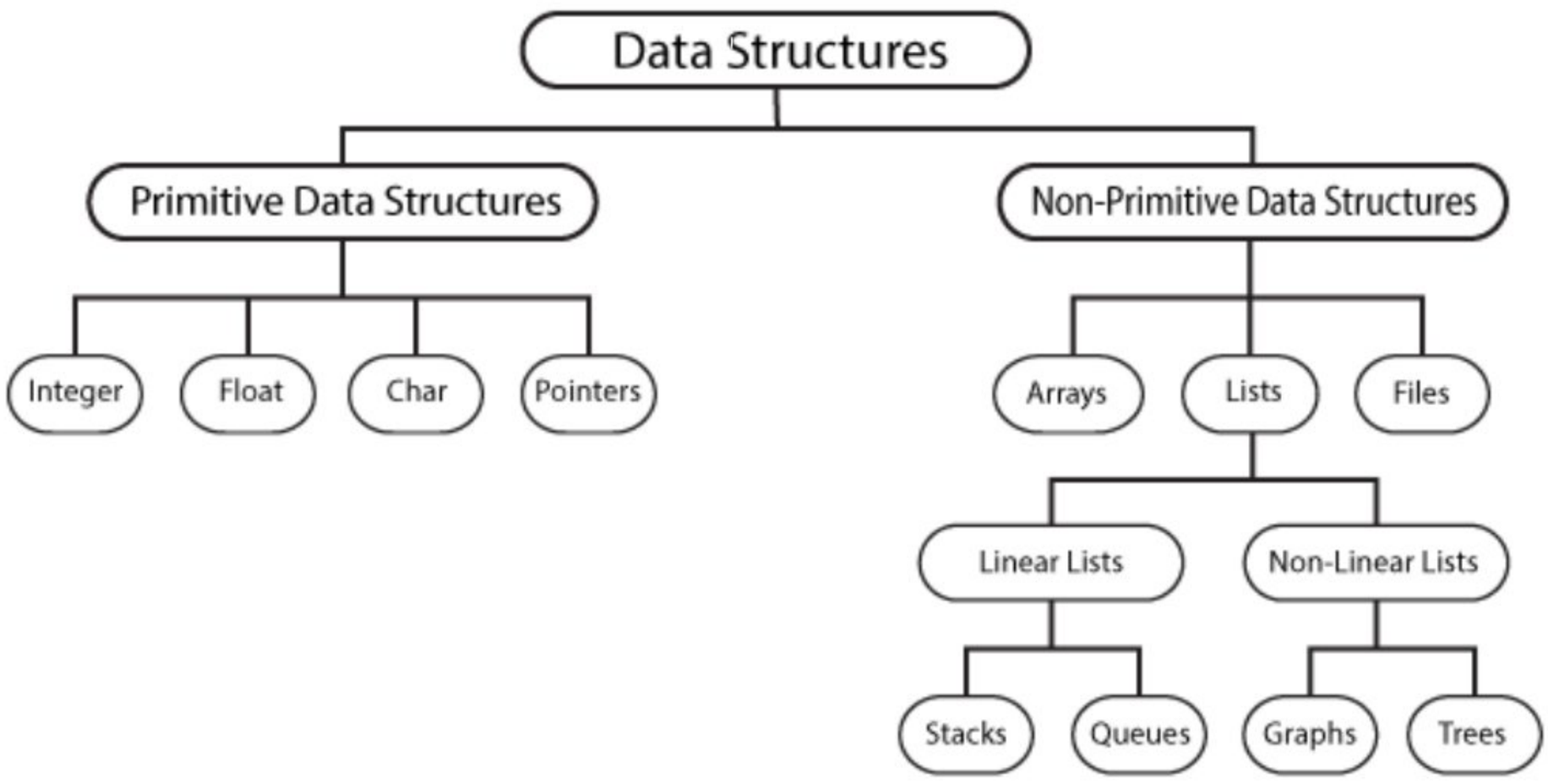
# Objective

## Q No 1: Write short answers.

### i. What is Data Structures?

A data structure is a specialized format for organizing and storing data. General data structure types include the array, the file, the record, the table, the tree, and so on. Any data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways. In computer programming, a data structure may be selected or designed to store data for the purpose of working on it with various algorithms.

