

**Data Structure & Algorithm (CMP-3112)**

Class: BS (CS, SE, IT), M.sc IT 3<sup>rd</sup>

**Short +Long Questions**

If you have any query feel free to contact at: [asim.rana63@gmail.com](mailto:asim.rana63@gmail.com)

---

**Answer the following Questions**

**1) What is data structure?**

**Ans:** In computer science, a data structure is a data organization, management and storage format that enables efficient access and modification. More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data

**2) What is an Algorithm?**

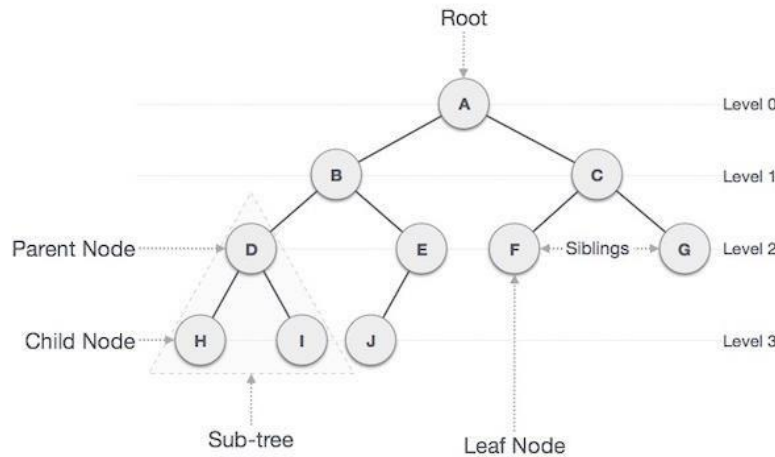
**Ans:** An algorithm is a step by step method of solving a problem. It is commonly used for data processing, calculation and other related computer and mathematical operations. An algorithm is also used to manipulate data in various ways, such as inserting a new data item, searching for a particular item or sorting an item.

**3) What are asymptotic notations?**

**Asymptotic Notations** are languages that allow us to analyze an algorithm's running time by identifying its behavior as the input size for the algorithm increases. This is also known as an algorithm's growth rate. Big-oh (O), Omega ( $\pi$ ), and theta ( $\Theta$ ) notations are asymptotic.

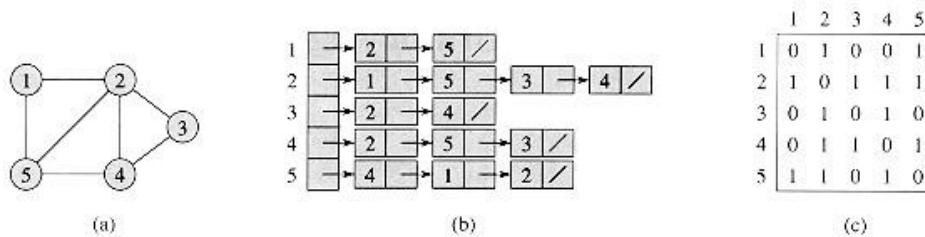
**4) What is tree?**

**Ans:** A data structure that simulates hierarchal tree structure, which a root value and sub tree of children with a parent node.



### 5) What is adjacency list?

**Ans:** 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.



### 6) How DFS and BFS traversal work?

**Ans:** BFS (Breadth First Search) traverse on graph using FIFO (First in first out) function and DFS (Depth First Search) traverse on graph using LIFO (Last in first out) function.

### 7) Why do we use stack?

**Ans:** In computer science, a stack is an abstract data type that serves as a collection of elements, with two principal operations: push, which adds an element to the collection, and pop, which removes the most recently added element that was not yet removed.

- Stack is an ordered list of similar data type
- Stack is a **LIFO** (Last in First out) structure or we can say **FIFO** (First in Last out).
- push () function is used to insert new elements into the Stack and pop () function is used to remove an element from the stack. Both insertion and removal are allowed at only one end of Stack called Top.
- Stack is said to be in Overflow state when it is completely full and is said to be in Underflow state if it is completely empty.

### Analysis of Stack Operations

Below mentioned are the time complexities for various operations that can be performed on the Stack data structure

- Push operation:  $O(1)$
- Pop operation:  $O(1)$
- Top operation:  $O(1)$
- Search operation:  $O(n)$

#### 8) What is divide and conquer approach?

**Ans:** In divide and conquer approach, the problem in hand, is divided into smaller subproblems 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.

#### 9) Define an Abstract Data Type (ADT)

Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of value and a set of operations. The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called "abstract" because it gives an implementation independent view. The process of providing only the essentials and hiding the details is known as abstraction.

Three ADTs namely List ADT, Stack ADT, Queue ADT.

(is question mn us na ADT k about puxha hy , just upper sa kuch definition jo underline hy wo ikh lyn to kafi hoga, but wo in types mn bhi puch skta hy is ly unki details bhi bta di hy )

##### List ADT

A list contains elements of same type arranged in sequential order and following operations can be performed on the list.

**get()** – Return an element from the list at any given position.

**insert()** – Insert an element at any position of the list. **remove()** – Remove the

first occurrence of any element from a non-empty list. **remove At()** – Remove the element at a specified location from a nonempty list. **replace()** – Replace an element at any position by another element.

**size()** – Return the number of elements in the list. **isEmpty()** –

Return true if the list is empty, otherwise return false. **isFull()** –

Return true if the list is full, otherwise return false.

### Stack ADT

A Stack contains elements of same type arranged in sequential order. All operations takes place at a single end that is top of the stack and following operations can be performed: **push()** – Insert an element at one end of the stack called top. **pop()** – Remove and return the element at the top of the stack, if it is not empty. **peek()** – Return the element at the top of the stack without removing it, if the stack is not empty.

**size()** – Return the number of elements in the stack.

**isEmpty()** – Return true if the stack is empty, otherwise return false. **isFull()**

– Return true if the stack is full, otherwise return false.

### Queue ADT

A Queue contains elements of same type arranged in sequential order. Operations takes place at both ends, insertion is done at end and deletion is done at front. Following operations can be performed: **enqueue()** – Insert an element at the end of the queue.

**dequeue()** – Remove and return the first element of queue, if the queue is not empty.

**peek()** – Return the element of the queue without removing it, if the queue is not empty.

**size()** – Return the number of elements in the queue. **isEmpty()** – Return true if the queue is empty, otherwise return false. **isFull()** – Return true if the queue is full, otherwise return false.

#### 10) What is the post fix notation of $(a + b) * (c + d)$ ?

- $*+ab+cd$

#### 11) What is the difference of PUSH and POP?

- Stack is an ordered list of similar data type
- Stack is a LIFO (Last in First out) structure or we can say FIFO (First in Last out).
- push () function is used to insert new elements into the Stack and pop() function is used to remove an element from the stack. Both insertion and removal are allowed at only one end of Stack called Top.
- Stack is said to be in Overflow state when it is completely full and is said to be in Underflow state if it is completely empty.

#### 12) Write an algorithm to POP a value from stack?

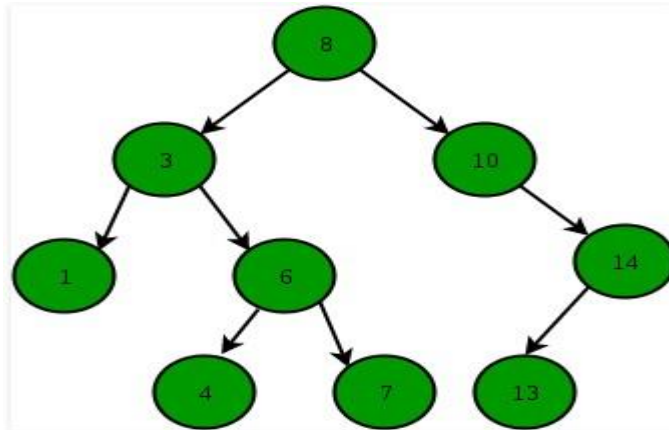
```
begin procedure pop: stack
    if stack is
empty
return null
endif      data
← stack[top]
top ← top - 1
return data
```

end procedure

### 13) What is Binary Search Tree (BST)?

**Binary Search Tree** is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.



### 14) What is non-linear Data Structure? Name any two non-linear Data Structure.

**Ans: Non-Linear Data Structures:** The **data structure** where **data** items are not organized sequentially is called **nonlinear data structure**. Graphs and trees, Binary search tree and Spanning tree.

### 15) Define Array.

**Ans:** Array is a container which can hold a fix number of items and these items should be of the same type. Most of the data structures make use of arrays to implement their algorithms. Following are the important terms to understand the concept of Array.

- **Element** – Each item stored in an array is called an element.
- **Index** – Each location of an element in an array has a numerical index, which is used to identify the element.

### 16) What is Data Type?

**Ans:** A **data type** is a set of **data** values having predefined characteristics. You will have encountered **data types** during computer programming classes. Example **data types** would be integer, floats, string and characters. Generally, in all programming languages we have a limited number of such **data types**.

### 17) Discuss any application of Stack?

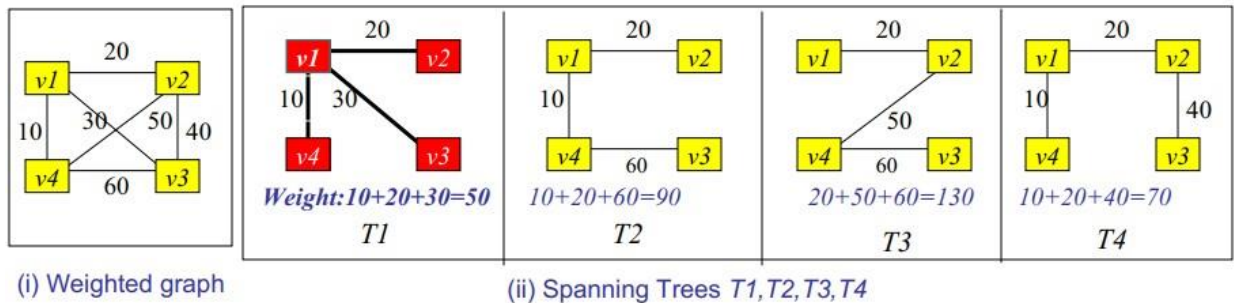
**Ans:** The simplest application of a stack is to reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack.

There are other uses also like:

- Parsing
- Expression Conversion (Infix to Postfix, Postfix to Prefix etc)

### 18) What is minimum spanning tree (MST)?

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



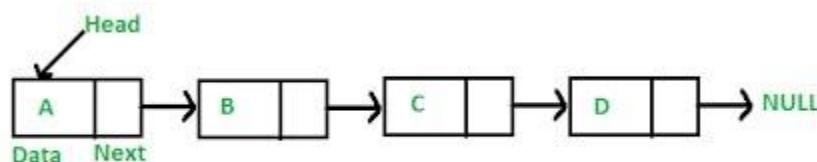
### 19) Why is not possible to find out the shortest path from source vertex to any other vertex if graph is having a cycle?

**Answer:** The question is incorrect, because it is possible to find the shortest path from the source vertex to any other vertex, even when the graph has a cycle.

In fact, if you think about it, if the graph has no cycle, then the graph is a forest of trees, each of which is unreachable from one another. That means, if two vertices have “a” path, that is also “the” path between them. So, finding the shortest path in this case reduces to connectivity problem, which is a simpler problem.

### 20) What is a linked list?

**Answer:** A **linked list** is represented by a pointer to the first node of the **linked list**. The first node is called head. If the **linked list** is empty, then value of head is NULL. Below is an **example** of a **linked list** node with an integer data. In Java, **Linked List** can be represented as a class and a Node as a separate class.



### 21) What is Binary search?

**Answer: Binary search** works on sorted arrays. **Binary search** begins by comparing the middle element of the array with the target value. If the target value matches the middle element, its position in the array is returned. If the target value is less than the middle element, the **search** continues in the lower half of the array

### 22) List out advantages of using linked list.

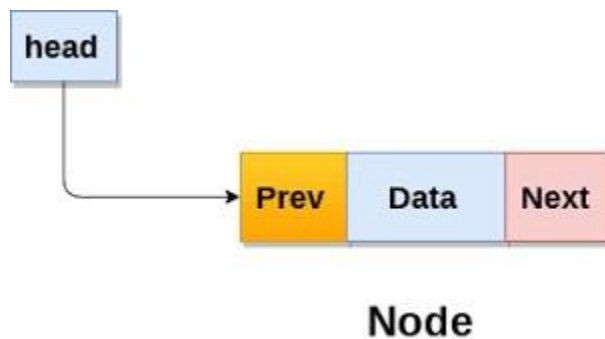
- Dynamic Data Structure. Linked list is a dynamic data structure so it can grow and shrink at runtime by allocating and de allocating memory.
- Insertion and Deletion. Insertion and deletion of nodes are really easier.
- No Memory Wastage.
- Implementation.
- Memory Usage.
- Traversal.
- Reverse Traversing.

### 23) Compare array and linked list.

**Answer:** The major difference between **Array** and **Linked list** regards to their structure. **Arrays** are index-based data structure where each element associated with an index. ... While a **linked list** is a data structure which contains a sequence of the elements where each element is **linked** to its next element.

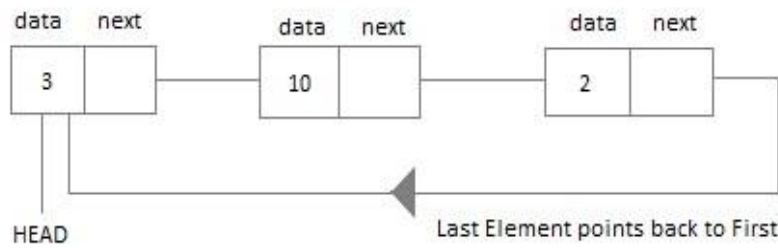
### 24) What is doubly linked list?

**Answer:** A **doubly linked list** is a **linked** data structure that consists of a set of sequentially **linked** records called nodes. Each node contains three fields: two **link** fields (references to the previous and to the next node in the sequence of nodes) and one data field.



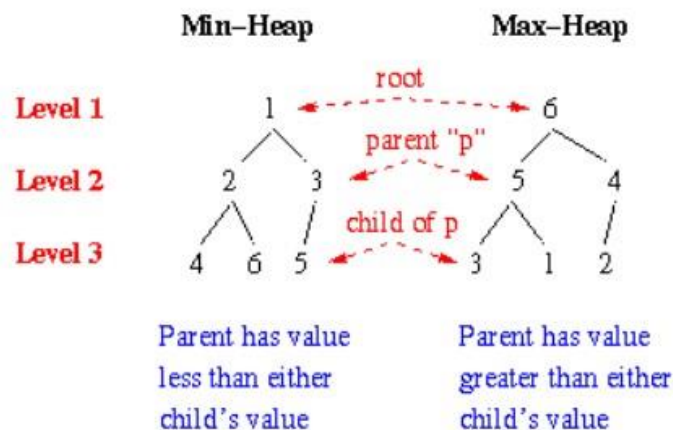
### 25) What is circular linked list?

**Answer: Circular linked list** is a sequence of elements in which every element has **link** to its next element in the sequence and the last element has a **link** to the first element in the sequence. That means **circular linked list** is similar to the single **linked list** except that the last node points to the first node in the **list**.



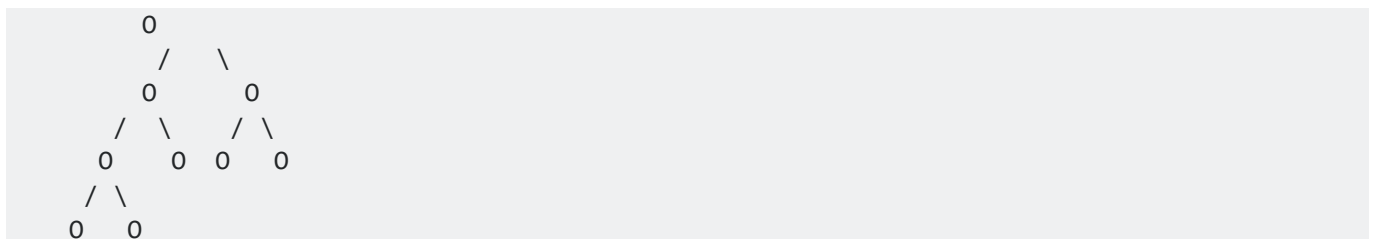
26) What is min and max heap?

Ans: A max-heap is a complete binary tree in which the value in each internal node is greater than or equal to the values in the children of that node. A min-heap is defined similarly.



27) What is almost complete binary tree (ACBT)?

**Ans:** A **complete tree** is a tree in which every level is completely filled and an **Almost complete tree** is a tree in which if last level is not completely filled then all nodes are as far as left as possible. my confusion is in following example of binary tree:



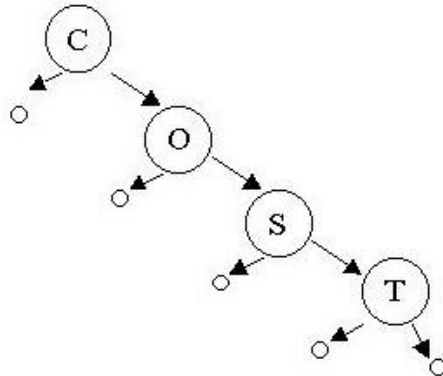
28) What is AVL tree?



**Ans:** 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 29)

**Write disadvantage of BST?**

**Ans:** when inserting or searching for an element, the key of each visited node has to be compared with the key of the element to be inserted/found. Keys may be long and the run time may increase much.



**30) What is LIFO and FIFO?**

**Ans:** LIFO is stand for last in first out and work using queue data structure. and FIFO stand for first in first out and work using stack data structure.

**31) A full node is a node with two children. Prove that the number of full nodes plus one is equal to the number of leaves in a nonempty binary tree. Method:1**

**Proof:** Let's use the following notation for our proof:

N= number of nodes in a nonempty binary tree

F=number of full nodes

H=number of nodes with one child

L=number of leaves

Then, we have  $N=F+H+L$ . We can get another equation based on the number of edges:  $N-1=2F+H$  is the number of edges for a N node binary tree and  $2F+H$  is another way to calculate the number of edges. Now based on these two equations we have:

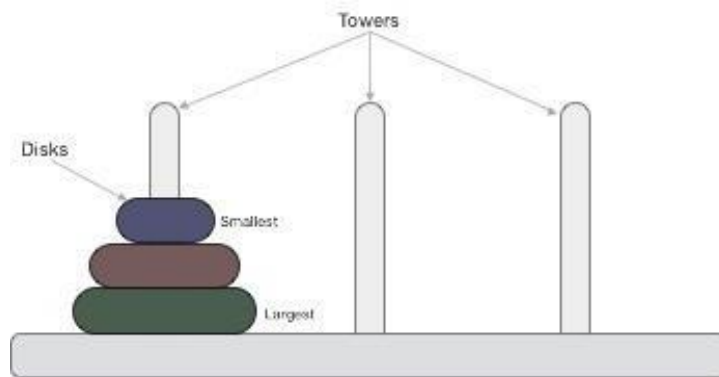
$$2F+H+1=F+H+L$$

**32) What is linear search?**

**Ans: Linear search** is a very simple **search** algorithm. In this type of **search**, a **sequential search** is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the **search** continues till the end of the **data** collection.

