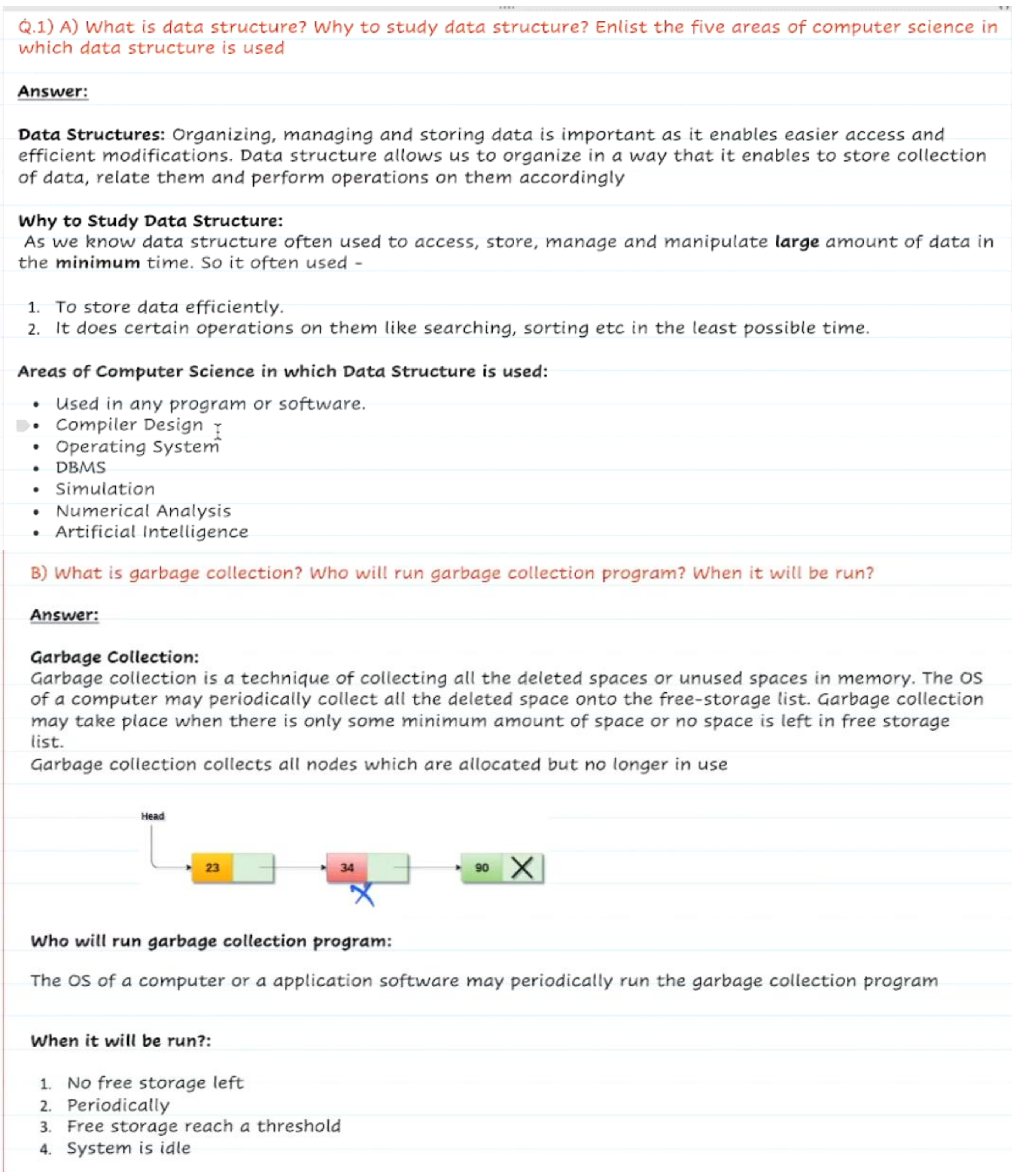
# DS WINTER 2018

08 March 2023 04:17 AM

Sorted & Merged By @official_stranger



Q.1) A) What is data structure? Why to study data structure? Enlist the five areas of computer science in which data structure is used

**Answer:**

**Data Structures:** Organizing, managing and storing data is important as it enables easier access and efficient modifications. Data structure allows us to organize in a way that it enables to store collection of data, relate them and perform operations on them accordingly

**Why to Study Data Structure:**
As we know data structure often used to access, store, manage and manipulate **large** amount of data in the **minimum** time. So it often used -

1. To store data efficiently.
2. It does certain operations on them like searching, sorting etc in the least possible time.