### ii. What are the various Data-structures available?




**Primitive and Non-primitive data structures**

- Primitive Data Structures

Primitive Data Structures are the basic data structures that directly operate upon the machine instructions. They have different representations on different computers. Integers, Floating point numbers, Character constants, String constants and Pointers come under this category.

- Non-primitive Data Structures

Non-primitive data structures are more complicated data structures and are derived from primitive data structures.
They emphasize on grouping same or different data items with relationship between each data item. Arrays, Lists and Files come under this category.

## iii. What is algorithm?

An algorithm is defined as a step-by-step procedure or method for solving a problem by a computer in a finite number of steps. Steps of an algorithm definition may include branching or repetition depending upon what problem the algorithm is being developed for.

From the data structure point of view, following are some important categories of algorithms −

**Search −** Algorithm to search an item in a data structure.

**Sort** − Algorithm to sort items in a certain order.

**Insert** − Algorithm to insert item in a data structure.

**Update** − Algorithm to update an existing item in a data structure.

**Delete** − Algorithm to delete an existing item from a data structure.

## iv. Why we need to do algorithm analysis?

Efficiency of an algorithm can be analyzed at two different stages, before implementation and after implementation. They are the following −

**A Priori Analysis** This is a theoretical analysis of an algorithm. Efficiency of an algorithm is measured by assuming that all other factors, for example, processor speed, are constant and have no effect on the implementation.

**A Posterior Analysis** – This is an empirical analysis of an algorithm. The selected algorithm is implemented using programming language. This is then executed on target computer machine. In this analysis, actual statistics like running time and space required, are collected.

We shall learn about a priori algorithm analysis. Algorithm analysis deals with the execution or running time of various operations involved. The running time of an operation can be defined as the number of computer instructions executed per operation.

## v. What are the criteria of algorithm analysis?

It can be easily seen that "algorithm" is a fundamental notion in computer science. So it deserves a precise definition. The dictionary's definition, "any mechanical or recursive computational procedure," is not entirely satisfying since these terms are not basic enough.

**Definition:** An algorithm is a finite set of instructions which, if followed, accomplish a particular task.

In addition every algorithm must satisfy the following criteria:

**Input:** there are zero or more quantities which are externally supplied;

**Output:** at least one quantity is produced;

**Definiteness:** each instruction must be clear and unambiguous;

**Finiteness:** if we trace out the instructions of an algorithm, then for all cases the algorithm will terminate after a finite number of steps;

Effectiveness: every instruction must be sufficiently basic that it can in principle be carried out by a person using only pencil and paper.

It is not enough that each operation be definite, but it must also be feasible. In formal computer science, one distinguishes between an algorithm, and a program. A program does not necessarily satisfy the fourth condition. One important example of such a program for a computer is its operating system which never terminates (except for system crashes) but continues in a wait loop until more jobs are entered.

## vi. What are the asymptotic notations?

Asymptotic analysis of an algorithm refers to defining the mathematical foundation /framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

Asymptotic analysis is input bound i.e., if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant.

Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation. For example, the running time of one operation is computed as f(n) and may be for another operation it is computed as g(n2). This means the first operation running time will increase linearly with the increase in n and the running time of the second operation will increase exponentially when n increases. Similarly, the running time of both operations will be nearly the same if n is significantly small.

Usually, the time required by an algorithm falls under three types −

**Best Case** − Minimum time required for program execution.

**Average Case** − Average time required for program execution.

**Worst Case** − Maximum time required for program execution.

## vii. Give some examples greedy algorithms?

An algorithm is designed to achieve optimum solution for a given problem. In 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.

### Examples

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

i. Travelling Salesman Problem
ii. Prim's Minimal Spanning Tree Algorithm
iii. Graph - Map Coloring
iv. Graph - Vertex Cover
v. Job Scheduling Problem

## viii. Why do we use stack?

Because they help manage your data in more a particular way than arrays and lists. Queue is first in, first out (FIFO) and Stack is last in, first out (LIFO).