### 33) What is Tower of Hanoi?

**Ans:** Tower of Hanoi, is a mathematical puzzle which consists of three towers (pegs) and more than one rings is as depicted.



These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same.

#### Rules:

- Only one disk can be moved among the towers at any given time.
- Only the "top" disk can be removed.
- No large disk can sit over a small disk.

### 34) Write a recursive program for tower of Hanoi?

```
START
Procedure Hanoi(disk, source, dest, aux)

  IF disk == 1, THEN
    move disk from source to dest
  ELSE
    Hanoi(disk - 1, source, aux, dest)    // Step 1    move disk
    from source to dest                    // Step 2    Hanoi(disk - 1, aux, dest,
    source)                                // Step 3
  END IF
```

END Procedure  
STOP

35) Every recursive algorithm has one or more base cases? Is it True or False? Explain to motivate your answer.

**Ans:** Yes, every recursive definition must have one (or more) base cases. The general case must eventually reduce to the base case, a solution that solves the problem by breaking it into smaller versions of itself, then using the necessary code to solve.

a function that calls itself in its body.

36) Explain why the number of item movements in Shell sort is less than the number of item movements in insertion sort.

37) Both merge sort and quicksort sort a list by partitioning the list. Explain how merge sort differs from quicksort in partitioning the list.

38) Give some examples of greedy algorithm.

**Ans:** Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So, the problems where choosing locally optimal also leads to global solution are best fit for Greedy. Examples: Dijkstra's Shortest Path Algorithm, Problem Solving for Minimum Spanning Trees (Kruskal's and Prim's)

39) Why we need to do algorithm analysis?

**Ans:**

40) What are the criteria of algorithm analysis?

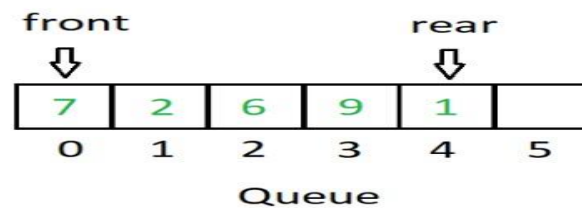
**Ans:** the analysis of algorithms is the determination of the computational complexity of algorithms, that is the amount of time, storage and/or other resources necessary to execute them.

41) Differentiate between Queue and Array.

**Ans:** Queue is a FIFO (First-In, First-Out) list, a list-like structure that provides restricted access to its elements: elements may only be inserted at the back and removed from the front. queues are less flexible than lists.

**Enqueue:** insert elements into queue at the back.

**Dequeue:** remove elements from the front.



**Array:** An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).

42) Compare linked list and Array

Ans:

Array	Linked List
Arrays are linear data structures.	Linked lists are linear and non-linear data structures. Linked lists are linear for accessing, and non-linear for storing in memory.
Array has homogenous values. And each element is independent of each other positions.	Each node in the linked list is connected with its previous node which is a pointer to the node.
Array elements can be modified easily by identifying the index value.	It is a complex process for modifying the node in a linked list.
Array elements <u>can not be added, deleted</u> once it is declared.	The nodes in the linked list can be added and deleted from the list.

43) Define O-notation.

**Definition:** A theoretical measure of the execution of an algorithm, usually the time or memory needed, given the problem size  $n$ , which is usually the number of items. **Formal**

**Definition:**  $f(n) = O(g(n))$  means there are positive constants  $c$  and  $k$ , such that  $0 \leq f(n) \leq cg(n)$  for all  $n \geq k$ .

44) Why we do use stack?

**Ans:** The **advantage** of using the **stack** to store variables, is that memory is managed for you. You don't have to allocate memory by hand, or free it once you don't need it any more.

What's more, because the CPU organizes **stack** memory so efficiently, reading from and writing to **stack** variables is very fast.

45) **When is a binary search best applied?**

**Ans:** A binary search is an algorithm that is best applied to search a list when the elements are already in order or sorted.

46) **What is Fibonacci series?**

Data Structure & Algorithms Fibonacci Series. Advertisements. Fibonacci series generates the subsequent number by adding two previous numbers.

Fibonacci series is defined as a sequence of numbers in which the first two numbers are 1 and 1, or 0 and 1, depending on the selected beginning point of the sequence, and each subsequent number is the sum of the previous two

47) **How insertion and selection sort are different?**

48) **Which sorting algorithm is best if array is already sorted?**

If it is already sorted, then selection sorting is the best since it will complete the entire list in a single iteration and hence the lowest time complexity. Insertion sort. Insertion sort is a **simple** sorting algorithm that builds the final sorted array (or list) one item at a time.

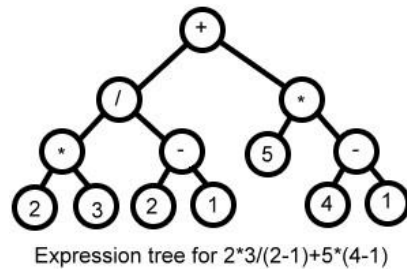
49) **State the difference between stack and linked list?**

**Ans: Stack** is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO (Last In First Out).

A **linked list** is a linear data structure where each element is a separate object. **Linked list** elements are not stored at contiguous location; the elements are **linked** using pointers. Each node of a **list** is made up of two items - the data and a reference to the next node. The last node has a reference to null.

50) **Explain the expression trees?**

**Ans:** An **expression tree** is a representation of **expressions** arranged in a **tree-like data structure**. In other words, it is a **tree** with leaves as operands of the **expression** and nodes contain the operators. Similar to other **data structures**, **data** interaction is also possible in an **expression tree**.



### 51) What is advantage of stack over heap?

Ans:

#### Stack vs Heap Pros and Cons

##### Stack

- very fast access
- don't have to explicitly de-allocate variables
- space is managed efficiently by CPU, memory will not become fragmented
- local variables only
- limit on stack size (OS-dependent)
- variables cannot be resized

##### Heap

- variables can be accessed globally
- no limit on memory size
- (relatively) slower access
- no guaranteed efficient use of space, memory may become fragmented over time as blocks of memory are allocated, then freed
- you must manage memory (you're in charge of allocating and freeing variables)
- variables can be resized using `realloc()`

### 52) What is connected graph?

An undirected **graph** is **connected** when there is a path between every pair of vertices. In a **connected graph**, there are no unreachable vertices. A **graph** that is not **connected** is disconnected.

### 53) What are the eight primitive types in java?

Ans: byte, short, int, long, float, double, char, Boolean

### 54) Differentiate between reference type and primitive type?

Ans: The main difference between primitive and reference type is that *primitive type always has a value*, it can never be null but reference type can be null, which denotes the absence of value. So if you create a primitive variable of type **int** and forget to initialize it then it's value would be 0, the default value of integral type in Java, but a reference variable by default has a null value, which means no reference is assigned to it.

### 55) List five operations that can be applied to a reference type?

**Ans:** Insert, delete, search, sort, assign

**56) List two advantage of linked list over array?**

**Ans:** insertion and deletion are easy in linked list.

**57) What is garbage collection?**

**Ans:** **garbage collection** (GC) is a form of automatic memory management. The **garbage collector**, or just **collector**, attempts to reclaim **garbage**, or memory occupied by objects that are no longer in use by the program.

**58) If a class provide no constructor, what is the result?**

**Ans:** If you don't specify a **constructor**, a **default constructor** will be generated by the compiler. However, any member variable that is **not** initialized **when** it's declared will be null.

**59) What is collection API?**

**Ans:** The **Collections API** provides a number of interfaces (including **Collection**, **List**, **Map** and **Set**) to define a standard way of using a range of concrete data structures. The interfaces and classes of the **Collections API** belong to the **java.util** package.

**60) To solve a problem two algorithms are required to run, first algorithm requires  $O(N^2)$  and second algorithm requires,  $O(N)$ . what is the total cost to solving the problem?**

**Ans:**  $O(N^2) + O(N) = O(N^2)$

**61) List the four fundamental rules for recursion?**

**Ans:**

i. base case    ii.

recursive case

iii. Assume that all recursive calls work.

iv. Never duplicate work by solving the same instance of a problem in separate recursive calls.

**62) Differentiate between worst case and best-case algorithm analysis?**

**Ans:** worst case means an algorithm take maximum time to solve problem while best case means to take minimum time to solve problem.

**63) Differentiate between Queue and priority Queue?**

**Ans:** **Queue is** A list of computer data constructed and maintained on a first in, first out basis.

A **priority queue** is a queue in which each element is inserted or deleted on the basis of their priority. A higher priority element is added first before any lower priority element. If in case priority of two element is same then they are added to the queue on FCFS basis (first come first serve).

64) **List two shortcomings of Dijkstra Algorithms.**

**Ans:** The major disadvantage of the algorithm is the fact that it does a blind search there by consuming a lot of time **waste** of necessary resources. Another disadvantage is that it cannot handle negative edges. This **leads** to acyclic graphs and most often cannot obtain the right shortest path.

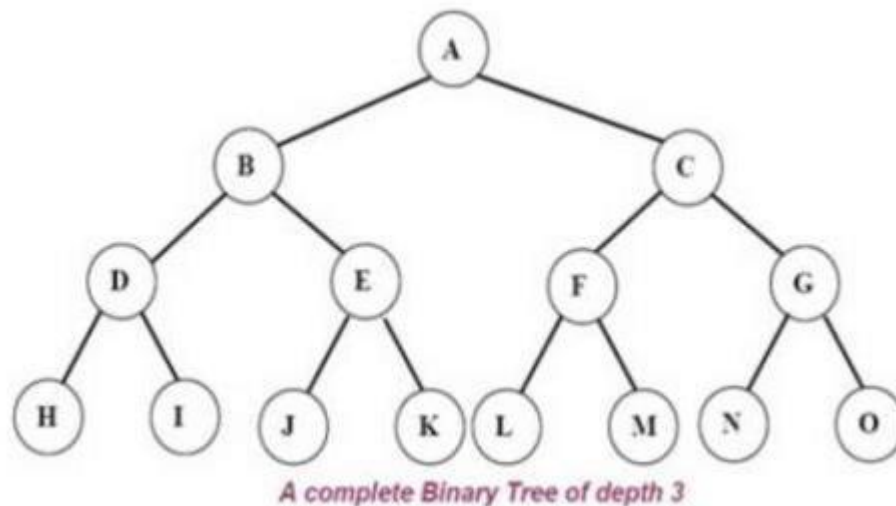
65) **Show the result of following sequence: add(4), add(8), add(1) and add (6). Remove (), and remove (), when the add and remove operations correspond to the basic operations in the following:**

a) **Stack**

b) **Queue**

66) **Define a complete binary tree?**

A complete binary tree is a binary tree in which every level, is completely filled, and all nodes are as far left as possible. A binary tree of depth  $d$  is called complete binary tree if all of whose leaves are at level  $d$ . Every Level of Tree A is completely filled is called complete binary tree.

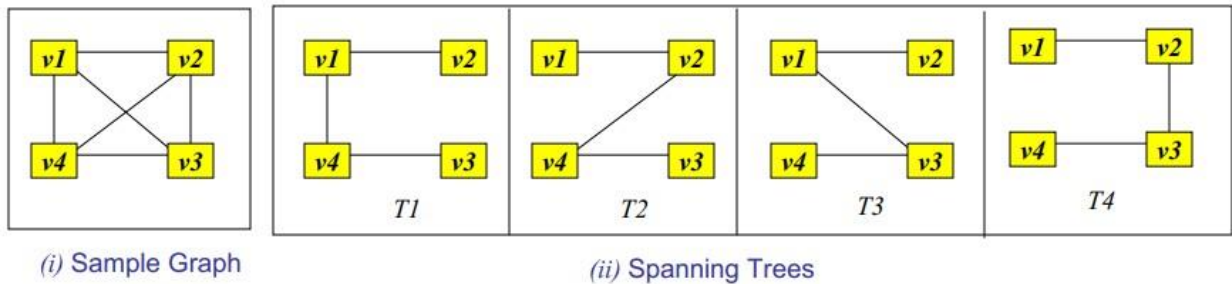


67) **What is spanning tree?**

A spanning tree is a subset of Graph  $G$ , which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected. By this definition, we can draw a conclusion that every connected and



undirected Graph G has at least one spanning tree. A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices.



68) What is linear data structure?

A data structure is classified into two categories: Linear and Non-Linear data structures. A data structure is said to be linear if the elements form a sequence, for example Array, Linked list, queue etc. Elements in a nonlinear data structure do not form a sequence, for example Tree, Hash tree, Binary tree, etc.

69) Define leaf nodes in tree?

Some binary tree implementations store data only at the leaf nodes, using the internal nodes to provide structure to the tree. By definition, a leaf node does not need to store pointers to its (empty) children. In simple we can say, Node which does not have any child node is called leaf node

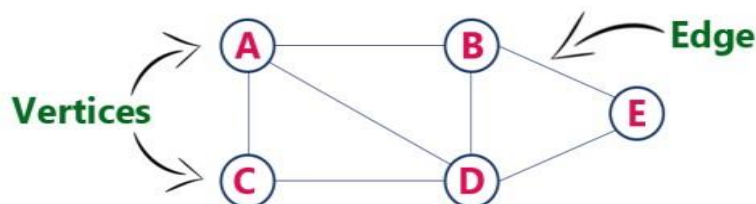
70) When is a binary search best applied?

Binary search is a type of algorithm. It is best applied to search in a list in which the elements are already in order or sorted. The binary search algorithm starts searching the list from the middle. If the middle value is not the correct one, then it will go on to search the top or the bottom half in a similar manner, i.e. it will then divide the top or the bottom part into halves and start searching from its middle. It will continue to do this until the searched for value is found.

71) What operations can be performed on Queues?

72) What is a graph?

A **graph** is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as vertices, and the links that connect the vertices are called edges.



73) Why we need to do algorithm analysis?

74) What is selection sort?

**Ans:** The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning.

The algorithm maintains two subarrays in a given array.

- 1) The subarray which is already sorted.
- 2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

75) What is shell sort?

**Ans:** The **Shell Sort**. The **shell sort**, sometimes called the “diminishing increment **sort**,” improves on the insertion **sort** by breaking the original list into a number of smaller subsists, each of which is **sorted** using an insertion **sort**. By **sorting** the subsets, we have moved the items closer to where they actually belong.

76) What is a heap in Data Structure?

**Ans:** A [heap](#) is a specialized tree-based data structure that satisfied the heap property: if B is a child node of A, then  $\text{key}(A) \geq \text{key}(B)$ . This implies that an element with the greatest key is always in the root node, and so such a heap is sometimes called a max-heap. Of course, there's also a min-heap

77) What is big Oh notation?

**Ans:** Big oh notation is an asymptotic notation and it can be used to calculate worst case of an algorithm  $f(n) \leq C \cdot g(n)$  for all  $n \geq n_0$

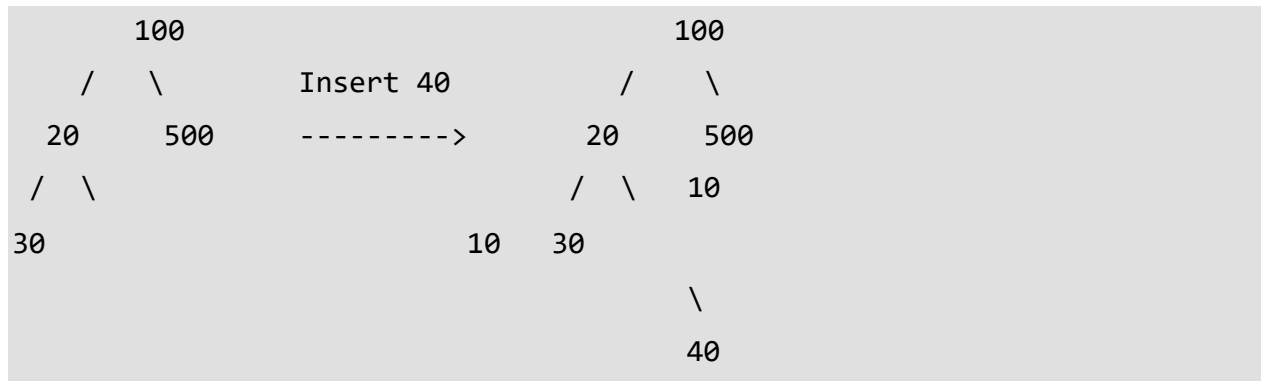
78) What is dynamic Data Structure?

**Ans:** A **dynamic data structure** (DDS) refers to an organization or collection of **data** in memory that has the flexibility to grow or shrink in size, enabling a programmer to control exactly how much memory is utilized

79) There are 8, 15, 13, and 14 nodes in 4 different trees. Which one of them can form a full binary tree? Why?

80) How do you insert a new node in binary search tree?

**Ans:** A new key is always inserted at the leaf node. Start searching a key from root till we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node.



### 81) Differentiate linear from nonlinear Data Structure.

Ans: The main difference between linear and nonlinear data structure is that linear structure arrange data in a sequential manner while nonlinear data structure arrange data in a hierarchical manner, creating a relationship among the data elements. A data structure is a way of storing and managing data.

### 82) Are linked list considered linear or non-linear data structure? Give reasons?

Ans: A linear data structure traverses the data elements sequentially, in which only one data element can directly be reached. I.e. Arrays, linked lists. But in doubly linked list we can reach two data elements using previous pointer and next pointer .so can we say that doubly linked list is nonlinear data structure

### 83) Which data structure is applied when dealing with recursive function?

Because of its LIFO (Last in First Out) property it remembers its 'caller' so knows whom to return when the function has to return. Recursion makes use of system stack for storing the return addresses of the function **calls**. Every recursive function has its **equivalent iterative (non-recursive)** function.

### 84) What is an ordered list?

The **structure** of an **ordered list** is a collection of items where each item holds a relative position that is based upon some underlying characteristic of the item. The ordering is typically either ascending or descending and we assume that **list** items have a meaningful comparison operation that is already defined.

### 85) What is data abstraction?

**Data abstraction** is the reduction of a particular body of **data** to a simplified representation of the whole. **Abstraction**, in general, is the process of taking away or removing characteristics from something in order to reduce it to a set of essential characteristics. **86) In what areas do data structure applied?**

**Data structure** is important in almost every aspect where **data** is involved.

In general, algorithms that involve efficient **data structure** is **applied** in the following **areas**: numerical analysis, operating system, A.I., compiler design, database management, graphics, and statistical analysis, to name a few.

87) What is the minimum number of nodes that a binary tree can have?

The maximum number of nodes in a binary tree of depth  $k$  is  $2^k - 1$ ,  $k \geq 1$ . The maximum number of nodes in a binary tree of depth  $k$  is  $2^k - 1$ ,  $k \geq 1$ . Here the depth of the tree is 1. So according to the formula, it will be  $2^1 - 1 = 1$ .

88) What is the maximum and minimum number of nodes in heap of height 'h'?

The levels above the **lowest** level form a complete binary tree of **height  $h - 1$**  and  $2^{h-1} - 1$  **nodes**. Hence the **minimum number of nodes** possible in a **heap of height  $h$**  is  $2^{h-1}$ . Clearly a **heap of height  $h$** , has the **maximum number** of elements when its **lowest** level is completely filled.

89) What is meant by a sorting algorithm is stable?

A **sorting algorithm** is said to be **stable** if two objects with equal keys appear in the same order in **sorted** output as they appear in the input array to be **sorted**. Informally, **stability means** that equivalent elements retain their relative positions, after **sorting**.