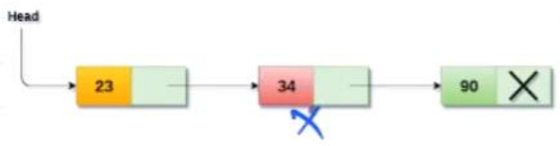
**Areas of Computer Science in which Data Structure is used:**

- Used in any program or software.
- Compiler Design
- Operating System
- DBMS
- Simulation
- Numerical Analysis
- Artificial Intelligence

B) What is garbage collection? Who will run garbage collection program? When it will be run?

**Answer:**

**Garbage Collection:**
Garbage collection is a technique of collecting all the deleted spaces or unused spaces in memory. The OS of a computer may periodically collect all the deleted space onto the free-storage list. Garbage collection may take place when there is only some minimum amount of space or no space is left in free storage list.
Garbage collection collects all nodes which are allocated but no longer in use

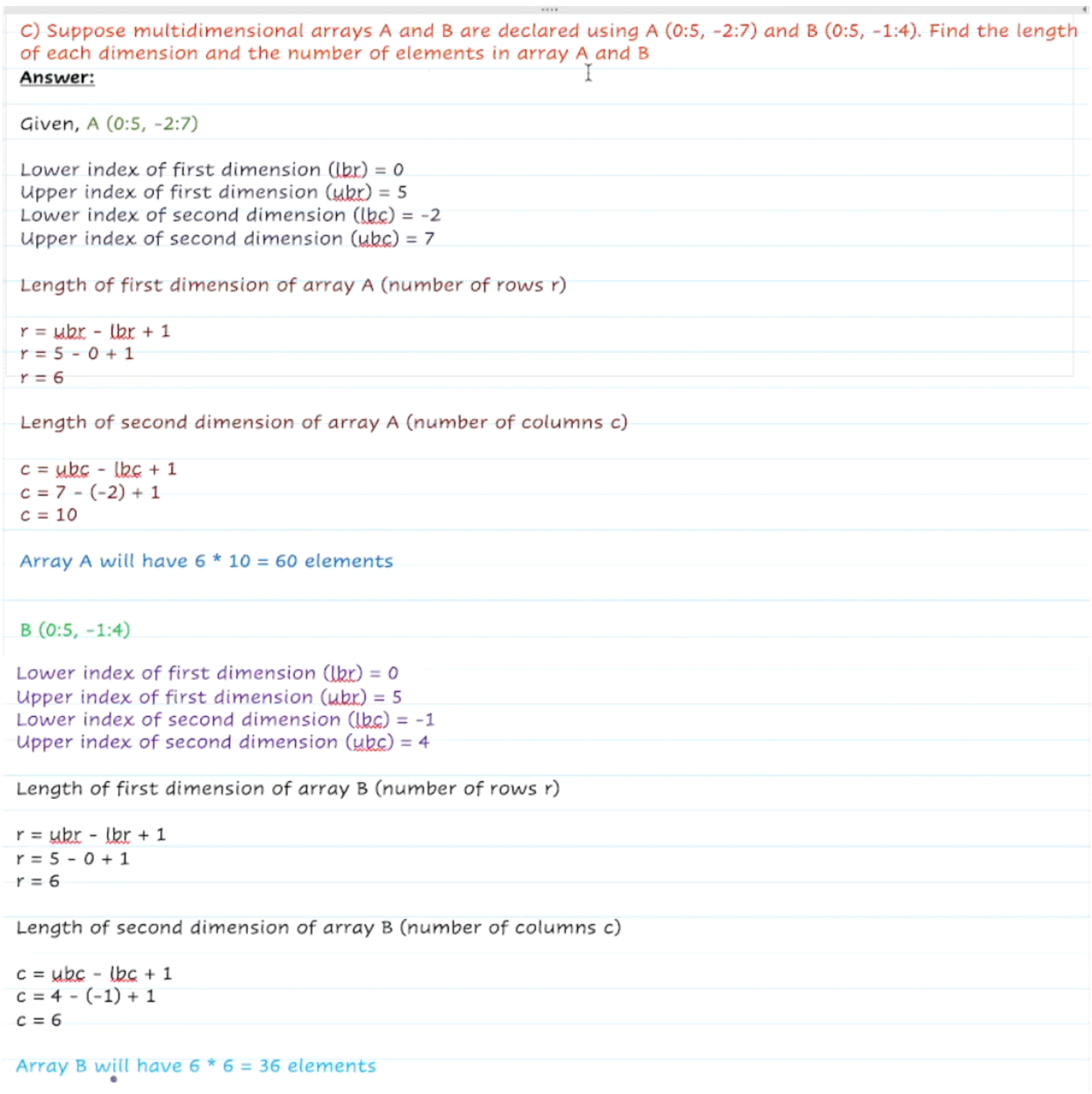


**Who will run garbage collection program:**

The OS of a computer or a application software may periodically run the garbage collection program

**When it will be run?:**

1. No free storage left
2. Periodically
3. Free storage reach a threshold
4. System is idle



C) Suppose multidimensional arrays A and B are declared using A (0:5, -2:7) and B (0:5, -1:4). Find the length of each dimension and the number of elements in array A and B

**Answer:**

Given, A (0:5, -2:7)

Lower index of first dimension (lbr) = 0
Upper index of first dimension (ubr) = 5
Lower index of second dimension (lbc) = -2
Upper index of second dimension (ubc) = 7