Arrays and lists are random access. They are very flexible and also easily corruptible. IF you want to manage your data as FIFO or LIFO it's best to use those, already implemented, collections.

## ix. What is Shell Sort?

Shell sort is a highly efficient sorting algorithm and is based on insertion sort algorithm. This algorithm avoids large shifts as in case of insertion sort, if the smaller value is to the far right and has to be moved to the far left.

This algorithm uses insertion sort on a widely spread elements, first to sort them and then sorts the less widely spaced elements. This spacing is termed as interval.

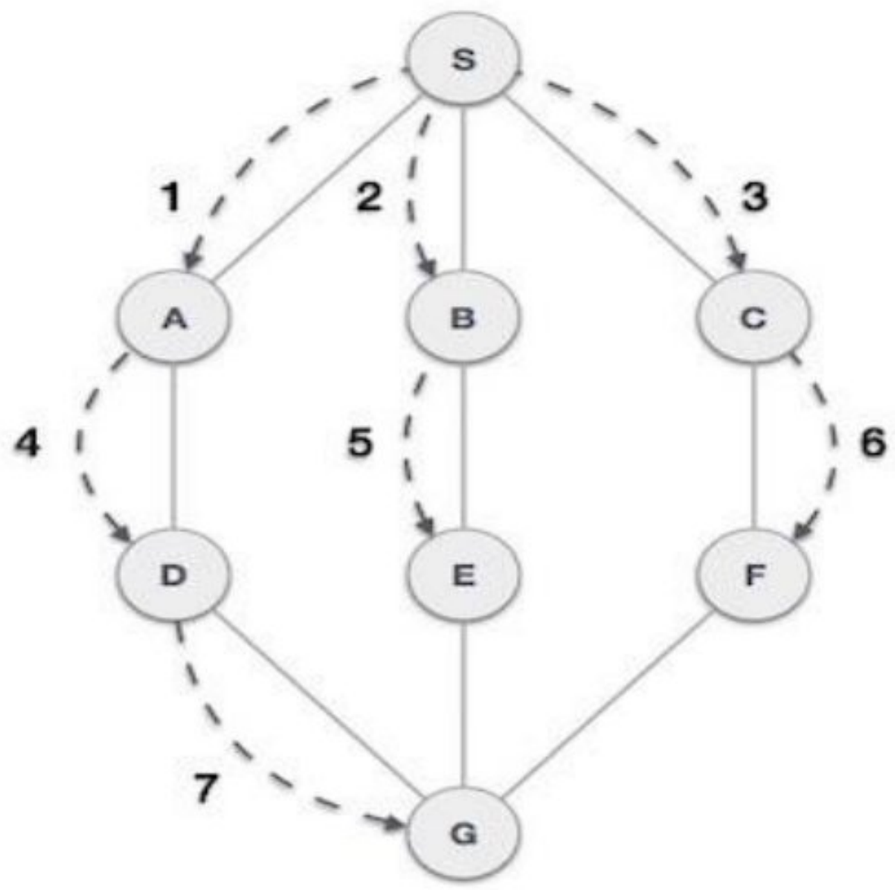
Running time analysis:

Best case: O(N logn)

Worst case: O(N2)

## x. How breadth traversal works?

Breadth First Search (BFS) algorithm traverses a graph in a breadth ward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.




As in the example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs the following rules.

**Rule 1** − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.

**Rule 2** − If no adjacent vertex is found, remove the first vertex from the queue.

**Rule 3** − Repeat Rule 1 and Rule 2 until the queue is empty.

## xi. What is an AVL Tree?

An AVL tree is another balanced binary search tree. Named after their inventors, Adelson-Velskii and Landis, they were the first dynamically balanced trees to be proposed. Like red-black trees, they are not perfectly balanced, but pairs of sub-trees differ in height by at most 1, maintaining an O(logn) search time. Addition and deletion operations also take O(logn) time.

An AVL tree is a binary search tree which has the following properties:

i. The sub-trees of every node differ in height by at most one.
ii. Every sub-tree is an AVL tree.