90) Differentiate between data structure and algorithm?

The way I see it is that **algorithms** are something that work with or on **data structures**, so there is a **difference between** the two. An **algorithm** is a set of steps used for accomplishing a task, while a **data structure** is something used to store **data**, the manipulation of said **data** is done with **algorithms**

91) What is time complexity of linear search?

Ans:  $O(n)$

92) What is time complexity of binary search?

$\log_2(n)$

93) What is time complexity of hash table?

$O(1)$

94) What is time complexity of bubble sort?

Ans:  $O(n^2)$

95) What is time complexity of quick sort?

Ans:  $\log(n)$

96) What is time complexity of merge sort?

Ans:  $O(n \log n)$

97) Differentiate between graph and tree?

Tree	Graph
------	-------

A data structure that simulates hierarchal tree structure, which a root value and sub tree of children with a parent node	A data structure that consist of a group of vertices connected through edges
There is a root node	There is no root nod
There are no loops	There can be loops
Representation of data in form of tree structure	Representation of data in form of network
Two major types of tree binary tree and binary search tree	Two major type of graph direct graph and undirected graph
Less complexity	more complexity

98) **Determine whether following is true about postfix expression  $XY-Z = XYZ- -$**

Ans: No, it's not true. It actually false it will  $XYZ- - = -ZYX$

99) **What are advantages and disadvantages of implementing a recursive solution instead of an iterative solution. Advantages of recursion over iterative:**

While iteration may use less memory than a recursive function that can't be optimized, it has some limitations in its expressive power. I personally prefer using Iterative over recursive function. Especially if you function has complex/heavy logic and number of iterations are large.

**Disadvantages of recursion:**

Recursive solution is always logical and it is very difficult to trace. (debug and understand). In recursive we must have an if statement somewhere to force the function to return without the recursive call being executed, otherwise the function will never return. Recursion takes a lot of stack space, usually not considerable when the program is small and running on a PC. Recursion uses more processor time

100) **How many leaves nodes and internal nodes does the full binary tree of  $h=3$  have?**

Ans leave node=4 internal node= 2

101) **Convert the postfix notation into infix notation  $65*432+1-/+$ .**

102) **Sort the given numbers using insertion sort 11, 9, 15, 8,13,**

**Ans: Try your self**

- 103) Suppose that we have numbers between 1 and 1000 in a binary search tree (BST) and we want to search the number 263. Which of the following sequence could not be the sequence of nodes examined?
- a) 2, 252, 401, 398, 330, 344, 397, 363
  - b) 925, 202, 911, 240, 912, 245, 363
  - c) 935, 278, 347, 621, 299, 392, 358, 363
- 104) Suppose that there are few numbers in an array in sorted order and we want to search for number 363 using binary search. Which of the following sequence could be the sequence of access?
- a) 2, 252, 401, 398, 330, 344, 397, 363
  - b) 925, 202, 911, 240, 912, 245, 363
  - c) 935, 278, 347, 621, 299, 392, 358, 363
- 105) How many references are normally used to maintain a queue data structure?
- 106) What is hash table? And why is it used?

**Ans:** A **hash table** is a data structure that is **used** to store keys/value pairs. It uses a **hash** function to compute an index into an array in which an element will be inserted or searched. By using a good **hash** function, **hashing** can work well. Some examples of how hashing is used in our lives include:

- In universities, each student is assigned a unique roll number that can be used to retrieve information about them.
- In libraries, each book is assigned a unique number that can be used to determine information about the book, such as its exact position in the library or the users it has been issued to etc.

- 107) Why to use prefix and postfix notations when we have simple INFIX notation?

**Ans:** Infix notation is easy to read for *humans*, whereas pre-/postfix notation is easier to parse for a machine. The big advantage in pre-/postfix notation is that there never arise any questions like operator precedence.

- 108) Write the name of two sorting technique that used recursion?

**Ans:** Bubble sort and Merge sort

- 109) Write the name of two parameters that define graph?

**Ans:** Graph has vertices and edges.

**110) Which data structure are used to for traversing graph by each algorithm?**

**Ans:** DFS and BFS

**111) Which data structure is applied when dealing with a recursive function? Ans:**

Recursion, which is basically a function that calls itself based on a terminating condition, makes use of the stack. Using LIFO, a call to a recursive function saves the return address so that it knows how to return to the calling function after the call terminates.

**112) Why Dijkstra's algorithm is used?**

**Ans:** Dijkstra's algorithm can be used to determine the single source shortest path from one node in a graph to every other node within the same graph data structure, provided that the nodes are reachable from the starting node. Dijkstra's algorithm can be used to find the shortest path

**113) How many nodes are there in a complete binary tree of level 5?**

**Ans:** if it is a full binary tree with 5 levels i.e., level 0,1,2,3,4. then the number of leaf nodes are  $2^4 = 16$ . but, from the given data it should be equal to 25. that means some of these 16 leaves have children in the next level i.e. each node in level 1-4 has 0 or 2 children.

**114) Define ADT with example?**

**Ans:** Abstract Data Type (ADT) is a data type, where only behavior is defined but not implementation. Opposite of ADT is Concrete Data Type (CDT), where it contains an implementation of ADT. **Examples:** Array, List, Map, Queue, Set, Stack, Table, Tree, and Vector are ADTs.

**115) Why do we used queues?**

**Ans:** Queue is useful in CPU scheduling, Disk Scheduling. When multiple processes require CPU at the same time, various CPU scheduling algorithms are used which are implemented using Queue data structure. When data is transferred asynchronously between two processes. Queue is used for synchronization.

**116) What is B tree?**

**Ans:** A B-tree is a tree data structure that keeps data sorted and allows searches, insertions, and deletions in logarithmic amortized time. Unlike self-balancing binary search trees, it is

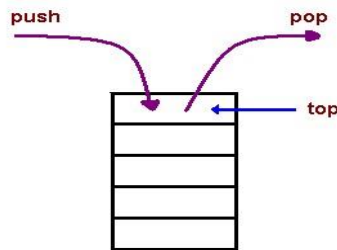
optimized for systems that read and write large blocks of data. It is most commonly used in database and file systems.

117) **What is data type?**

**Ans:** A data type (in computing terms) is a set of data values having predefined characteristics. You will have encountered data types during computer programming classes. **Example:** Data types would be integer, floats, string and characters. Generally, in all programming languages we have a limited number of such data type.

118) **Discuss any application of stack?**

119) **Ans:** The simplest application of a stack is to reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack. Another application is an "undo" mechanism in text editors; this operation is accomplished by keeping all text changes in a stack. **Example:**





120) **What is the index number of the middle element of an array of size 11?**

**Ans: 5**

121) **What do you mean by space complexity?**


122) **Ans:** Space complexity is a measure of the amount of working storage an algorithm needed. That means how much memory, in the worst case, is needed at any point in the algorithm. As with time complexity, we're mostly concerned with how the space needs grow, in big-Oh terms, as the size  $N$  of the input problem grows.

123) **What do you mean by time complexity?**

**Ans:** Time complexity is a concept in computer science that deals with the quantification of the amount of time taken by a set of code or algorithm to process or run as a function of the amount of input. In other words, time complexity is essentially efficiency, or how long a program function takes to process a given input.

124) **What is a circuit? Give example.**

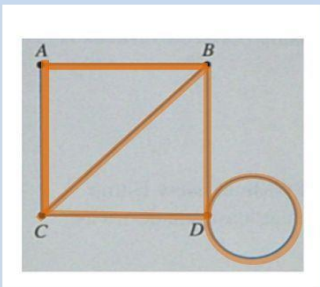
**Ans:** A **circuit** is path that begins and ends at the same vertex.



## Graph Theory

T. Serino

A circuit is a path that begins and ends at the same vertex.



Given the graph to the left, some examples of circuits could be:

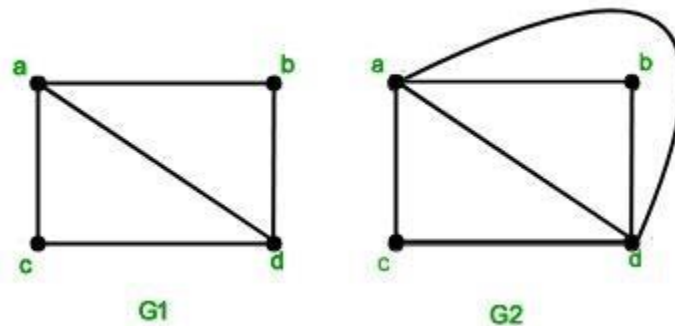
ABCA  
CDDBAC

125) **Define Euler and Hamiltonian Path with example.**

**Ans:** An **Euler path** is a path that uses every edge in a graph with no repeats. Being a path, it does not have to return to the starting vertex.

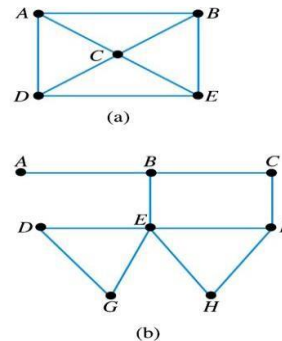
A **Hamiltonian path**, also called a **Hamilton path**, is a graph **path** between two vertices of a graph that visits each vertex exactly once.

**Euler Path:**



### Example: Hamilton Path

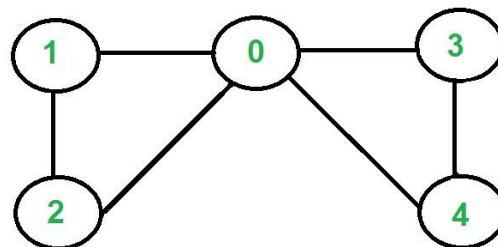
- Graph (a) shown has Hamilton path A, B, C, E, D. The graph also has Hamilton path C, B, A, D, E. Can you find some others?
- Graph (b) shown has Hamilton path A, B, C, F, H, E, G, D. The graph also has Hamilton path G, D, E, H, F, C, B, A. Can you find some others?



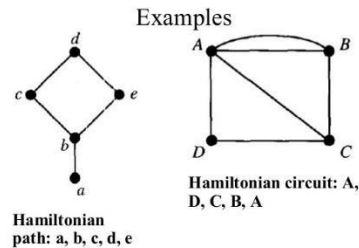
### 126) Define Euler and Hamiltonian circuit and give example.

**Ans:** A **Euler circuit** is a circuit that uses every edge in a graph with no repeats. Being a circuit, it must start and end at the same vertex.

A **Hamiltonian circuit** is a circuit that visits every vertex once with no repeats. Being a circuit, it must start and end at the same vertex.



The graph has Eulerian Cycles, for example "2 1 0 3 4 0 2"  
Note that all vertices have even degree



**127) What is Hashing?**

**Ans:** In computing, a hash table (hash map) is a data structure which implements an associative array abstract data type, a structure that can map keys to values. A hash table uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.

- A technique used to uniquely identify a specific object from a group of similar objects. To convert a range of key values into a range of indexes of an array by using a hash function.

- 128) Define structure of a node used in doubly linked list?
- 129) Which data structure is used in recursion and why?
- 130) Write the name of two prominent operations performed on queue?
- 131) Write the name of two collision resolution techniques used in hashing?
- 132) What are the two necessary conditions of recursion?
- 133) Write the name of two recursive algorithm used in sorting?
- 134) What is strictly binary tree?
- 135) Define merging?
- 136) In which sorting algorithms there are more number of swaps? Selection or Bubble?
- 137) Write the name of sorting algorithm in which a pivot element is selected to sort the array?
- 138) What is a special in Binary search tree?
- 139) How many references are maintained in a queue?
- 140) Write the different categories of strings?
- 141) What are pointers?
- 142) How do you understand hash tale?
- 143) Define acyclic graph with example.
- 144) What is circular queue? Why we use it?

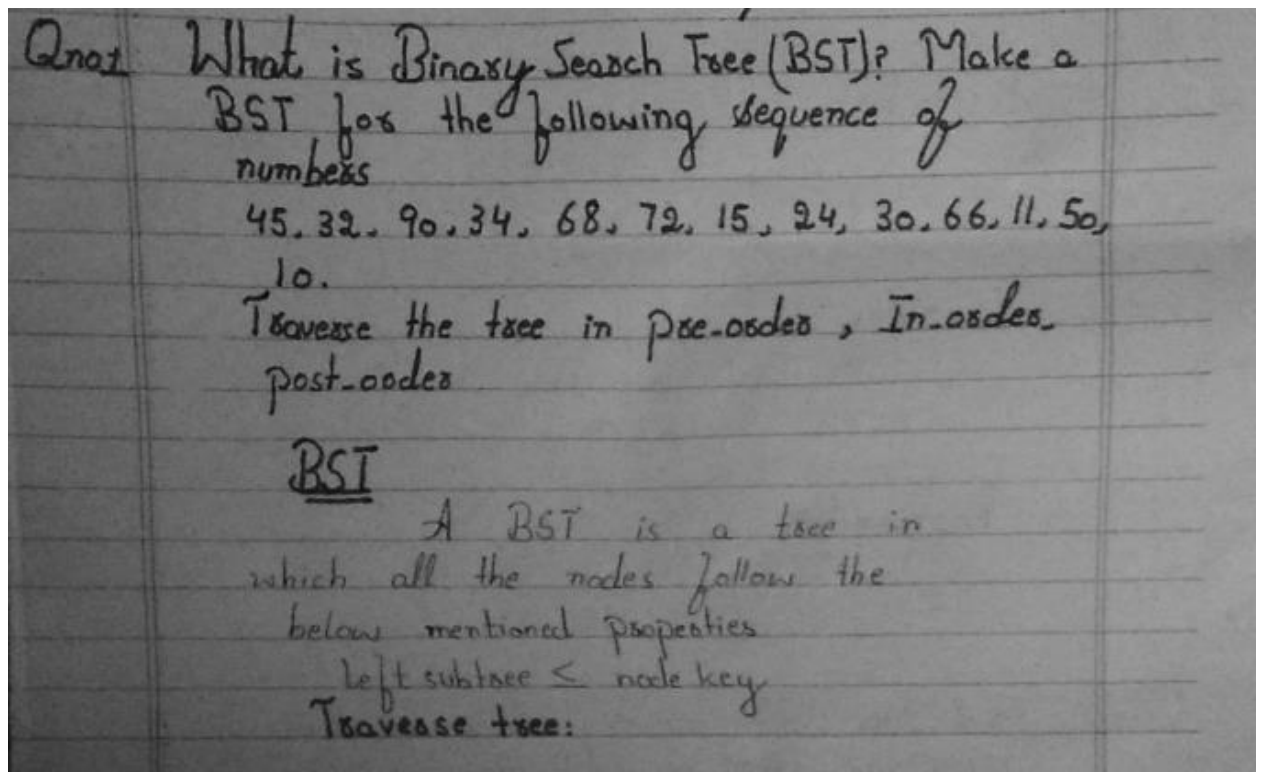
- 145) How to find total number of nodes?
- 146) What is complete binary search tree?
- 147) What is tree traversal? How many traversals of binary tree are possible?
- 148) When we should not use sequential search?
- 149) What are different methods to represent a graph?
- 150) What is adjacency list?
- 151) Define full binary tree?

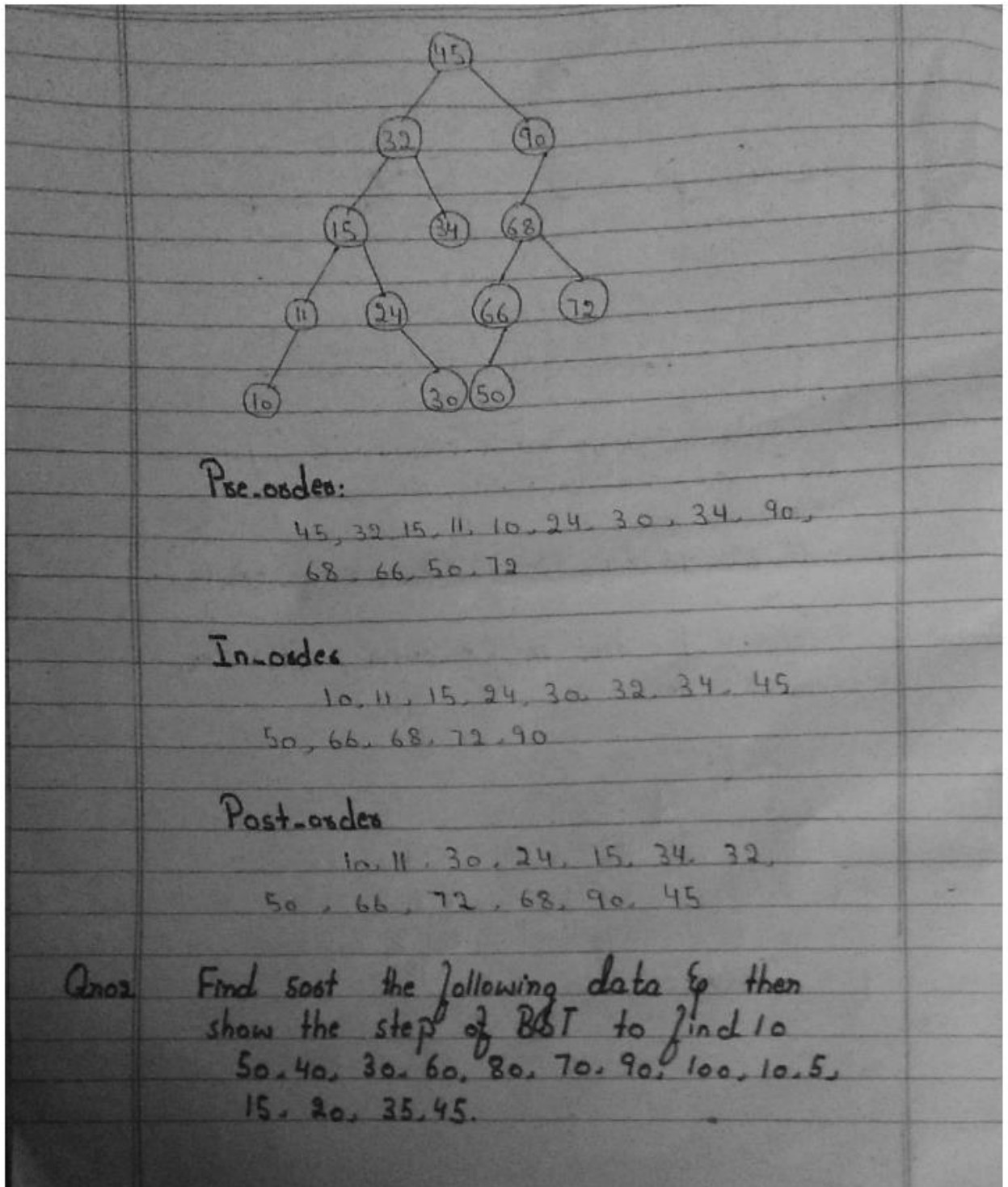
### Long Questions

- 1) What is Binary Search Tree (BST)? Make a BST for the following sequence of numbers.

45,32,90,34,68,72,15,24,30,66,11,50,10

Traverse the tree in **Preorder**, **Inorder**, and **Postorder**.





- 2) First sort the following data and then show the steps of Binary search to find 10 from the following data.