Length of first dimension of array A (number of rows r)

r = ubr - lbr + 1
r = 5 - 0 + 1
r = 6

Length of second dimension of array A (number of columns c)

c = ubc - lbc + 1
c = 7 - (-2) + 1
c = 10

Array A will have 6 * 10 = 60 elements

B (0:5, -1:4)

Lower index of first dimension (lbr) = 0
Upper index of first dimension (ubr) = 5
Lower index of second dimension (lbc) = -1
Upper index of second dimension (ubc) = 4

Length of first dimension of array B (number of rows r)

r = ubr - lbr + 1
r = 5 - 0 + 1
r = 6

Length of second dimension of array B (number of columns c)

c = ubc - lbc + 1
c = 4 - (-1) + 1
c = 6

Array B will have 6 * 6 = 36 elements

D) What is primitive data structure? Enlist the differences between primitive and non-primitive data structures

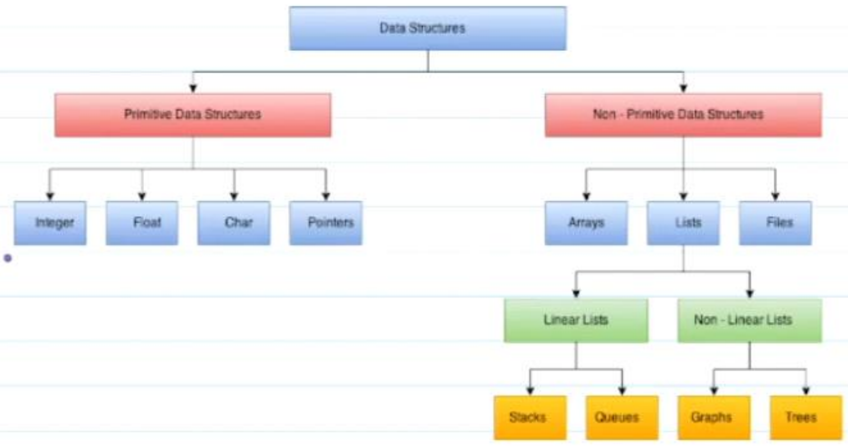
**Answer:**

Data structures are classified into two types:

Primitive data structure

Non - primitive data structure

**Primitive data structure:** primitive data structure is the fundamental data structure, which stores the values of only one data type for example: A variable with the data type integer can allow storing the values of integer type only, a variable with the float data type can allow storing the values of float data type. Basically, the primitive data structure consists of fundamental data types like float, character, integer, etc. The classification of data structure is as follows:



**Difference between primitive and non-primitive data structures:**

| Primitive data structure | Non - primitive data structure |
|---|---|
| Primitive data structure is a kind of data structure that stores the data of one type | Non-primitive data structure is a type of data structure that can store the data of more than one type. |
| Examples of primitive data structure are integer, character, float. | Examples of non-primitive data structure are Array, Linked list, stack. |
| Primitive data structure will contain some value, i.e., it cannot be NULL. | Non-primitive data structure can consist of a NULL value. |
| The size depends on the type of the data structure. | In case of non-primitive data structure, size is not fixed |
| It starts with a lowercase character. | It starts with an uppercase character. |
| Primitive data structure can be used to call the methods. | Non-primitive data structure cannot be used to call the methods. |

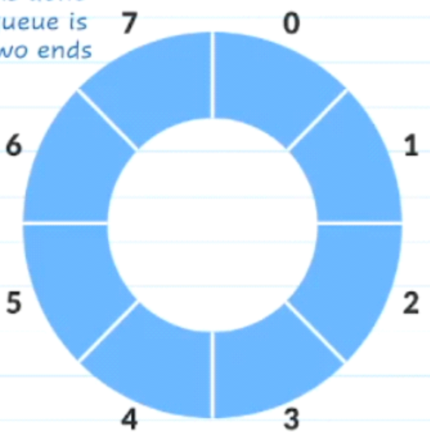
Q.2) A) What is circular queue? Let the following circular queue can accommodate maximum six elements with the following data, front = 2, rear = 4 and initial queue content is queue = ----, L, M, N, ----, ---
Show the queue content with front and rear value after the following operations.
i) Insert A ii) Delete iii) Insert B iv) Delete

**Answer:**

**Circular Queue:**

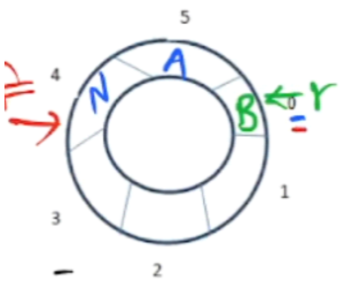
A circular queue is one in which the insertion of new element is done at the very first location of the queue. If the last location of queue is full it can be viewed as a mesh or loop of wire, in which the two ends of wire connected together.

Fig. shows a empty circular queue Q[8]



Circular queue overcomes the problem of unutilized space in linear queues implemented as arrays.

It has also a front and rear to keep the track of the elements to be deleted and inserted



Insert A

Rear = 5, front = 2

Delete

Rear = 5, front = 3

Insert B

Rear = 0, front = 3

Delete

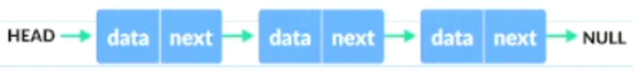
Rear = 0, front = 4

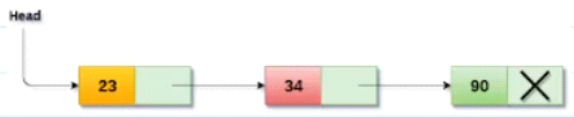
20:17

Q.2) B) What is singly linked list? Write algorithm to find the number of times a given ITEM occurs in the singly linked list.

**Answer:**

Singly Linked List:
Singly linked list can be defined as the collection of ordered set of elements.
The number of elements may vary according to the need.
A node in the singly linked list consist of two parts: data part and link part.
Data part of the node stores actual information that is to be represented by the node while,
The link part of the node stores the address of its immediate successor.