## xii. What is a minimum spanning tree (MST)?

A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted (un)directed graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight.

## xiii. What is interpolation search technique?

Linear Search finds the element in O(n) time, Jump Search takes O(√ n) time and Binary Search take O(Log n) time.

The Interpolation Search is an improvement over Binary Search for instances, where the values in a sorted array are uniformly distributed. Binary Search always goes to middle element to check. On the other hand interpolation search may go to different locations according the value of key being searched. For example if the value of key is closer to the last element, interpolation search is likely to start search toward the end side. Linear Search finds the element in O(n) time, Jump Search takes O(√ n) time and Binary Search take O(Log n) time. The Interpolation Search is an improvement over Binary Search for instances, where the values in a sorted array are uniformly distributed. Binary Search always goes to middle element to check. On the other hand interpolation search may go to different locations according the value of key being searched. For example if the value of key is closer to the last element, interpolation search is likely to start search toward the end side.

**Algorithm**

Rest of the Interpolation algorithm is same except the above partition logic.

**Step1:** In a loop, calculate the value of "pos" using the probe position formula.
**Step2:** If it is a match, return the index of the item, and exit.
**Step3:** If the item is less than arr[pos], calculate the probe position of the left sub-array. Otherwise calculate the same in the right sub-array.
**Step4:** Repeat until a match is found or the sub-array reduces to zero.

## xiv. What is recursion?

Some computer programming languages allow a module or function to call itself. This technique is known as recursion. In recursion, a function α either calls itself directly or calls a function β that in turn calls the original function α. The function α is called recursive function.

## xv. What is adjacency list?

In graph theory and computer science, an adjacency list is a collection of unordered lists used to represent a finite graph. Each list describes the set of neighbors of a vertex in the graph. This is one of several commonly used representations of graphs for use in computer programs.

## xvi. Define Full Binary Tree?

A full binary tree (sometimes proper binary tree or 2-tree) is a tree in which every node other than the leaves has two children. A complete binary tree is binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

# Subjective

**Q 2: Give an algorithm, in O(nlogn)that takes as input a sequence A of n integers, and gives back as output a sequence in which every integer from A occurs only once.**

Ans: Here in the given question A is the Name of an Array. And we have to sort the Array A with a sort whose time complexity is O(nlogn).

Firstly we have to check that how many sorts are there whose time complexity is O(nlogn),

These are

a) **Merge Sort**
b) **Quick Sort**
c) **Heap Sort**

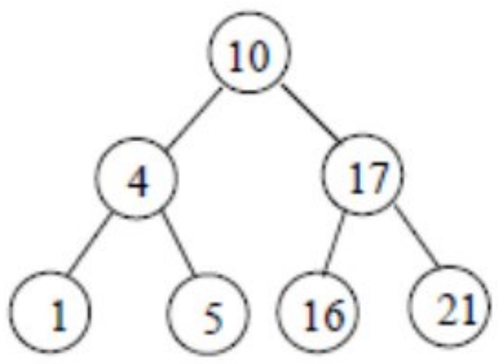
So, we choose Quick Sort for this Purpose.

```
public class QuickSort {
        private int arr[];
        private int length;

        public void sort(int[] A) {

            if (A == null || A.length == 0) {
                return;
            }
            this.arr = A;
            length = A.length;
            quickSort(0, length - 1);
        }

        private void quickSort(int lowerIndex, int higherIndex) {

            int i = lowerIndex;
            int j = higherIndex;
            // calculate pivot number, I am taking pivot as middle index number
            int pivot = arr[lowerIndex+(higherIndex-lowerIndex)/2];
            // Divide into two arrays
            while (i <= j) {
                while (arr[i] < pivot) {
                    i++;
                }
                while (arr[j] > pivot) {
                    j--;
                }
                if (i <= j) {
                    exchangeNumbers(i, j);
                    //move index to next position on both sides
                    i++;
                    j--;
                }
            }
            // call quickSort() method recursively
            if (lowerIndex < j)
                quickSort(lowerIndex, j);
            if (i < higherIndex)
                quickSort(i, higherIndex);
    }

     private void exchangeNumbers(int i, int j) {
         int temp = arr[i];
         arr[i] = arr[j];
         arr[j] = temp;
     }

        public static void main(String a[]){

            QuickSort sorter = new QuickSort();
            int[] A = {24,2,91,49,20,68,17};
            System.out.println("Array Before Quick Sort:");
            for(int i:A){
            System.out.print(i+" ");
            }
            System.out.println();

            sorter.sort(A);
            System.out.println("Array After Quick Sort:");
            for(int i:A){
                System.out.print(i);
                System.out.print(" ");
            }
        }
}
```