50, 40, 30, 60, 80, 70, 90, 100, 10, 5, 15, 20, 35, 45

Using Binary search:

5	10	15	20	30	25	40	45	50	60	70	75
								80	90	100	

1) Find mid  
$$\text{mid} = (L+H)/2$$
$$= (0+13)/2$$

5	10	15	20	30	35	40	45	50	60	70	80	90	100
---	----	----	----	----	----	----	----	----	----	----	----	----	-----

2) Take first half because  $10 < 40$

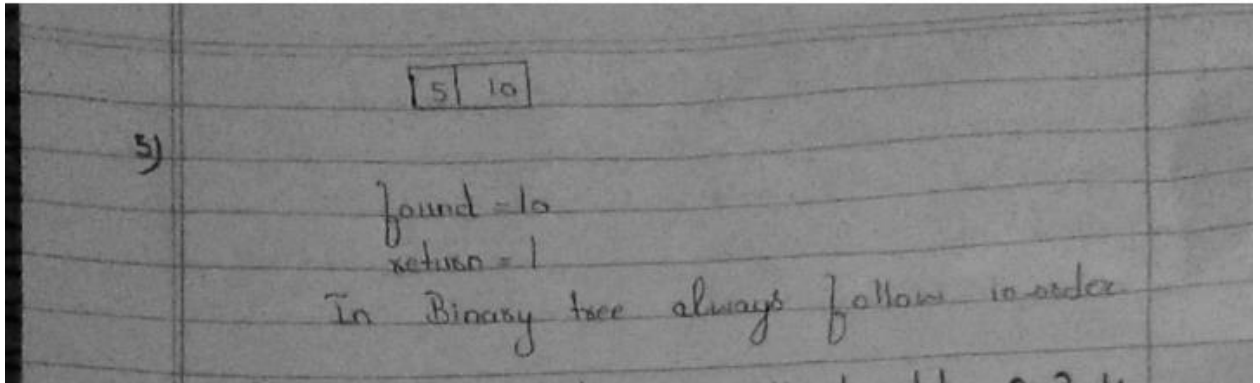
5	10	15	20	30	35
---	----	----	----	----	----

3 Repeat step 1  
$$\text{mid} = (L+H)/2$$
$$= (0+5)/2$$
$$\text{mid} = 2$$

5	10	15	20	30	35
---	----	----	----	----	----

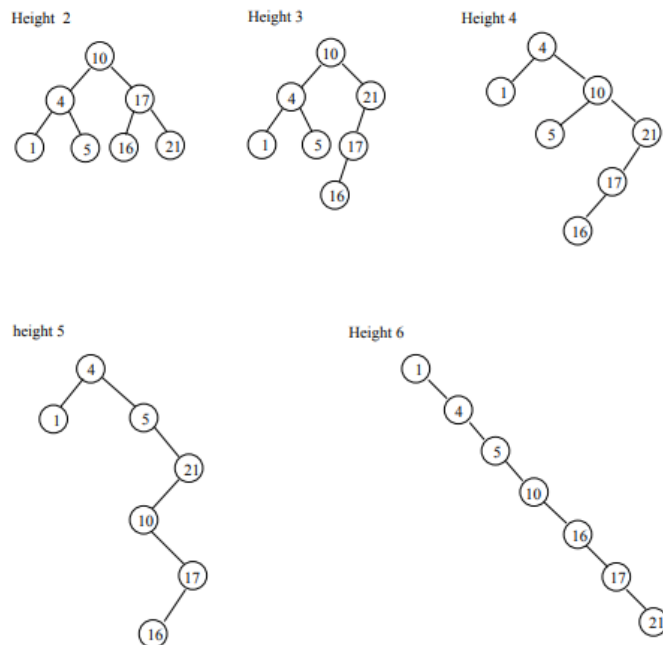
5	10		15		20	30	35
---	----	--	----	--	----	----	----

4)  $10 < 15$   
Take I<sup>st</sup> half  
 $10 < 15$



- 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, and 21.

For the set of keys {1, 4, 5, 10, 16, 17, 21}, draw binary search trees of height 2, 3, 4, 5, and 6.



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

4) Given an array  $A = \{12, 11, 13, 5, 6\}$ . Sort it out using a technique illustrated in insertion sort. You have to discuss only passes in detail & there is no need to write an algorithm of insertion sort.

Initial array:  $[12, 11, 13, 5, 6]$   $i=1$

Temp = 11

After first pass:  $[11, 12, 13, 5, 6]$   $i=2$



Temp = 5

11	12	5	13	6
----	----	---	----	---

Temp = 6

11	5	12	13	6
----	---	----	----	---

5	11	12	13	6
---	----	----	----	---

Temp = 6

5	11	12	6	13
---	----	----	---	----

5	11	6	12	13
---	----	---	----	----

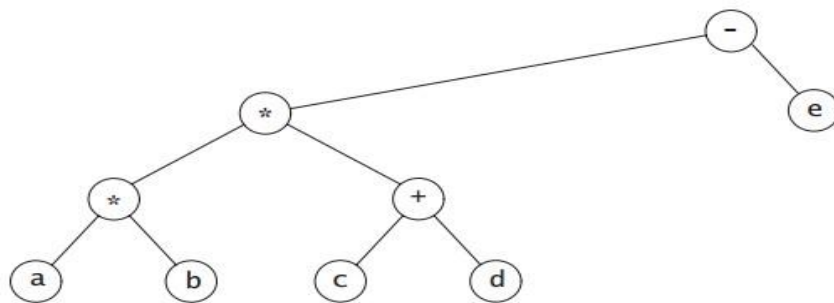
5	6	11	12	13
---	---	----	----	----

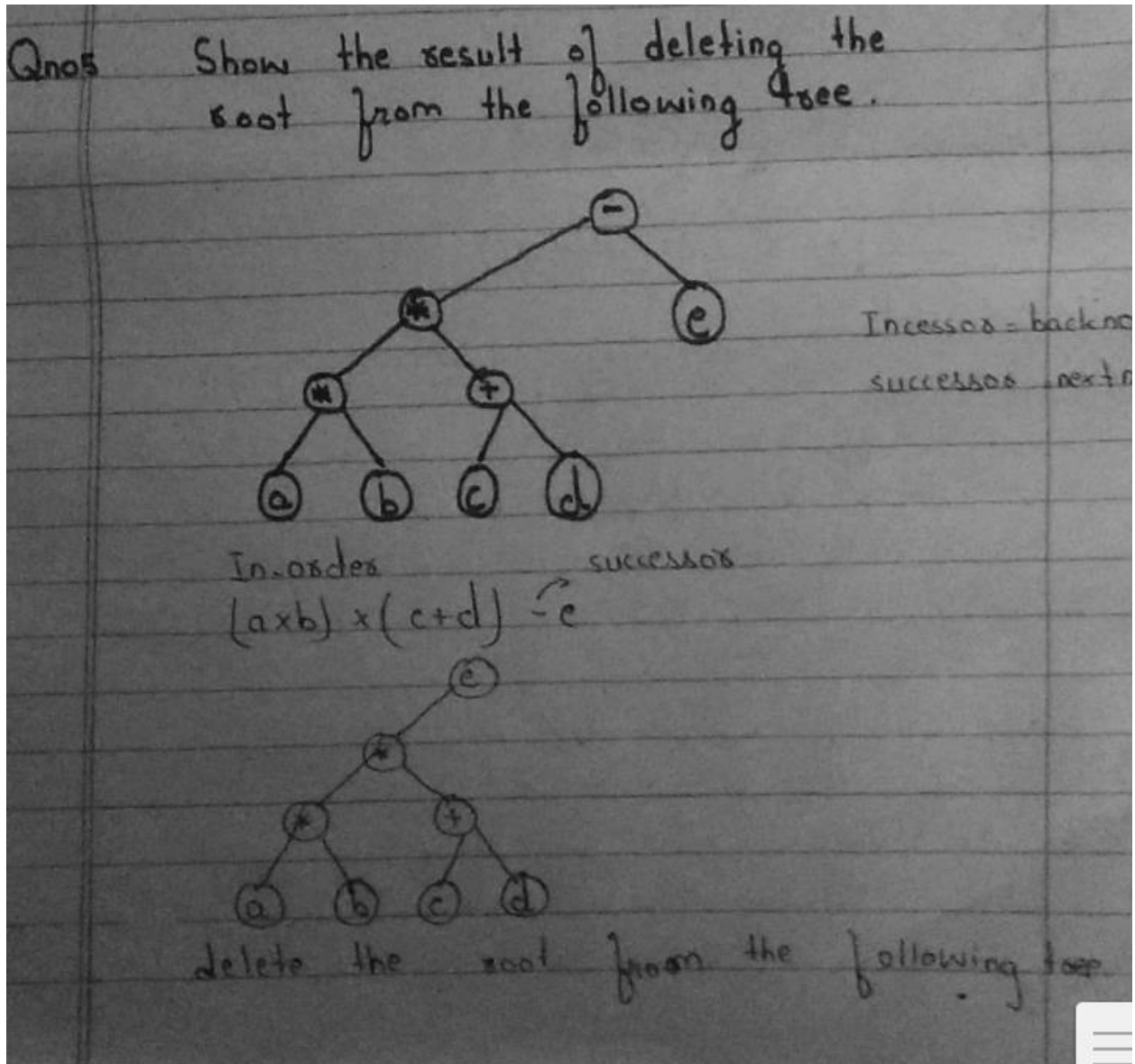
Sol

- 1 let us loop for  $i = 1$  to 5 (size of array)  
 $i = 1$  since 11 is smaller than 12  
move 12 & insert 11 before 12  
such as 11, 12, 13, 5, 6
- 2  $i = 2$  13 will be remain at its position as all elements in array are smaller than 13  
such as 11, 12, 13, 5, 6
- 3  $i = 3$  5 will move to the beginning & all other element from 11 to 13 will move one

position a head their current position.  
such as 5, 11, 12, 13, 6  
4 i-4, 6 will move to position after 5 and elements from 11 to 13 will move one position ahead of their current position  
such as 5, 6, 11, 12, 13.

5) Show the result of deleting the root from the following tree.





6) Write a program/algorithm to do the following operations.

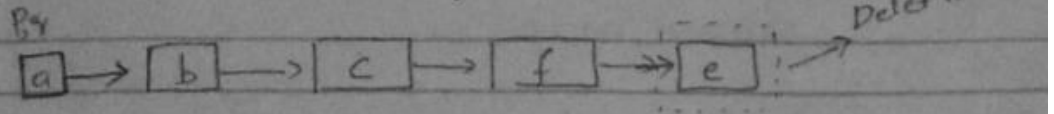
- Delete a node at the end of a single linked list.
- Insert the first node in the circular link list.

Q#06

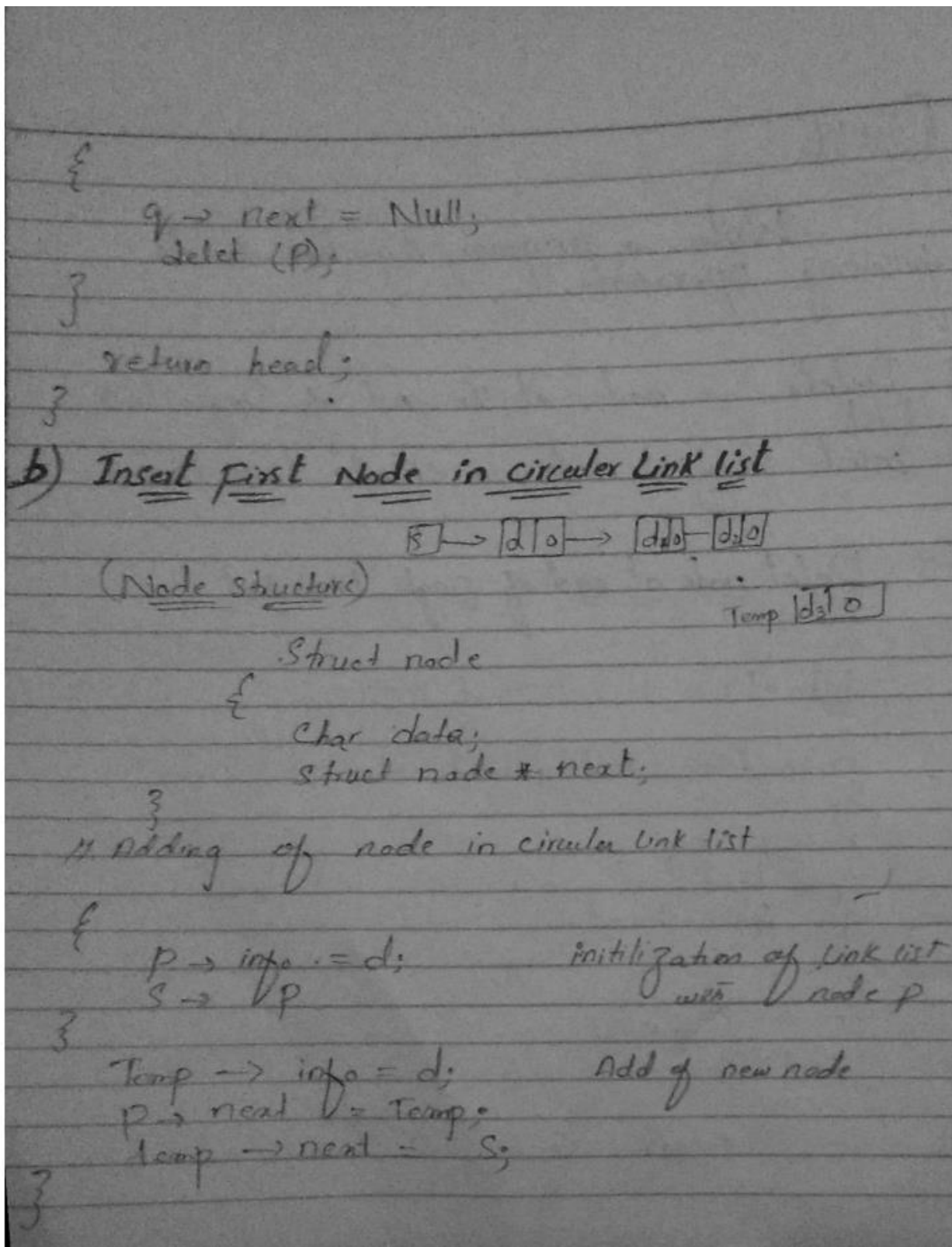
Write a program / Algorithm to do following operations.

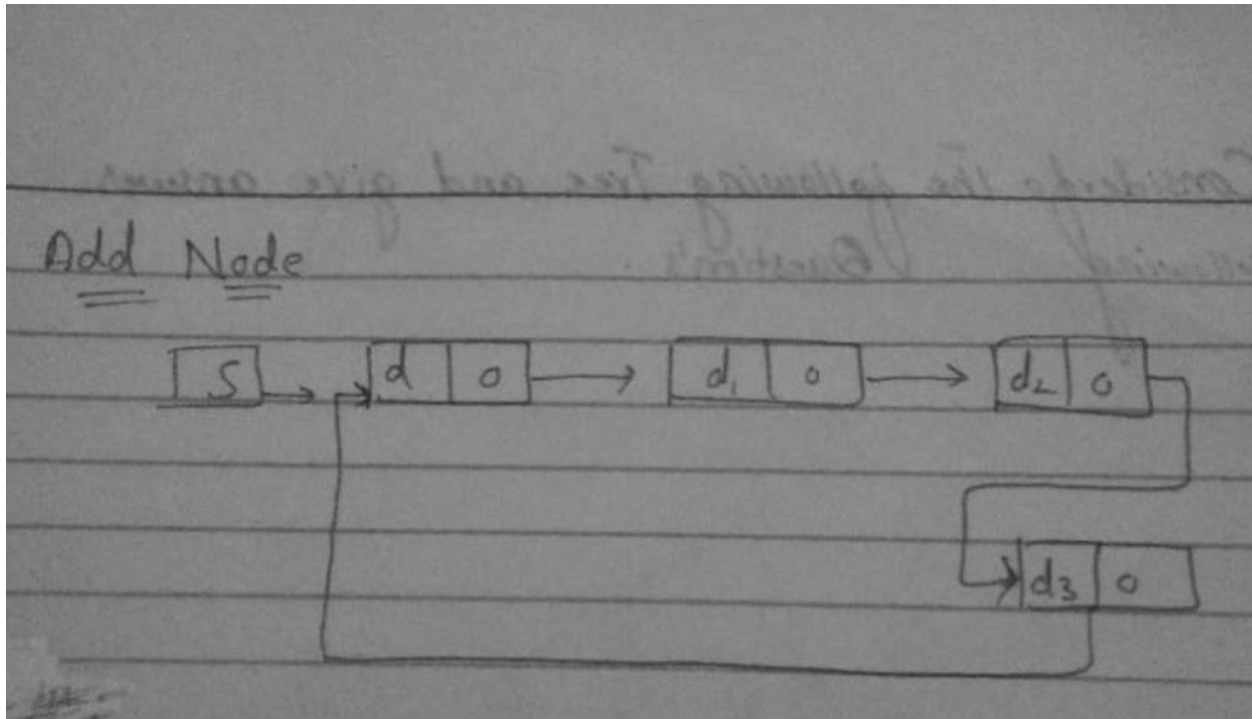
- Delete a node at the end of Single Link List
- Insert the first node in circular Link List

⇒ Delete node at end of Single Link List

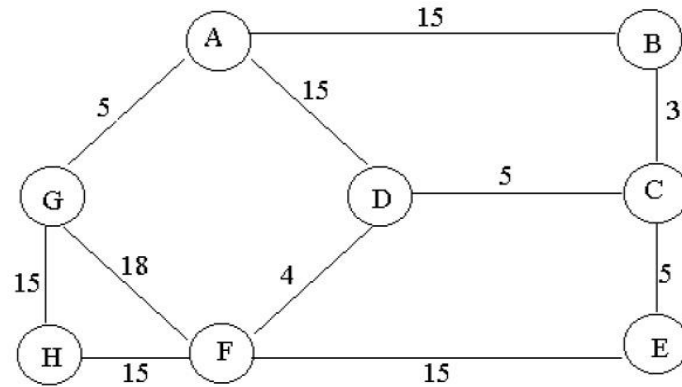


```
node* delete (node* head, char d)
{
    node* p = q;
    q = head;
    p = q->next;
    while (p != NULL)
    {
        p = p->next;
        q = p;
    }
    if (q->next == NULL) // Last node
```





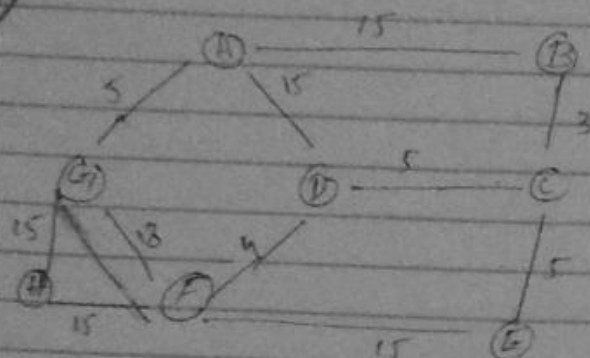
- 7) Traverse the following graph using BFS algorithm. Answer should contain Queue at each step and parameters values of vertices (i.e parent of vertex and distance from source node).