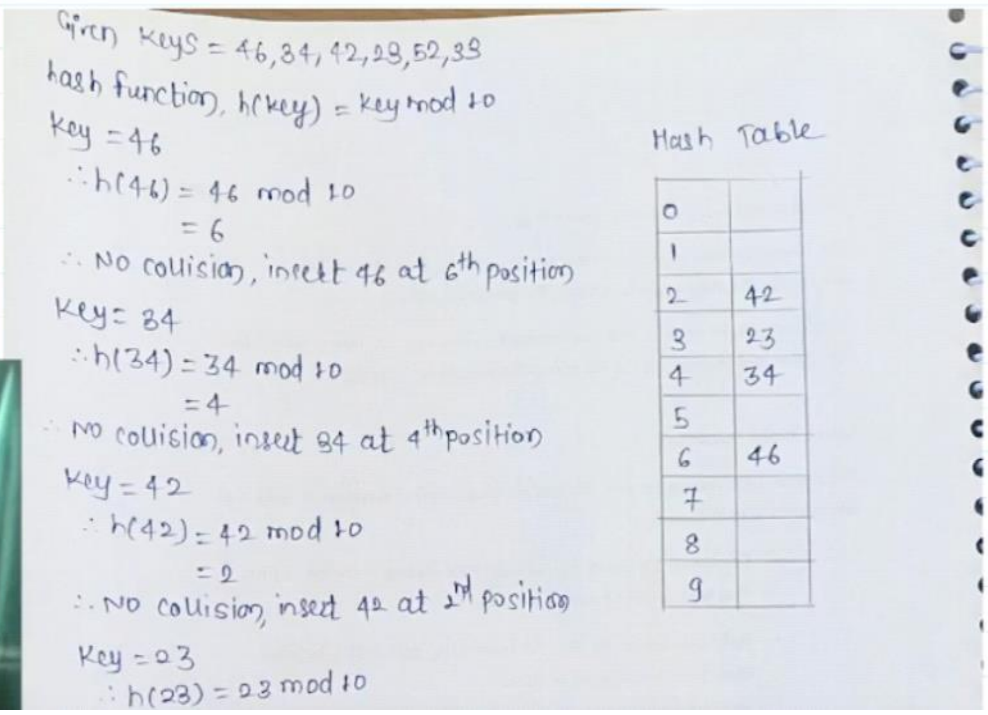


Algorithm:

```
1. Initialize count = 0
2. While L contains node do
       if L --> info == element then
           count = count + 1
       endif
           L = L --> next
   end while
3. Stop
```

Q. 2) C) Let the keys: 46, 34, 42, 23, 52, 33 are inserted into an empty hash table using function h (key) = key mod 10. Give hash table content after every insertion, if open addressing with linear probing is used to deal with collision.