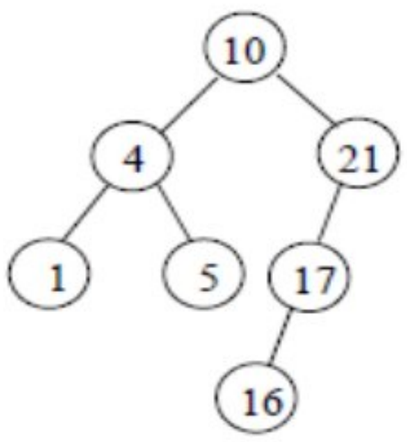
## Q 3: Depict binary tree with heights 2,3,4,5 and 6. All should be depicted with the following seven keys 1, 4,5,10,16,17,21.

For Key values 1, 4,5,10,16,17,21 the binary trees of heights 2,3,4,5 and 6 are following

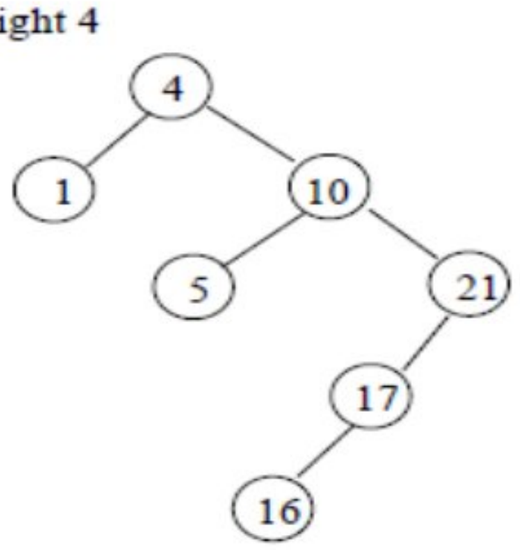
Height 2



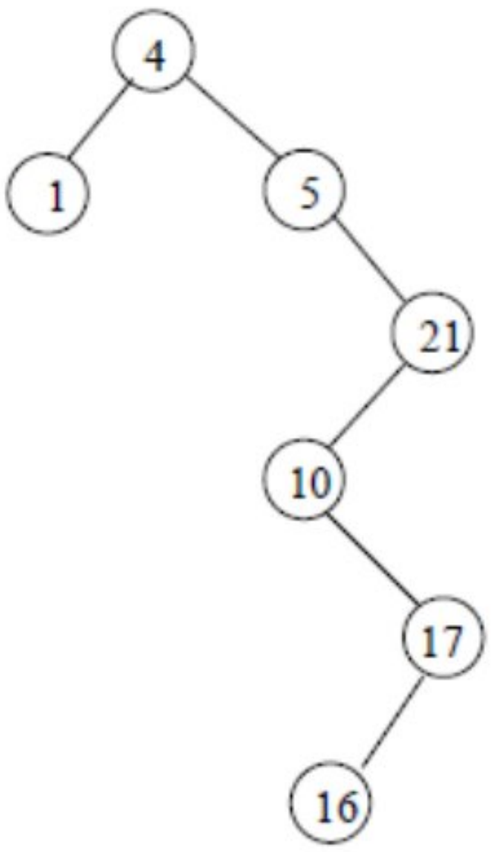

Height 3



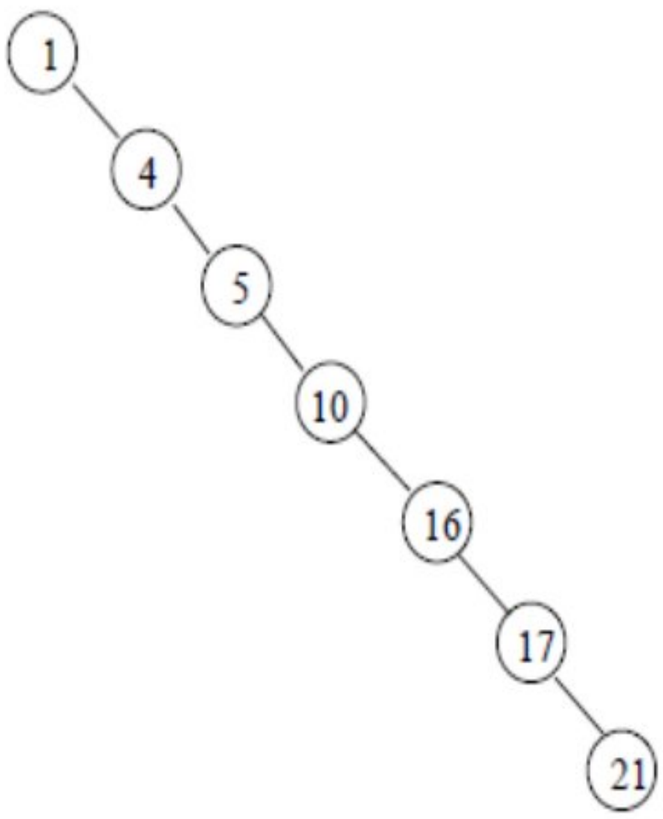

Height 4




height 5




Height 6




## Q 4: what is time complexity? Discuss time complexity of any TWO sorting algorithms.

For any defined problem, there can be N number of solution. This is true in general. Time complexity of an algorithm signifies the total time required by the program to run till its completion.

The time complexity of algorithms is most commonly expressed using the **big O notation**. It's an asymptotic notation to represent the time complexity. Time Complexity is most commonly estimated by counting the number of elementary steps performed by any algorithm

to finish execution. And since the algorithm's performance may vary with different types of input data, hence for an algorithm we usually use the **worst-case Time complexity** of an algorithm because that is the maximum time taken for any input size.

We can have three cases to analyze an algorithm:
1) Worst Case
2) Average Case
3) Best Case

- **Worst Case Analysis (Usually Done)**

In the worst case analysis, we calculate upper bound on running time of an algorithm. We must know the case that causes maximum number of operations to be executed. For Linear Search, the worst case happens when the element to be searched (x in the above code) is not present in the array. When x is not present, the search () functions compares it with all the elements of arr[] one by one. Therefore, the worst case time complexity of linear search would be O(n).