(Take A to be the source node)

Q#7

Traverse following Graph using  
BFS algorithm. Answer should contain  
Queue at each step and  
parameter value of vertex and Distance  
from source node



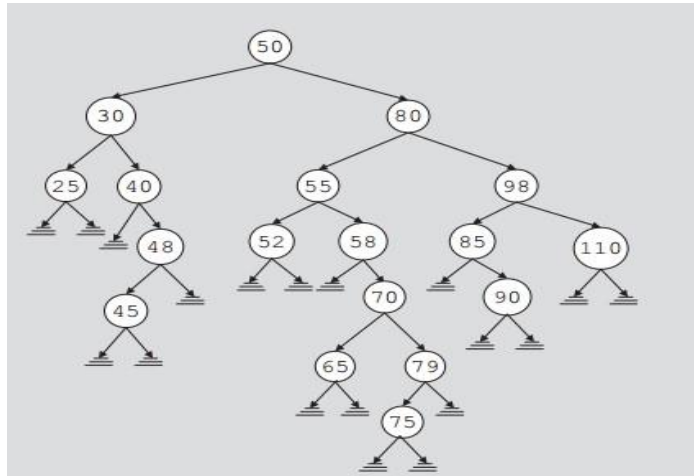
B		d		g		e		F		h				
d		g		c		F		h		E				
g	→	c	→	F	→	h	→	E	→		→	E	→	

output

A → B → d → g → c → F → h → e



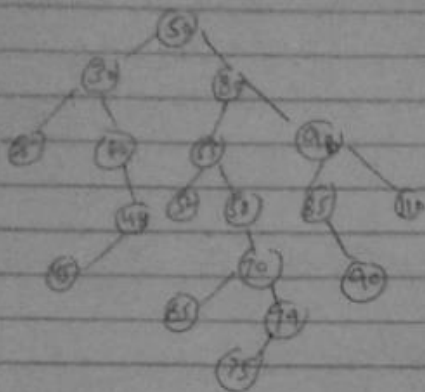
8) Consider the following tree and give answer for the following questions.



- List the path from the node with **info 80** to the node with info **79**.
- A node with info **35** is to be inserted in the tree. List the nodes that are visited by the function **insert** to insert **35**. Redraw the tree after inserting **35**.
- Delete node **52** and redraw the binary tree.
- Delete node **40** and redraw the binary tree
- Delete nodes **80** and **58** in that order. Redraw the binary tree after each deletion.



Q#8 Consider the following Tree and give answer for following Question's.



(a) List the path from node with info 80 to node with info 79

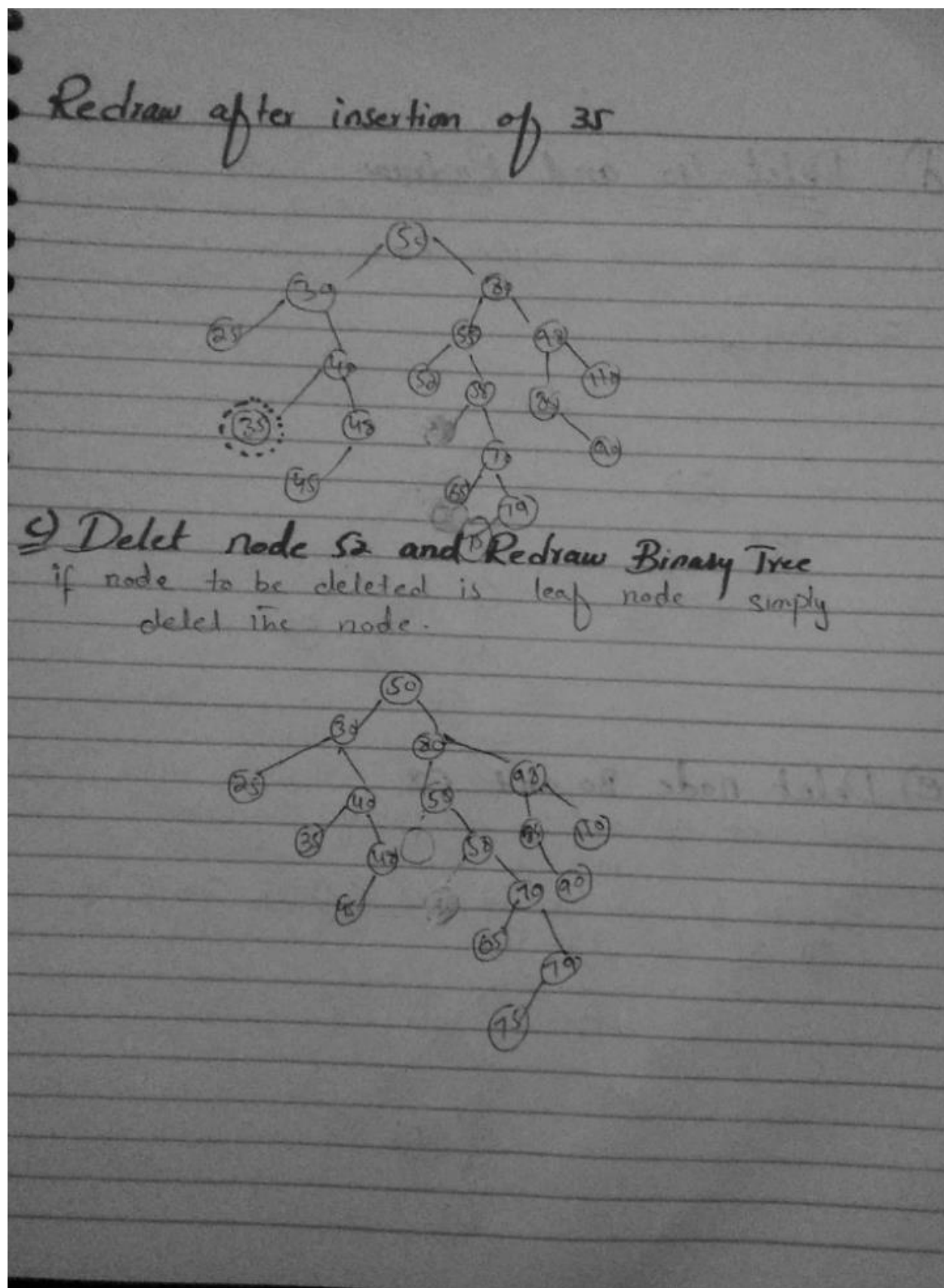
⇒ 80, 55, 52

⇒ 80, 55, 58, 70, 65

⇒ 80, 55, 58, 70, 79

B) A node with info 35 is insert and Redraw after inserted.

Visit left Sub Tree and insert 35

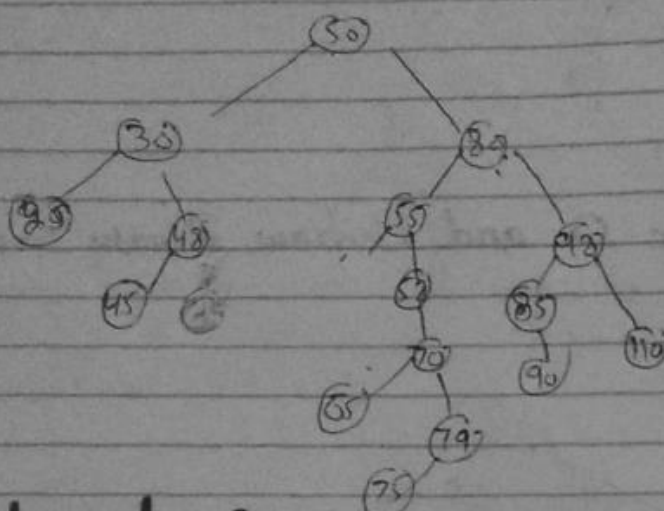




d) Delete 40 and Redraw

delete 40 and apply inorder

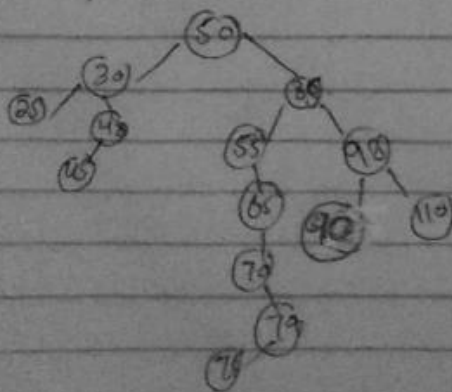
25, 30, 40 → 48, 45



e) Delete node 80 and 58

delete node 80 and 58 and apply inorder

25, 30, 50, 45, 48, 52, 55, 58 65, 70, 75, 79  
80 85, 90, 98, 110

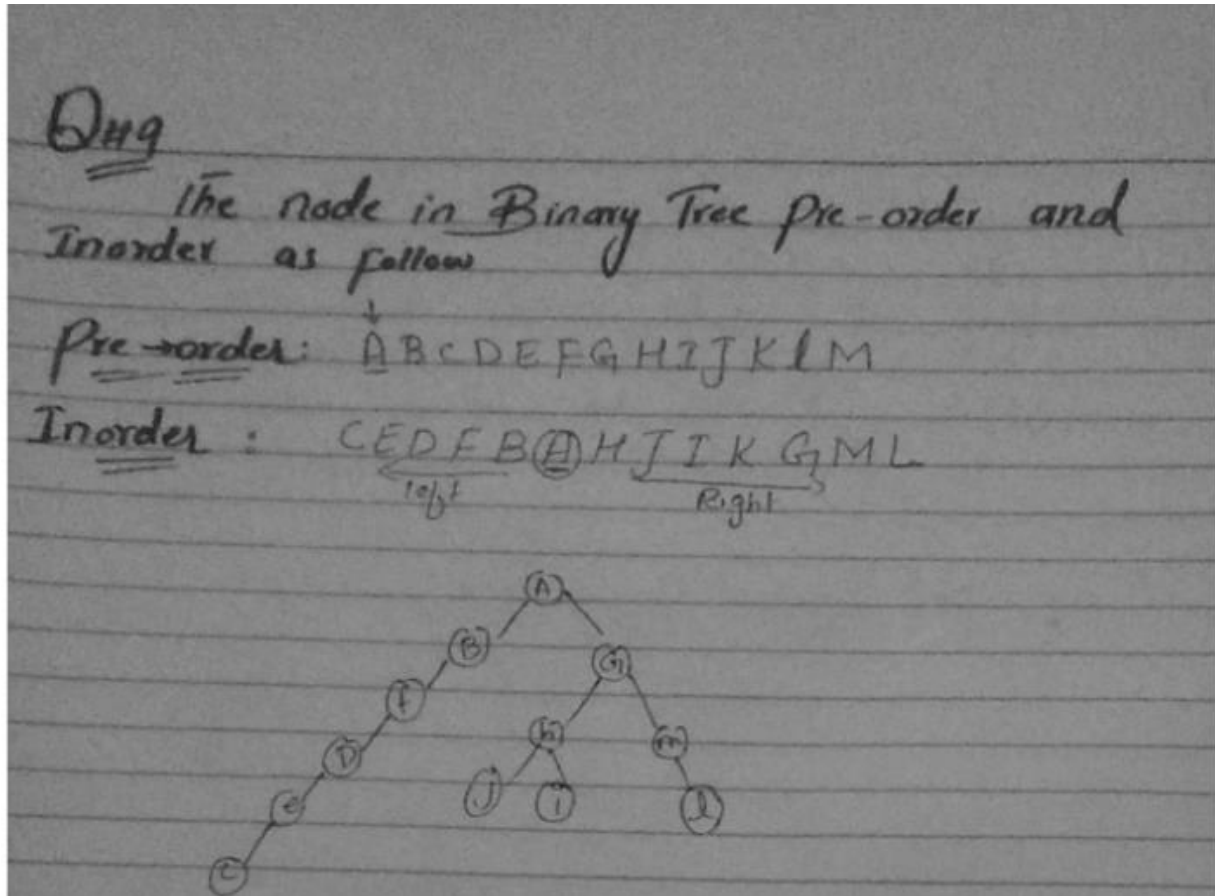


9) The nodes in a binary tree in Preorder and In order sequences are as follows:

**Pre order: ABCDEFGHIJKLM in**

**order: CEDFBAHJIKGML**

Draw the binary tree.



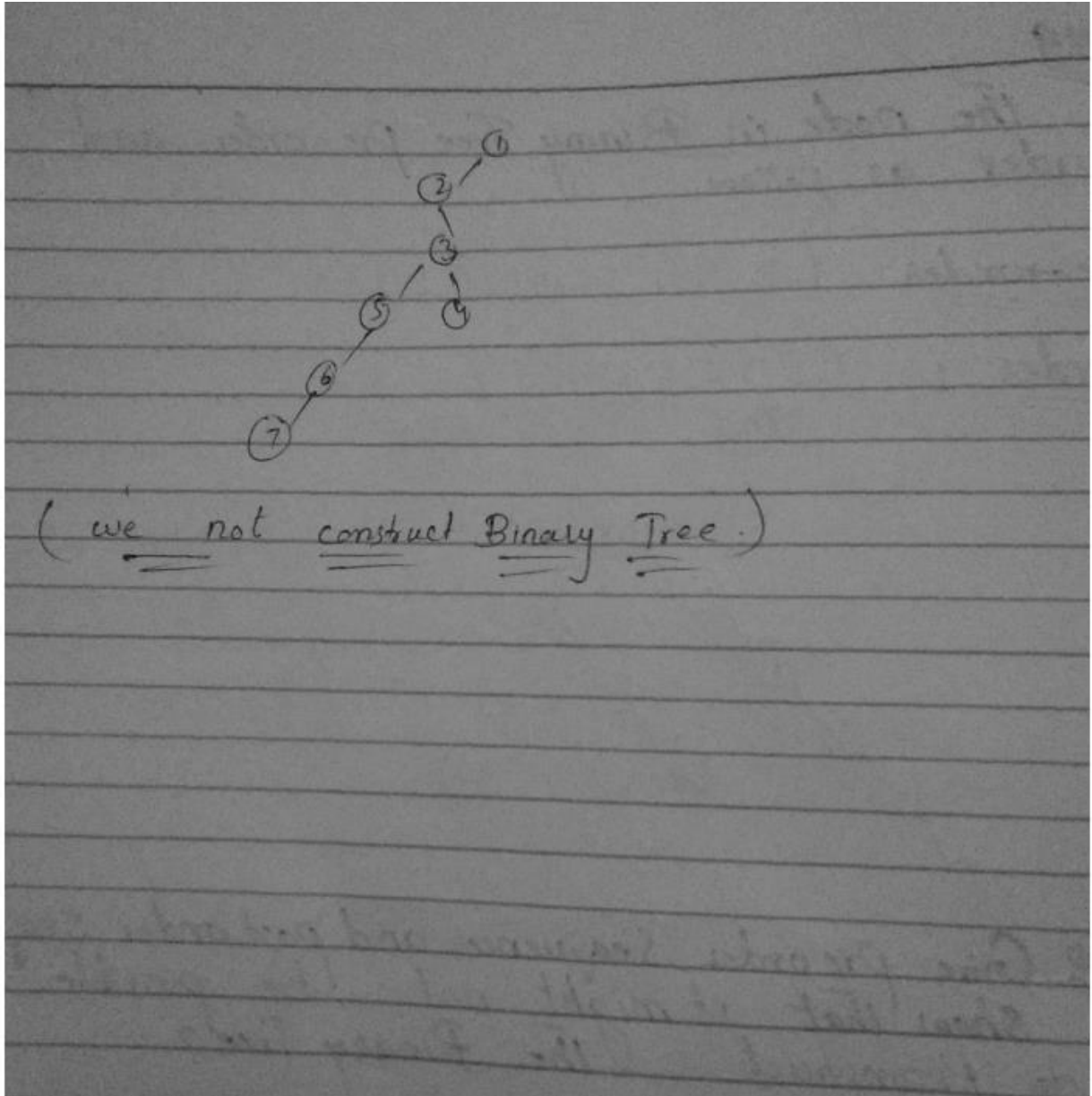
10) Given the preorder sequence and the post order sequence, show that it might not be possible to reconstruct the binary tree.

Q10 Give pre order sequence and post order sequence  
Show that it might not be possible  
to Reconstruct the Binary Tree?

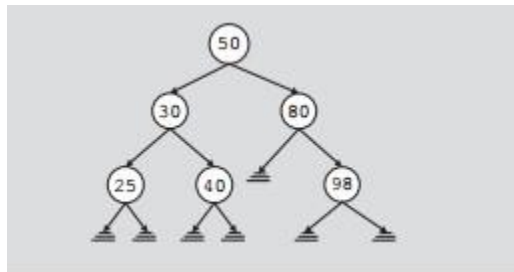
```
graph TD; 1((1)) --> 2((2)); 1 --> 3((3)); 2 --> 4((4)); 4 --> 5((5)); 5 --> 6((6)); 5 --> 7((7));
```

Pre order : ①, 2, 4, 5, 6, 7

Post order : 2 4 6, 7 5 3 ①



- 11) Insert 100 in the AVL tree of given figure. The resulting tree must be an AVL tree. What is the balance factor at the root node after the insertion?





Long Question's

Q#11 Insert 100 in AVL tree of given figure. The Resulting Tree must be an AVL Tree

⇒ What is Balance factor at the root node after insertion?

Rotation's  $[-1, 0, +1]$

LL  
RR  
LR  
RL

After insertion

Left Right Rotation

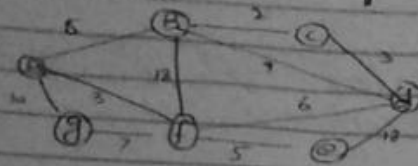
Balance factor of Root =  $2 - 2 = 0$

12) Write a program to solve the single-source shortest-path problem.



Q# 12

Write a program to Solve Single Source Shortest path problem?



	a	b	c	d	e	f	g
a	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
b	5	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
c	12	5	0	$\infty$	9	8	$\infty$
d	3	5	7	0	8	$\infty$	10
e	6	5	7	9	0	10	$\infty$
f	6	5	7	9	10	0	$\infty$
g	6	5	7	9	10	$\infty$	0

Single source shortest path

Dijkstra (a, w, s)


```

{
    initialize Single Source (G, s)
    S ← ∅
    Q ← V[G]
    while (Q ≠ ∅)

```

```
do
     $u \leftarrow \text{extract-min}(Q)$ 
     $S \leftarrow S \cup \{u\}$ 
    for each vertex  $v \in \text{adj}(u)$ 
    do
        Relax( $u, v, w$ )

initialize single source ( $G, s$ )
for each vertex  $v \in V(G)$ 
     $d[v] \leftarrow \infty$ 
     $\pi[v] \leftarrow \text{NIL}$ 
     $d[s] \leftarrow 0$ 
}
Relax( $u, v, w$ )
{
    if ( $d[v] > d[u] + w(u, v)$ )
    {
         $d[v] \leftarrow d[u] + w(u, v)$ 
         $\pi[v] \leftarrow u$ 
    }
}
```



13) Consider the following array and sort this array using.

a) Bubble sort

**Prepared by Rana Asim**

- b) Merge sort
  - c) Insertion sort
  - d) Shell sort
  - e) Selection sort
  - f) Quick sort
- A= {50, 40, 30, 10, 20, 60}

And also write their program/algorithm.

Q#313

Consider the following array and sort this array

- a) Bubble Sort
- b) Merge Sort
- c) Insertion Sort
- d) Shell Sort
- e) Selection Sort
- f) Quick Sort

Bubble Sort  $O(n^2)$

1st phase

50	40	30	10	20	60
----	----	----	----	----	----

40	50	30	10	20	60
----	----	----	----	----	----

40	30	50	10	20	60
----	----	----	----	----	----

40	30	10	50	20	60
----	----	----	----	----	----

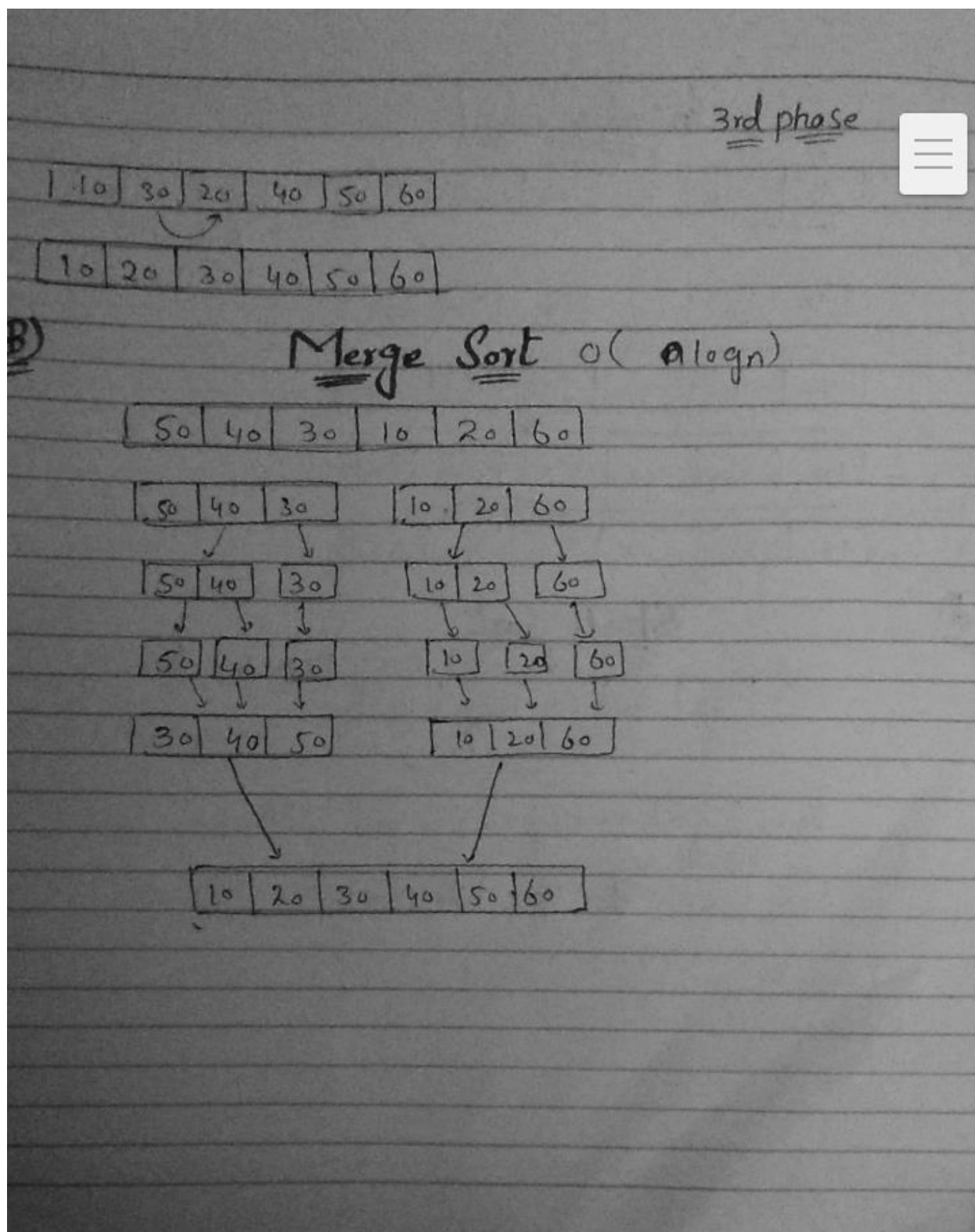
40	30	10	20	50	60
----	----	----	----	----	----

40	30	10	20	50	60
----	----	----	----	----	----

2nd phase

30	10	40	20	50	60
----	----	----	----	----	----

30	10	20	40	50	60
----	----	----	----	----	----



c)

### Insertion Sort

0	1	2	3	4	5
50	40	30	10	20	60

→

$$(n-1 = 5)$$
$$6-1 = 5$$

40	50	30	10	20	60
----	----	----	----	----	----

→

30	40	50	10	20	60
----	----	----	----	----	----

→

10	30	40	50	20	60
----	----	----	----	----	----

→

10	20	30	40	50	60
----	----	----	----	----	----

d)

### Shell Sort

0	1	2	3	4	5
50	40	30	10	20	60

↑ ↑ ↑

Phase 1  $\frac{6}{2} = 3$

10	20	60	50	40	30
----	----	----	----	----	----

→

Phase 2  $\frac{3}{2} = 1$

10	20	30	40	50	60
----	----	----	----	----	----

e Selection Sort

50	40	30	10	20	60
----	----	----	----	----	----

↑

10	40	30	50	20	60
----	----	----	----	----	----

↑

10	20	30	50	40	60
----	----	----	----	----	----

↑

10	20	30	50	40	60
----	----	----	----	----	----

↑

10	20	30	40	50	60
----	----	----	----	----	----

i Quick Sort (Time complexity  $\log(n)$ )

50	40	30	10	20	60
----	----	----	----	----	----

i j Pivot = 50

20	40	30	10	50	60
----	----	----	----	----	----

j

20	10	30	40	50	60
----	----	----	----	----	----

↑ Pivot = 20

20	10	30	40	50	60
----	----	----	----	----	----

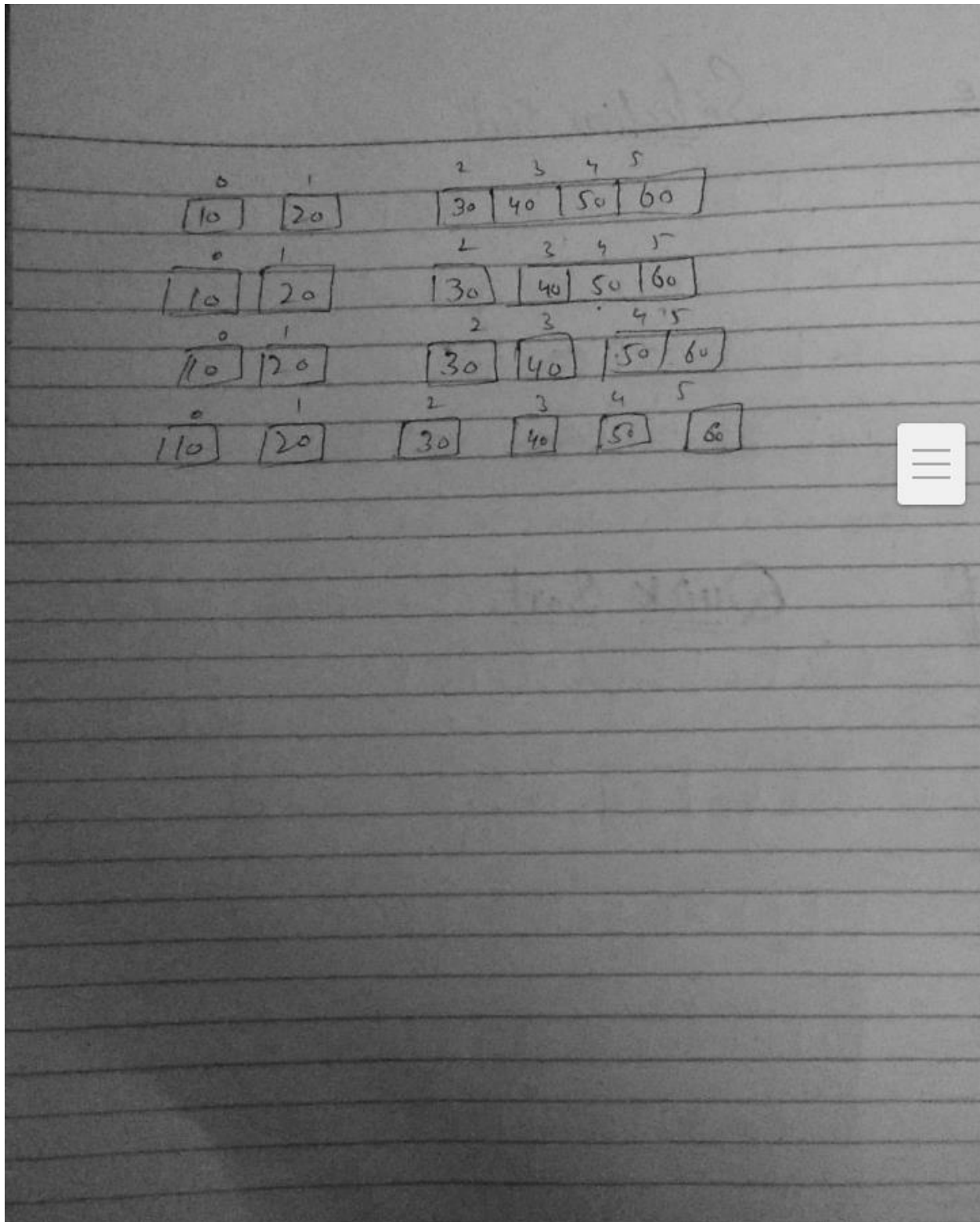
↑

10	20	30	40	50	60
----	----	----	----	----	----

↑

10	20	30	40	50	60
----	----	----	----	----	----





14) Insert the following keys in hash table using hash function  $h(x) = (2x+3) \bmod 10$

7, 10, 11, 17, 20, 22, 21, 25, 35, 42



Resolve collision using

(a) Chaining method (b) Linear Probe (c) Quadratic Probe

Q# 14

Insert following key in hash Table

Using Hash function  $h(x) = (2x+3) \bmod 10$

7, 10, 11, 17, 20, 22, 21, 25, 35, 42

Hash Table

Index		Time complexity
0		
1		
2		
3	10, 20, 25, 35 Collision	$O(1)$
4		
5	11, 21 Collision	
6		
7	7, 17, 22, 42 Collision	
8		
9		
10		

$h(x) = (2x+3) \bmod 10$

$\Rightarrow (2(7)+3) \bmod 10$   
 $17 \bmod 10$   
 $= 7$

$\Rightarrow 2(10)+3 \bmod 10$   
 $23 \bmod 10$   
 $= 3$

$\Rightarrow 2(11)+3 \bmod 10$   
 $25 \bmod 10$   
 $= 5$

$\Rightarrow 2(17)+3 \bmod 10$   
 $37 \bmod 10 = 7$

$\Rightarrow 2(20)+3 \bmod 10$   
 $43 \bmod 10 = 3$

$\Rightarrow 2(21)+3 \bmod 10$   
 $45 \bmod 10 = 5$

$\Rightarrow 2(25)+3 \bmod 10$   
 $53 \bmod 10 = 3$

$\Rightarrow 2(35)+3 \bmod 10$   
 $73 \bmod 10 = 3$

$\Rightarrow 2(42)+3 \bmod 10$   
 $87 \bmod 10 = 7$

## Resolve Collision

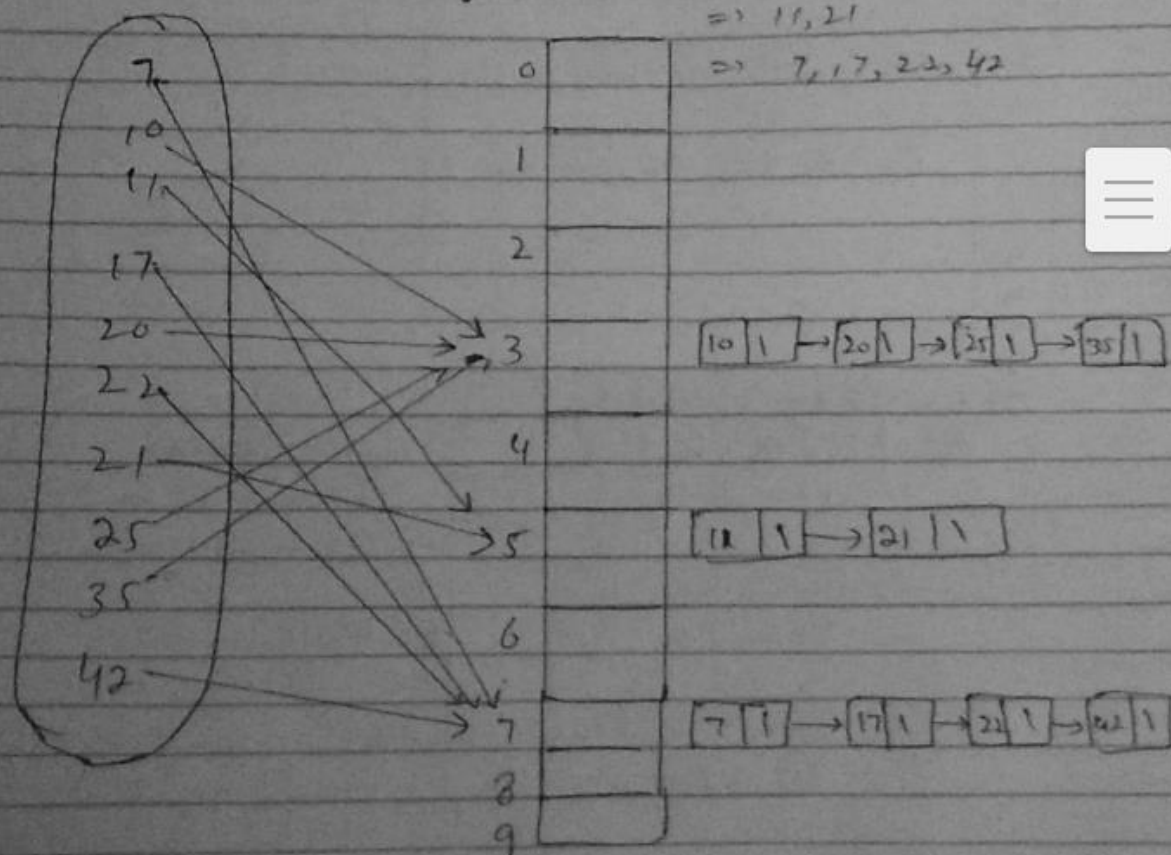
- a) Chaining Method
- b) Linear probe
- c) Quadratic probe

### Chaining Method

$\Rightarrow 10, 20, 25, 35$

$\Rightarrow 11, 21$

$\Rightarrow 7, 17, 22, 42$



b) Linear probe  $h(x) = x \% \text{Size}$

7	0	25
10	1	35
17	2	42
20	3	3
22	4	20
21	5	11
25	6	21
35	7	7
42	8	17
	9	22

3  $\Rightarrow$  10, 20, 25, 35,

5  $\Rightarrow$  11, 21

7  $\Rightarrow$  7, 17, 22, 42

c

## Quadratic probe

$$h(\text{key}) = \text{hash}(\text{key}) + i^2 \pmod{10}$$

7, 10, 11, 17, 20, 22, 21, 25, 35, 42

0	42
1	22
2	25
3	10
4	20
5	11
6	21
7	7
8	17
9	35

$$h(x) = (2x+3) i^2 \pmod{10}$$
$$37 + i^2 \pmod{10}$$
$$= 8$$

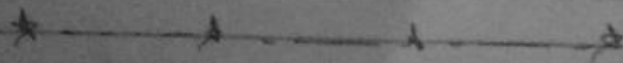
$$\Rightarrow 2(20+3) + i^2 \text{ mod } 10 \\ 43 + 1 \text{ mod } 10 \\ = 4$$

$$\Rightarrow 2(22+3) + i^2 \text{ mod } 10 \\ 47 + 4 \text{ mod } 10 \\ 51 \text{ mod } 10 = 1$$

$$\Rightarrow 2(21+3) + i^2 \text{ mod } 10 \\ 46 \cdot \text{mod } 10 \\ = 6$$

$$\Rightarrow 2(35+3) \cdot 6 \text{ mod } 10 \\ 73 + 4656 \text{ mod } 10 \\ 46729 \text{ mod } 10 \\ = 9$$

$$\Rightarrow 2(42+3) + i^2 \text{ mod } 10 \\ 87 + 7^7 \text{ mod } 10 \\ = 823636 \text{ mod } 10 \\ = 0$$



- 15) The following algorithm is supposed to compute the product of the elements of its input array. Prove that the algorithm is correct.

**Input:** Array A [1,....., n],  $n \geq 1$

**Output:** product of the array's elements PROD(A)

1. Tulo = 1
2. I=1
3. While  $i \leq n$
4. Tulo = tulo\* A[i]
5.  $i = i + 1$
6. return tulo

- 16) Algorithms **A** and **B** sort their input arrays. Algorithm **A** performs  $32 \times n \log(n)$  operations and algorithm **B**  $3 \times n^2$  operations, when the array is of size n. Figure out, when to use algorithm **A** and when to use algorithm B if the size of the array is known.

### **$O(n^2)$ algorithms**

#### **Bubble Sort**

The algorithm works by comparing each item in the list with the item next to it, and swapping them if required. In other words, the largest element has bubbled to the top of the array. The algorithm repeats this process until it makes a pass all the way through the list without swapping any items.

```
void bubble Sort(int ar[])
{
    for (int i = (ar.length - 1); i >= 0; i--)
    {
        for (int j = 1; j ≤ i; j++)
        {
            if (ar[j-1] > ar[j])
            {
                int temp = ar[j-1];
                ar[j-1] = ar[j];
                ar[j] = temp;
            }
        }
    }
}
```

### **$O(n \log n)$ algorithms**

#### **Merge sort**

Merge-sort is based on the divide-and-conquer paradigm. It involves the following three steps:

- Divide the array into two s(or more) subarrays
- Sort each subarray (Conquer)
- Merge them into one (in a smart way!)

17) Write a program to insert or delete item from a circular queue.