- **Average Case Analysis (Sometimes done)**

In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by total number of inputs. We must know (or predict) distribution of cases. For the linear search problem, let us assume that all cases are uniformly distributed (including the case of x not being present in array). So we sum all the cases and divide the sum by (n+1). Following is the value of average case time complexity.

**Best Case Analysis (Bogus)**

In the best case analysis, we calculate lower bound on running time of an algorithm. We must know the case that causes minimum number of operations to be executed. In the linear search problem, the best case occurs when x is present at the first location. The number of operations in the best case is constant (not dependent on n). So time complexity in the best case would be O(1)

## Bubble sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.
**Worst and Average Case Time Complexity:** O(n*n). Worst case occurs when array is reverse sorted.
**Best Case Time Complexity:** O(n). Best case occurs when array is already sorted.

# Quick Sort

Like Merge Sort, Quick Sort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quick Sort that pick pivot in different ways.
Always pick first element as pivot.

Always pick last element as pivot (
Pick a random element as pivot.
Pick median as pivot.

**Worst Case:** The worst case occurs when the partition process always picks greatest or smallest element as pivot. The solution of such situation will cause complexity O(n2).

**Best Case:** The best case occurs when the partition process always picks the middle element as pivot. The solution of such situation will cause complexity O(nLogn).