## Algorithm for Insert Operation

```
Step 1: If REAR = SIZE-1 then  
    REAR = 0  
Else  
    REAR=REAR + 1  
Step 2: If FRONT = REAR then  
    Write ("Circular Queue Overflow")  
Step 3: CQ[REAR]=X  
Step 4: If FRONT = -1 then  
    FRONT=0
```

## Algorithm for Delete Operation

```
Step 1: If FRONT = -1 then  
    Write ("Circular Queue Underflow")  
Step 2: Return (CQ [FRONT])  
Step 3: If FRONT = REAR then  
    FRONT=REAR=-1  
Step 4: If FRONT = SIZE-1 then  
    FRONT=0  
Else  
    FRONT=FRONT+1
```

```
#include <iostream>  
  
using namespace std;  
  
int cqueue[5];  
  
int front = -1, rear = -1, n=5;
```

```
void insertCQ(int val) {  
    if ((front == 0 && rear == n-1) || (front == rear+1)) {  
        cout<<"Queue Overflow \n";  
        return;  
    }  
    if (front == -1) {  
        front = 0;  
        rear = 0;  
    } else {  
        if (rear == n - 1)  
            rear = 0;  
        else  
            rear = rear + 1;  
    }  
    cqueue[rear] = val ;  
}  
  
void deleteCQ() {  
    if (front == -1) {  
        cout<<"Queue Underflow\n";  
        return ;  
    }  
    cout<<"Element deleted from queue is : "<<cqueue[front]<<endl;  
  
    if (front == rear) {  
        front = -1;  
        rear = -1;  
    } else {  
        front = front + 1;  
    }  
}
```



```
        if (front == n - 1)

            front = 0;

        else

            front = front + 1;

    }

}

void displayCQ() {

    int f = front, r = rear;

    if (front == -1) {

        cout<<"Queue is empty"<<endl;

        return;

    }

    cout<<"Queue elements are :\n";

    if (f <= r) {

        while (f <= r){

            cout<<cqueue[f]<<" ";

            f++;

        }

    } else {

        while (f <= n - 1) {

            cout<<cqueue[f]<<" ";

            f++;

        }

        f = 0;

        while (f <= r) {

            cout<<cqueue[f]<<" ";

            f++;

        }

    }

}
```

```
    }

    cout<<endl;
}

int main() {

    int ch, val;

    cout<<"1)Insert\n";
    cout<<"2)Delete\n";
    cout<<"3)Display\n";
    cout<<"4)Exit\n";

    do {

        cout<<"Enter choice : "<<endl;
        cin>>ch;
        switch(ch) {

            case 1:

                cout<<"Input for insertion: "<<endl;
                cin>>val;
                insertCQ(val);
                break;

            case 2:

                deleteCQ();
                break;

            case 3:

                displayCQ();
                break;
```

```
        case 4:
            cout<<"Exit\n";
            break;
            default: cout<<"Incorrect!\n";
        }
    } while(ch != 4);
    return 0;
}
```

18) Write down a routine to delete a particular node from doubly linked list.

**Algorithm to delete node from any position**

**% Input :** *head* {Pointer to the first node of the list}

*Last* {Pointer to the last node of the list}

*N* {Position to be deleted from list}

**Begin:**

*current*  $\leftarrow$  *head*;

**For** *i*  $\leftarrow$  1 to *N* **and** *current*  $\neq$  NULL **do**

*current*  $\leftarrow$  *current.next*;

**End for**

**If** (*N* == 1) **then**

deleteFromBeginning()

**End if**

**Else if** (*current* == *Last*) **then**

deleteFromEnd()

**End if**

```
Else if (current != NULL) then

    current.prev.next ← current.next

    If (current.next != NULL) then

        current.next.prev ← current.prev;

    End if

    unalloc (current)

    write ('Node deleted successfully from ', N, ' position')

End if

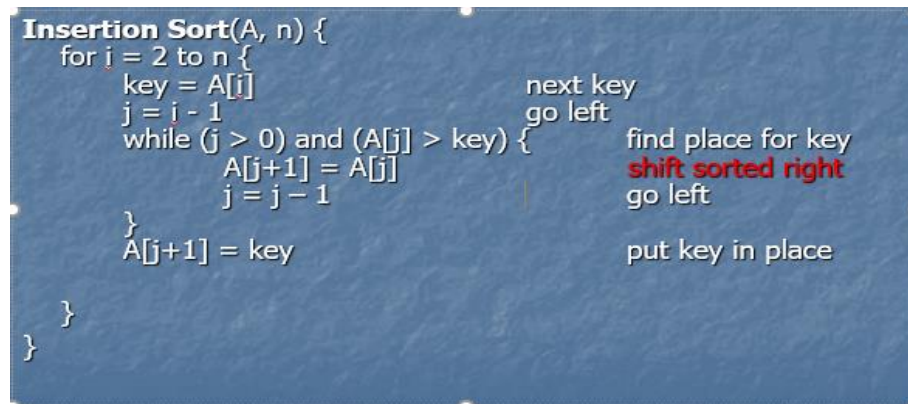
Else then

    write ('Invalid position')

End if

End
```

- 19) Write down a routine/algorithm to insert a node before a node from singly linked list.
- 20) Write down a routine/algorithm to perform all elements from circular queue till number  $n$  is found in circular queue.
- 21) (a) write the pseudocode of insertion sort.



```
Insertion Sort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}
```

Annotations:

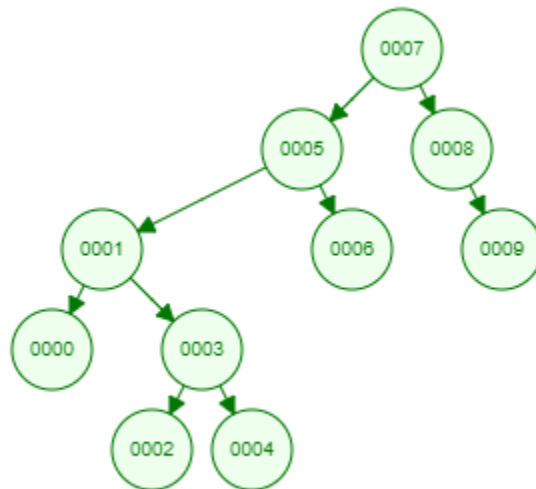
- next key
- go left
- find place for key
- shift sorted right
- go left
- put key in place

- (b) Write the pseudocode of selection sort.

```
Selection Sort(A)
1 n: = length[A]
```

```
2 for i:= 1 to n
3   // GOAL: place the correct number in A[i]
4   j:= FindIndexOfSmallest( A, i, n )
5   swap A[i] with A[j]
   // L.I. A[1..i] the i smallest numbers sorted
6 end-for
7 end-procedure
```

- 22) Suppose that we implement a hash table using a character array of size 13 (indexed 0 to 12) to store the keys/characters themselves. Show the elements of the map/array when the keys “SILVERMEDAL” are inserted in that order into an empty hash table with linear probing to resolve conflicts. (Note: S=19<sup>th</sup> letter of alphabet and I=9<sup>th</sup> and so on)
- 23) (a) suppose the numbers 7, 5, 1, 8, 3, 6, 0, 9, 4, 2 are inserted in that order into an initially empty binary search tree. The binary search tree uses the usual ordering on natural numbers. What is the in-order traversal sequence of the resultant tree?



In order: 0,1,2,3,4,5,6,7,8,9

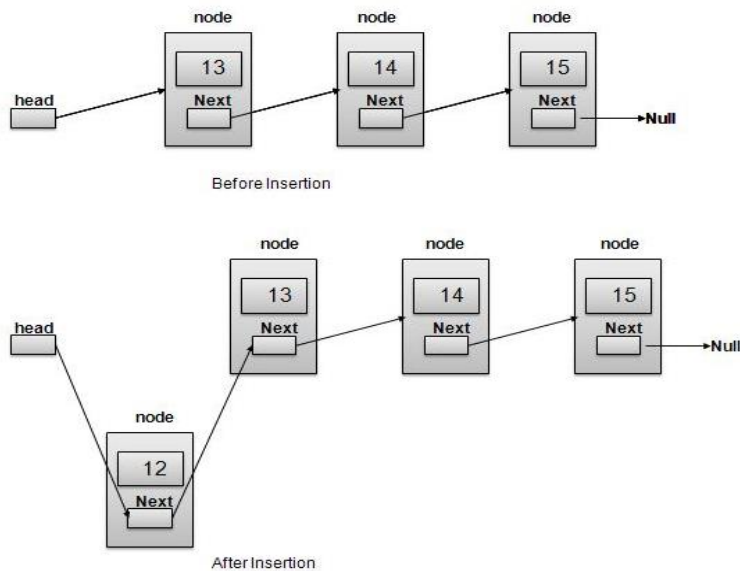
- (b) The in-order and pre-order traversal of a binary tree are d b e a f c g and a b d e c f, g, respectively. What is the post-order traversal of the binary tree?

24) Write down the algorithm/code for insert, delete and search elements(nodes) in singly linked list.

**Ans: Insertion Operation**

Insertion is a three-step process –

- Create a new Link with provided data.
- Point New Link to old First Link.
- Point First Link to this New Link.



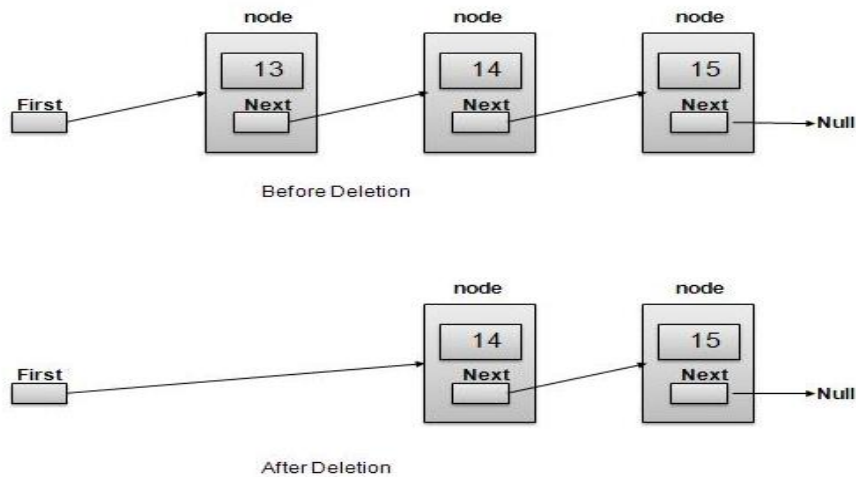
**Algorithm:**

```
//insert link at the first location
void insert First(int key, int data){
    struct node *link = (struct node*) malloc(sizeof(struct node));
    link->key = key;
    link->data = data;
    link->next = head;
    head = link;
}
```

**Deletion Operation:**

Deletion is a twostep process –

- Get the Link pointed by First Link as Temp Link.
- Point First Link to Temp Link's Next Link.



```
//delete first item  
  
struct node* deleteFirst(){  
    struct node *tempLink = head;  
  
    head = head->next;  
  
    return tempLink;  
}
```

#### Algorithm to Search Node in Linked List:

**Step 1:** FLAG = 0

SAVE=FIRST

**Step 2:** Repeat step 3 while SAVE ≠ NULL

**Step 3:** If SAVE->INFO = X then

FLAG = 1

SAVE=SAVE->LINK

Else

SAVE=SAVE->LINK

**Step 4:** If FLAG = 1 then

Write “Search is Successful”

Else

Write “Search is not successful”

**Step 5: Exit**

25) (a) write a recursive function that returns the sum of the square of the first n positive integers.

```
#include <iostream>
using namespace std;

int summation(int n)
{
    int sum = 0;
    for (int i = 1; i <= n; i++)
        sum += (i * i);

    return sum;
}

int main()
{
    int n = 2;
    cout << summation(n);
    return 0;
}
```

(b) write down the procedure of merge sort.

**How Merge Sort Works?**

To understand merge sort, we take an unsorted array as the following –



We know that merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. We see here that an array of 8 items is divided into two arrays of size 4.



This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves.





We further divide these arrays and we achieve atomic value which can no more be divided.



Now, we combine them in exactly the same manner as they were broken down. Please note the color codes given to these lists.

We first compare the element for each list and then combine them into another list in a sorted manner. We see that 14 and 33 are in sorted positions. We compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27. We change the order of 19 and 35 whereas 42 and 44 are placed sequentially.



In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.



After the final merging, the list should look like this –



Now we should learn some programming aspects of merge sorting.

### Algorithm

Merge sort keeps on dividing the list into equal halves until it can no more be divided. By definition, if it is only one element in the list, it is sorted. Then, merge sort combines the smaller sorted lists keeping the new list sorted too.

```
void merge(int Arr[], int start, int mid, int end) {  
  
    int temp[] = new int[end - start + 1];
```

```
int i = start, j = mid+1, k = 0

while(i <= mid && j <= end) {
    if(Arr[i] <= Arr[j]) {
        temp[k] = Arr[i];
        k += 1; i += 1;
    }
    else {
        temp[k] = Arr[j];
        k += 1; j += 1;
    }
}

while(i <= mid) {
    temp[k] = Arr[i];
    k += 1; i += 1;
}

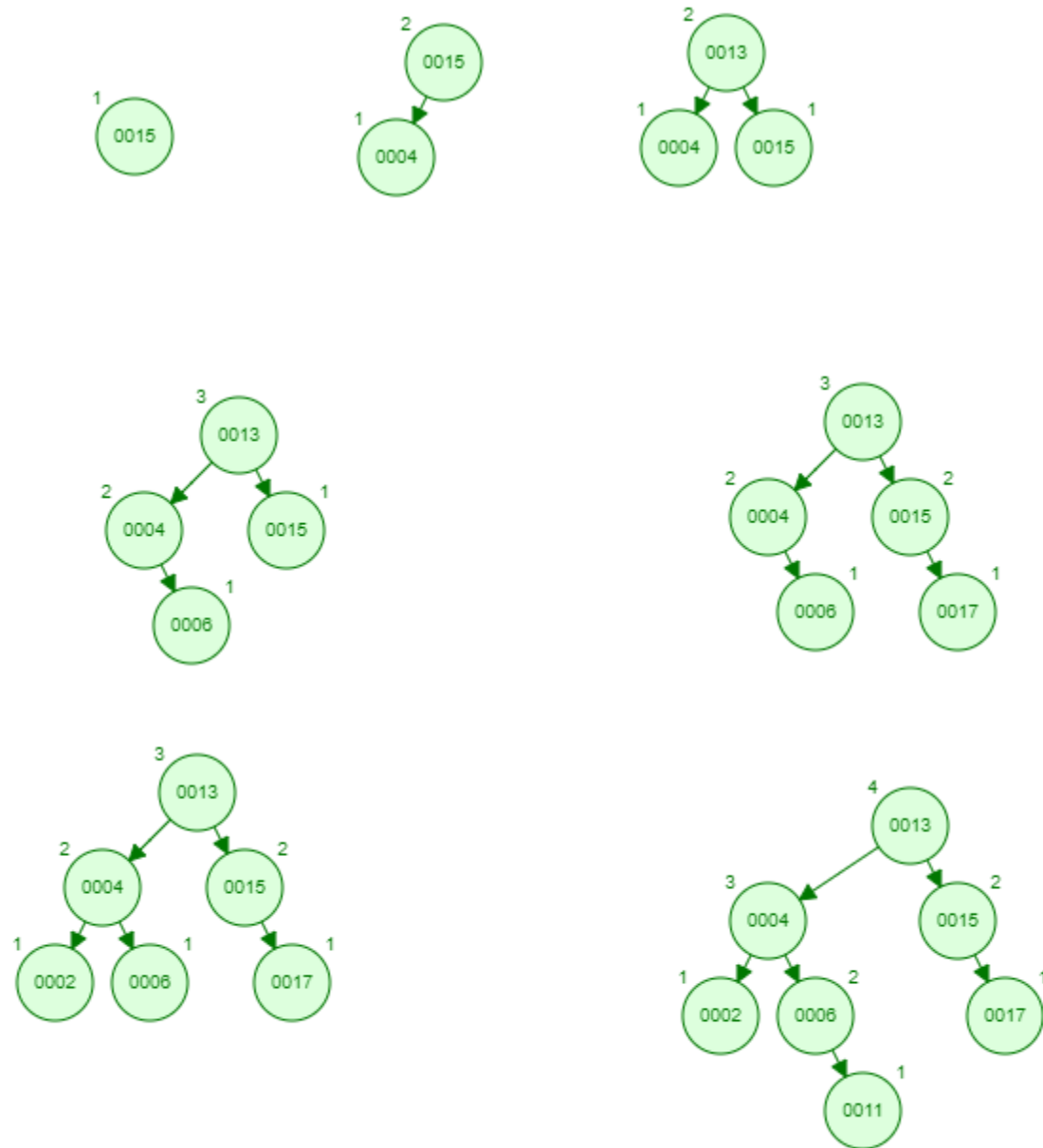
while(j <= end) {
    temp[k] = Arr[j];
    k += 1; j += 1;
}

for(i = start; i <= end; i += 1) {
    Arr[i] = temp[i - start]
}

}

void mergeSort(int Arr[], int start, int end) {
    if(start < end) {
        int mid = (start + end) / 2;
        mergeSort(Arr, start, mid);
        mergeSort(Arr, mid+1, end);
        merge(Arr, start, mid, end);
    }
}
```

26) (a) Draw AVL tree for the following digits 15, 4, 13, 6, 17, 2, 11 and also perform necessary rotations, while showing all intermediate trees being created in the process.



(b)What is meant by collision? What are the different ways to deal with collision while doing hashing?

When we put objects into a hash table, it is possible that different objects (by the *equals ()* method) might have the same hash code. This is called a **collision**. Here is the example of collision. Two different strings ""Aa" and "BB" have the same key:

$$\text{"Aa"} = \text{'A'} * 31 + \text{'a'} = 2112$$

$$\text{"BB"} = \text{'B'} * 31 + \text{'B'} = 2112$$

In the small number of cases, where multiple keys map to the same integer, then elements with different keys may be stored in the same "slot" of the hash table. It is clear that when the hash function is used to locate a potential match, it will be necessary to compare the key of that element with the search key. But there may be more than one element which should be stored in a single slot of the table. Various techniques are used to manage this problem:

### Handling the collisions:

1. chaining,
2. overflow areas,
3. re-hashing,
4. Linear Probe
5. quadratic probing,
6. random probing, ...