**Average Case:** To do average case analysis, we need to consider all possible permutation of array and calculate time taken by every permutation which doesn't look easy. The solution of such situation will cause complexity O(nLogn)

**Q 6: Given an Array A= {12, 11, 13, 5, 6}. Sort it out using a technique illustrated in insertion sort. You have to discuss only the passes in details and there is no need to write an algorithm of insertion sort.**

In insertion sort we have two group of items:

i. sorted group, and
ii. unsorted group

Initially, all items in the unsorted group and the sorted group is empty. We assume that items in the unsorted group unsorted. We have to keep items in the sorted group sorted. Pick any item from unsorted, then insert the item at the right position in the sorted group to maintain sorted property. Repeat the process until the unsorted group becomes empty.

Original Array: {12, 11, 13, 5, 6}

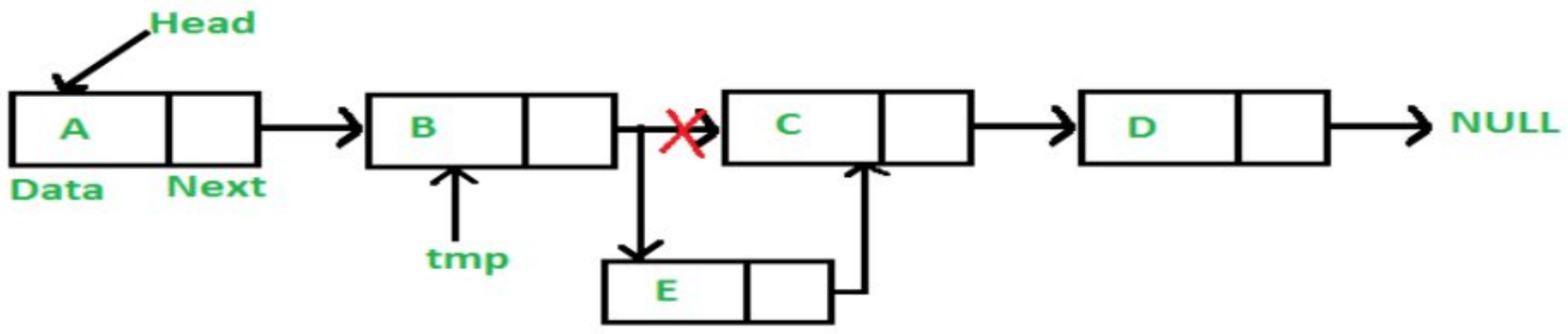
After Pass 1: {11, 12, 13, 5, 6}

After Pass 2: {11, 12, 13, 5, 6}

After Pass 3: {5, 11, 12, 13, 6}

After Pass 4: {5, 6, 11, 12, 13}

## Q 7: Write a program to insert a value at specified location in a linked list.




Code on next page

```
public class LinkList {

    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d) {data = d; next = null; }
    }



    /* Inserts a new node after the given prev_node. */
    public void insertAfter(Node prev_node, int new_data)
    {
        /* 1. Check if the given Node is null */
        if (prev_node == null)
        {
            System.out.println("The given previous node cannot be null");
            return;
        }

        /* 2 & 3: Allocate the Node &
                  Put in the data*/
        Node new_node = new Node(new_data);

        /* 4. Make next of new Node as next of prev_node */
        new_node.next = prev_node.next;

        /* 5. make next of prev_node as new_node */
        prev_node.next = new_node;
    }
    /* This function prints contents of linked list starting from
    the given node */
public void printList()
{
    Node tnode = head;
    while (tnode != null)
    {
        System.out.print(tnode.data+" ");
        tnode = tnode.next;
    }
}

/* Driver program to test above functions. Ideally this function
   should be in a separate user class.  It is kept here to keep
   code compact */
public static void main(String[] args)
{
    /* Start with the empty list */
    LinkedList llist = new LinkedList();

        // Insert 7 at the beginning. So linked list becomes
    // 7->NUllist
    llist.push(7);

    // Insert 1 at the beginning. So linked list becomes
    // 1->7->NUllist
    llist.push(1);

    // Insert 4 at the end. So linked list becomes
    // 1->7->4->NUllist
    llist.append(4);


    System.out.println("\nCreated Linked list is: ");
    llist.printList();
}
}
```