### Chaining

One simple scheme is to chain all collisions in lists attached to the appropriate slot. This allows an unlimited number of collisions to be handled and doesn't require *a priori* knowledge of how many elements are contained in the collection. The tradeoff is the same as with linked lists versus array implementations of collections: linked list overhead in space and, to a lesser extent, in [time](#).

$$H(x) = x \bmod n$$

### Linear Probe:

One of the simplest re-hashing functions is +1 (or -1), *ie* on a collision, look in the neighbouring slot in the table. It calculates the new address extremely quickly.

$$H(x) = (x+1) \bmod n$$

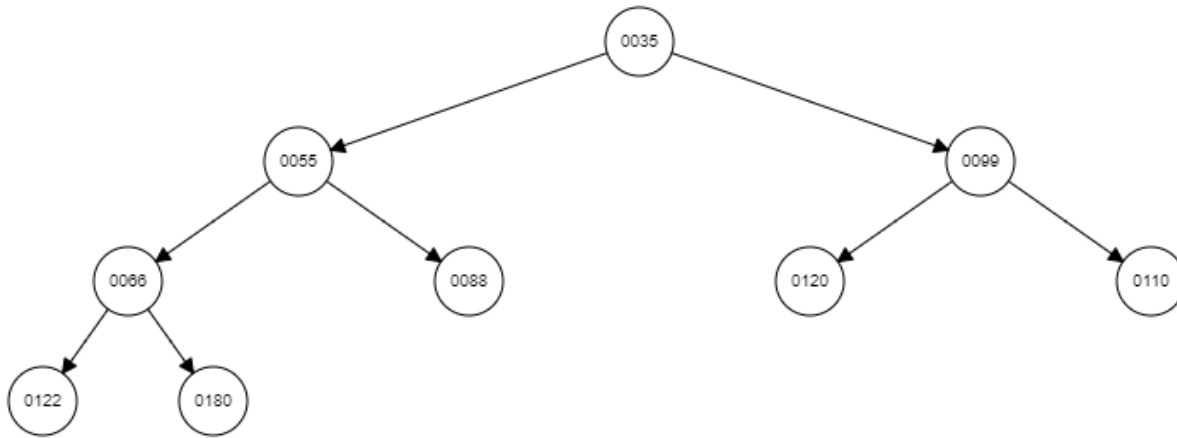
### Quadratic Probe:

Better behavior is usually obtained with quadratic probing, where the secondary hash function depends on the re-hash index:

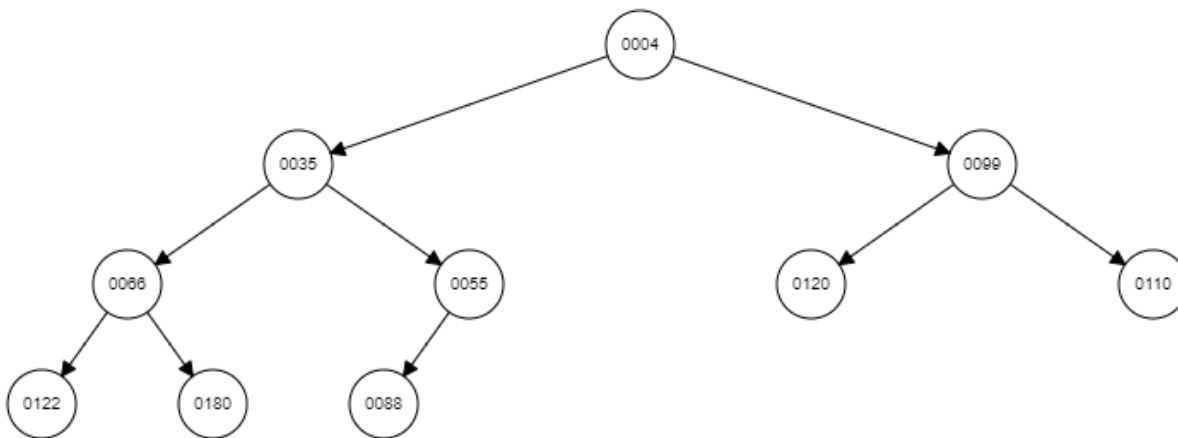
$$H(x) = (x+1^2) \bmod n$$

**27) (a) Following array is given which represent min heap. Insert 4 in the following array and convert it into a min heap again. Show process steps by drawing heap trees.**

35	55	99	66	88	120	110	122	180
----	----	----	----	----	-----	-----	-----	-----



after inserting 4



**(b)write down the algorithm for MIN- HEAPIFY?**

```
MIN-HEAPIFY(A, i)
    l = LEFT(i)
    r = RIGHT(i)
    if l ≤ A.heap-size and A[l] < A[i]
        smallest = l
    else
        smallest = i
    if r ≤ A.heap-size and A[r] < A[i]
        smallest = r
```

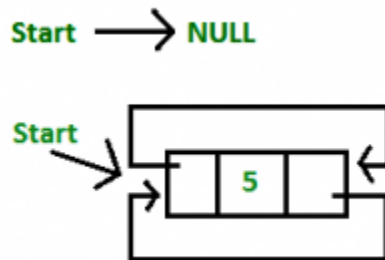
```
if smallest ≠ i
    exchange A[i] with A[smallest]
    MIN-HEAPIFY(A, smallest)
```

28) (a) Write down the code that inserts an element at the end of doubly circular linked list.

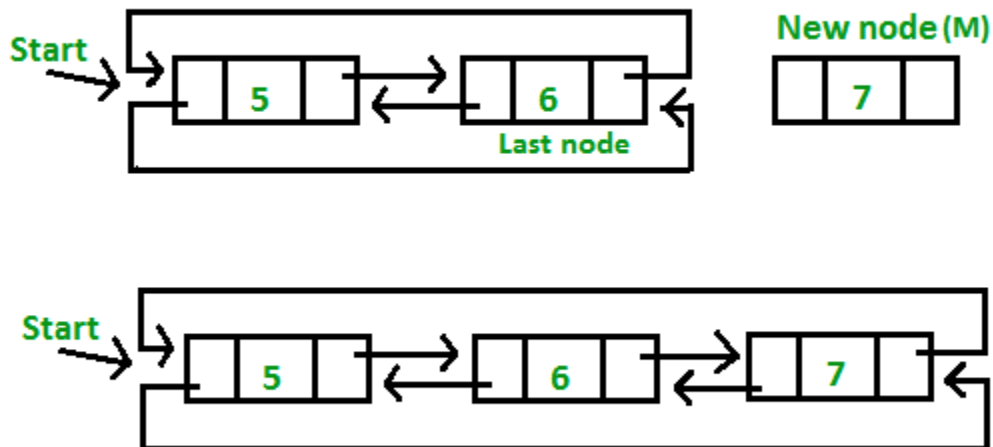
Ans:

**Insertion at the end of list or in an empty list**

- **Empty List (start = NULL):** A node (Say N) is inserted with data = 5, so previous pointer of N points to N and next pointer of N also points to N. But now start pointer points to the first node the list.



- **List initially contain some nodes, start points to first node of the List:** A node (Say M) is inserted with data = 7, so previous pointer of M points to last node, next pointer of M points to first node and last node's next pointer points to this M node and first node's previous pointer points to this M node.



```
void insertEnd(struct Node** start, int value)
{
    if (*start == NULL)
    {
        struct Node* new_node = new Node;
        new_node->data = value;
        new_node->next = new_node->prev = new_node;
        *start = new_node;
        return;
    }

    Node *last = (*start)->prev;

    struct Node* new_node = new Node;
    new_node->data = value;

    new_node->next = *start;

    (*start)->prev = new_node;

    new_node->prev = last;
    last->next = new_node;
}
```

**(b)Write down the code that deletes a node from singly linear link list having value '15'.**

**29) Write down the function for binary searching an element from an array recursively?**

**Ans:**

```
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l) {
        int mid = l + (r - l) / 2;
        if (arr[mid] == x)
            return mid;
        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);
        return binarySearch(arr, mid + 1, r, x);
    }

    return -1;
}

public static void main(String args[])
{
    BinarySearch ob = new BinarySearch();
    int arr[] = { 2, 3, 4, 10, 40 };
    int n = arr.length;
    int x = 10;
    int result = ob.binarySearch(arr, 0, n - 1, x);
    if (result == -1)
        System.out.println("Element not present");
    else

```

```

        System.out.println("Element found at index " + result);
    }
}

```

30) Write down a function to count the total number of elements in a singly linked list.

```

int count(node *temp)
{
    if(temp == NULL)
        return(0);
    return(1 + count(temp->next));
}

```

31) (a) convert the following infix expression to postfix expression using stack.

$A + (((B - C) * (D - E) + F) / G) \$ (H - J)$

**ABC-DE-\*F+G/\$HJ-+**

Sr. no.	Expression	Stack	Postfix
0	A	(	A
1	+	(+	A
2	(	(+(	A
3	(	(+((	A
4	(	(+(((	A
5	B	(+(((	AB
6	-	(+((( -	AB
7	C	(+((( -	ABC
8	)	(+((	ABC-
9	*	(+((*	ABC-
10	(	(+((*(	ABC-
11	D	(+((*(	ABC-D
12	-	(+((*(-	ABC-D
13	E	(+((*(-	ABC-DE
14	)	(+((*	ABC-DE-
15	+	(+((+	ABC-DE-*
16	F	(+((+	ABC-DE-*F
17	)	(+(	ABC-DE-*F+
18	/	(+(/	ABC-DE-*F+



Sr. no.	Expression	Stack	Postfix
19	G	(+(/	ABC-DE-*F+G
20	)	(+	ABC-DE-*F+G/
21	\$	(+	ABC-DE-*F+G/\$
22	(	(+(	ABC-DE-*F+G/\$
23	H	(+(	ABC-DE-*F+G/\$H
24	-	(+(-	ABC-DE-*F+G/\$H
25	J	(+(-	ABC-DE-*F+G/\$HJ
26	)	(+	ABC-DE-*F+G/\$HJ-
27	)		ABC-DE-*F+G/\$HJ-+

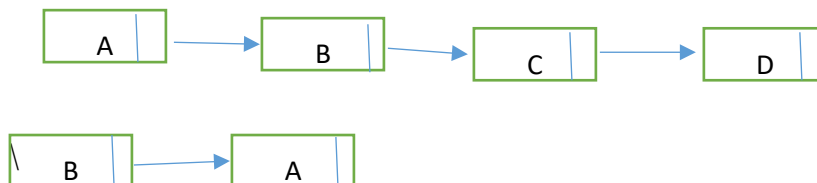
(b)write down the recursive function to all elements of the array.

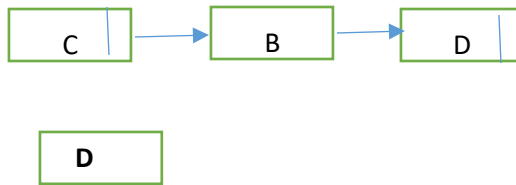
```
#include <stdio.h>

int findSum(int A[], int N)
{
    if (N <= 0)
        return 0;
    return (findSum(A, N - 1) + A[N - 1]);
}

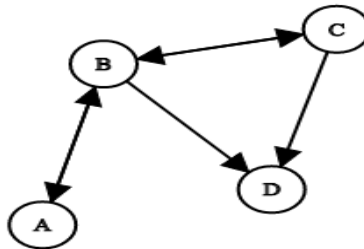
int main()
{
    int A[] = { 1, 2, 3, 4, 5 };
    int N = sizeof(A) / sizeof(A[0]);
    printf("%dn", findSum(A, N));
    return 0;
}
```

32) Draw a picture of the directed graph that has following adjacency list representation.





Ans:



### 33) Give an interface and implementation of queue.

**Ans:** Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

A real-world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first. More real-world examples can be seen as queues at the ticket windows and bus-stops

### Queue Representation

As we now understand that in queue, we access both ends for different reasons. The following diagram given below tries to explain queue representation as data structure —



As in stacks, a queue can also be implemented using Arrays, Linked-lists, Pointers and Structures. For the sake of simplicity, we shall implement queues using one-dimensional array.

### Basic Operations

Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with queues –

- **enqueue ()** – add (store) an item to the queue.
- **dequeue ()** – remove (access) an item from the queue.

Few more functions are required to make the above-mentioned queue operation efficient. These are –

- **peek ()** – Gets the element at the front of the queue without removing it.
- **isfull()** – Checks if the queue is full.
- **isempty()** – Checks if the queue is empty.

In queue, we always dequeue (or access) data, pointed by **front** pointer and while enqueueing (or storing) data in the queue we take help of **rear** pointer.

Let's first learn about supportive functions of a queue –

### **peek ()**

This function helps to see the data at the **front** of the queue. The algorithm of peek () function is as follows –

### **Algorithm**

```
begin procedure peek
    return queue[front]
end procedure
```

Implementation of peek () function in C programming language –

### **Example**

```
int peek() {
    return queue[front];
}
```

### **isfull()**

As we are using single dimension array to implement queue, we just check for the rear pointer to reach at MAXSIZE to determine that the queue is full. In case we maintain the queue in a circular linked-list, the algorithm will differ. Algorithm of isfull() function –

### Algorithm

```
begin procedure isfull

    if rear equals to MAXSIZE
        return true
    else
        return false
    endif

end procedure
```

Implementation of isfull() function in C programming language –

### Example

```
bool isfull() {
    if(rear == MAXSIZE - 1)
        return true;
    else
        return false;
}
```

### isempty()

Algorithm of isempty() function –

### Algorithm

```
begin procedure isempty

    if front is less than MIN OR front is greater than rear
        return true
    else
```

```
        return false
    endif

end procedure
```

If the value of **front** is less than MIN or 0, it tells that the queue is not yet initialized, hence empty.

Here's the C programming code –

### Example

```
bool isempty() {
    if(front < 0 || front > rear)
        return true;
    else
        return false;
}
```

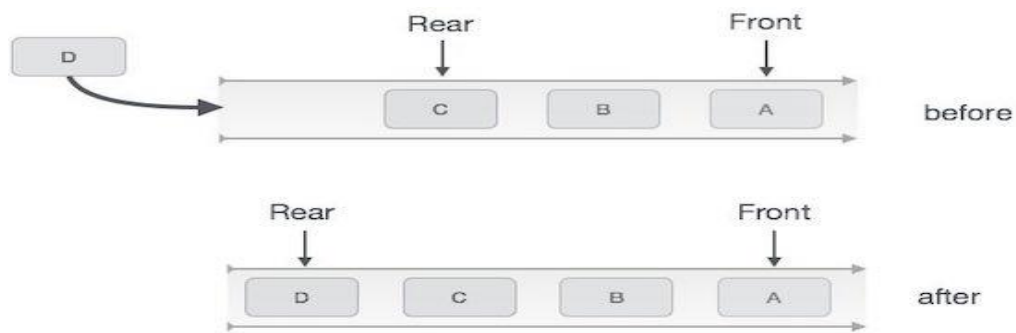
### Enqueue Operation

Queues maintain two data pointers, **front** and **rear**. Therefore, its operations are comparatively difficult to implement than that of stacks.

The following steps should be taken to enqueue (insert) data into a queue –

- **Step 1** – Check if the queue is full.
- **Step 2** – If the queue is full, produce overflow error and exit.
- **Step 3** – If the queue is not full, increment **rear** pointer to point the next empty space.
- **Step 4** – Add data element to the queue location, where the rear is pointing.

- **Step 5** – return success.



### Queue Enqueue

Sometimes, we also check to see if a queue is initialized or not, to handle any unforeseen situations.

### Algorithm for enqueue operation

```
procedure enqueue(data)

    if queue is full
        return overflow
    endif

    rear ← rear + 1
    queue[rear] ← data
    return true

end procedure
```

Implementation of enqueue () in C programming language –

### Example

```
int enqueue(int data)

    if(isfull())

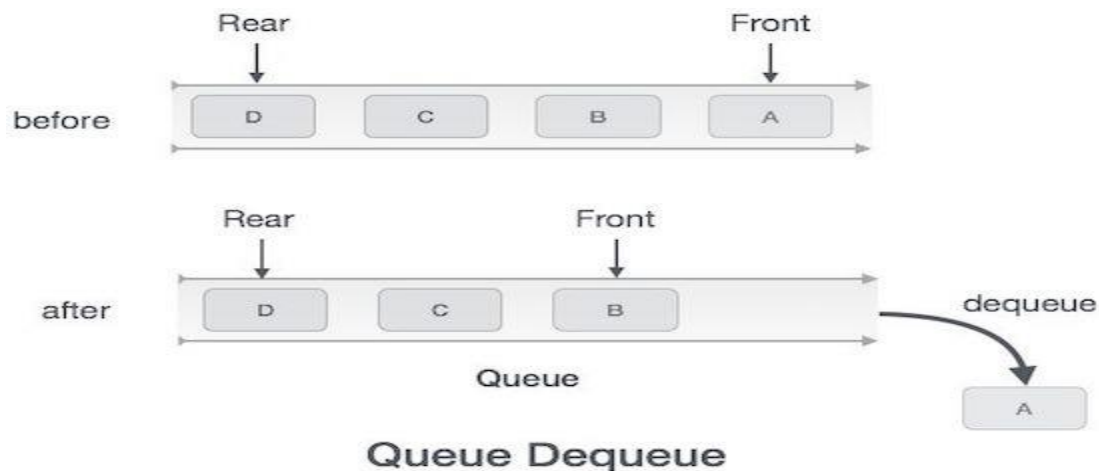
        return 0;
```

```
    rear = rear + 1;  
    queue[rear] = data;  
  
    return 1;  
end procedure
```

### Dequeue Operation

Accessing data from the queue is a process of two tasks – access the data where **front** is pointing and remove the data after access. The following steps are taken to perform **dequeue** operation –

- **Step 1** – Check if the queue is empty.
- **Step 2** – If the queue is empty, produce underflow error and exit.
- **Step 3** – If the queue is not empty, access the data where **front** is pointing.
- **Step 4** – Increment **front** pointer to point to the next available data element.
- **Step 5** – Return success.



### Algorithm for dequeue operation

```
procedure dequeue  
  
    if queue is empty  
        return underflow
```

```
end if

data = queue[front]

front ← front + 1

return true

end procedure
```

Implementation of dequeue() in C programming language –

### Example

```
int dequeue() {
    if(isempty())
        return 0;

    int data = queue[front];
    front = front + 1;

    return data;
}
```

**34) Write an algorithm (program) of insertion sort and perform its analysis.**

```
for i = 1 to length(A)
    x = A[i]
    j = i - 1
    while j >= 0 and A[j] > x
        A[j+1] = A[j]
        j = j - 1
    end while
    A[j+1] = x
```



```
end for
```

**Analysis:**

insertion sort runs in  $O(n)$  time in its best case and runs in  $O(n^2)$  in its worst and average cases.

**35) Describe recursive routine using an appropriate example.**

**36) Suppose a linked list is empty. Write a function to add a node at start of linked list?**

**37) Write a function to delete a node from circular queue?**

**38) Write a recursive function to add first 10 integers from 1 to 10.**

**39) Suppose there are 8 keys with numbers 25,351,37,30,79,5,23. There are 10 has addresses available. Try to accommodate them in available slots using linear probing technique. The hashing function used is  $h(k) = \text{key} \% 9$**

**40) Write methods for the following.**

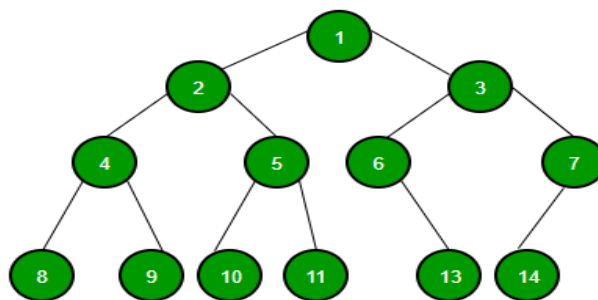
a) To delete an item from sorted array.

b) To count and return all nodes having odd values (A node has only integers as its informational part).

**41) Define a binary tree, full binary tree, complete binary tree. Also tell maximum number of nodes on a given level of a binary tree? 37)**

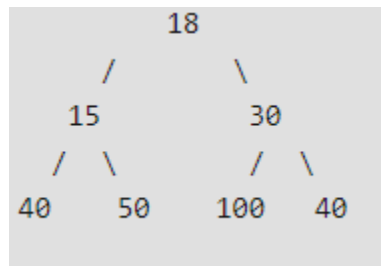
**Ans: Binary Tree:**

A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.



**Full Binary Tree:**

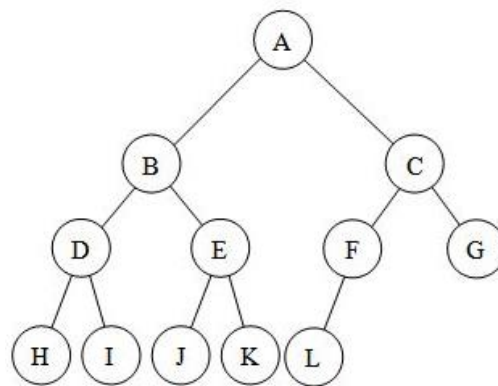
A Binary Tree is full if every node has 0 or 2 children. Following are examples of a full binary tree. We can also say a full binary tree is a binary tree in which all nodes except leaves have two children.



### Complete Binary Tree:

A Binary Tree is complete Binary Tree if all levels are completely filled except possibly the last level and the last level has all keys as left as possible.

Following are examples of Complete Binary Trees



The maximum number of nodes in a binary tree of height  $h = 2^{h+1} - 1$

For any query contact at:

[asim.rana63@gmail.com](mailto:asim.rana63@gmail.com)

.....Good Luck.....