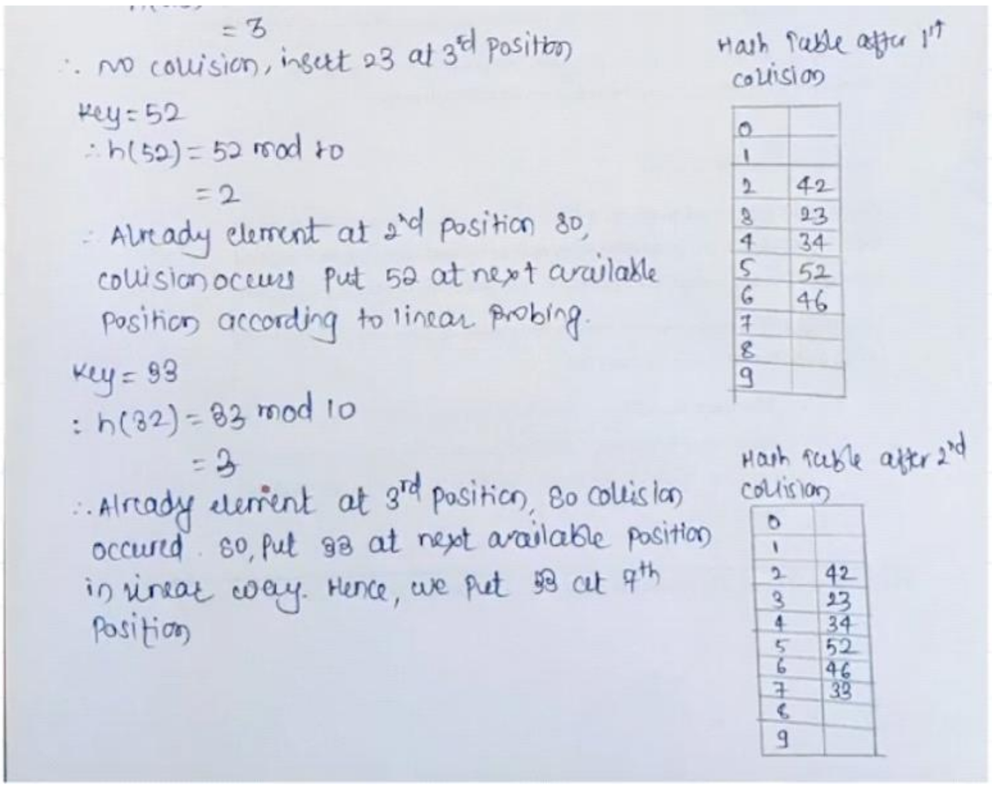
**Answer:**



Given Keys = 46,34,42,23,52,33
hash function, h(key) = key mod 10
Key = 46
∴ h(46) = 46 mod 10
= 6
∴ No collision, insert 46 at 6th position
Key = 34
∴ h(34) = 34 mod 10
= 4
∴ No collision, insert 34 at 4th position
Key = 42
∴ h(42) = 42 mod 10
= 2
∴ No collision, insert 42 at 2nd position
Key = 23
∴ h(23) = 23 mod 10

Hash Table

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 42 |
| 3 | 23 |
| 4 | 34 |
| 5 | |
| 6 | 46 |
| 7 | |
| 8 | |
| 9 | |



= 3

∴ no collision, insert 23 at 3rd position

Key = 52

∴ h(52) = 52 mod 10

= 2

∴ Already element at 2nd position so, collision occurs put 52 at next available position according to linear probing.

Key = 33

∴ h(33) = 33 mod 10

= 3

∴ Already element at 3rd position, so collision occured. so, put 33 at next available position in linear way. Hence, we put 33 at 7th position

Hash Table after 1st collision

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 42 |
| 3 | 23 |
| 4 | 34 |
| 5 | 52 |
| 6 | 46 |
| 7 | |
| 8 | |
| 9 | |

Hash Table after 2nd collision

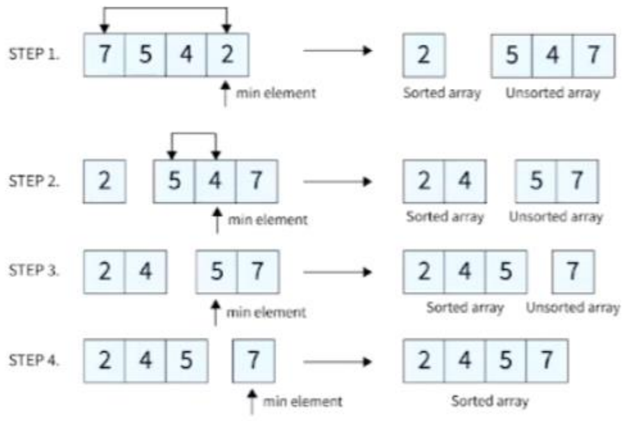
| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 42 |
| 3 | 23 |
| 4 | 34 |
| 5 | 52 |
| 6 | 46 |
| 7 | 33 |
| 8 | |
| 9 | |

Q.3) A) What is selection sort? Sort the numbers in ascending order and also show the worst case time complexity of selection sort is O ($n^2$)

**Answer:**

Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list. The selection sort performs in iterations. In every iteration of the selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the beginning of unsorted subarray. After every iteration sorted subarray size increase by one and unsorted subarray size decrease by one.

**Example:** Consider a array of 4 elements: 7, 5, 4, 2 which are not in ascending order. We have to convert it into an ascending order using selection sort.



**Algorithm:**
The selection sort algorithm is as follows:
Step 1: Set Min to location 0 in Step 1.
Step 2: Look for the smallest element on the list.
Step 3: Replace the value at location Min with a different value.
Step 4: Increase Min to point to the next element
Step 5: Continue until the list is sorted.

**Complexity:**

| Iterations | Number of comparisons |
|---|---|
| 1 | (n-1) |
| 2 | (n-2) |
| 3 | (n-3) |
| ... | ... |
| n | 1 |

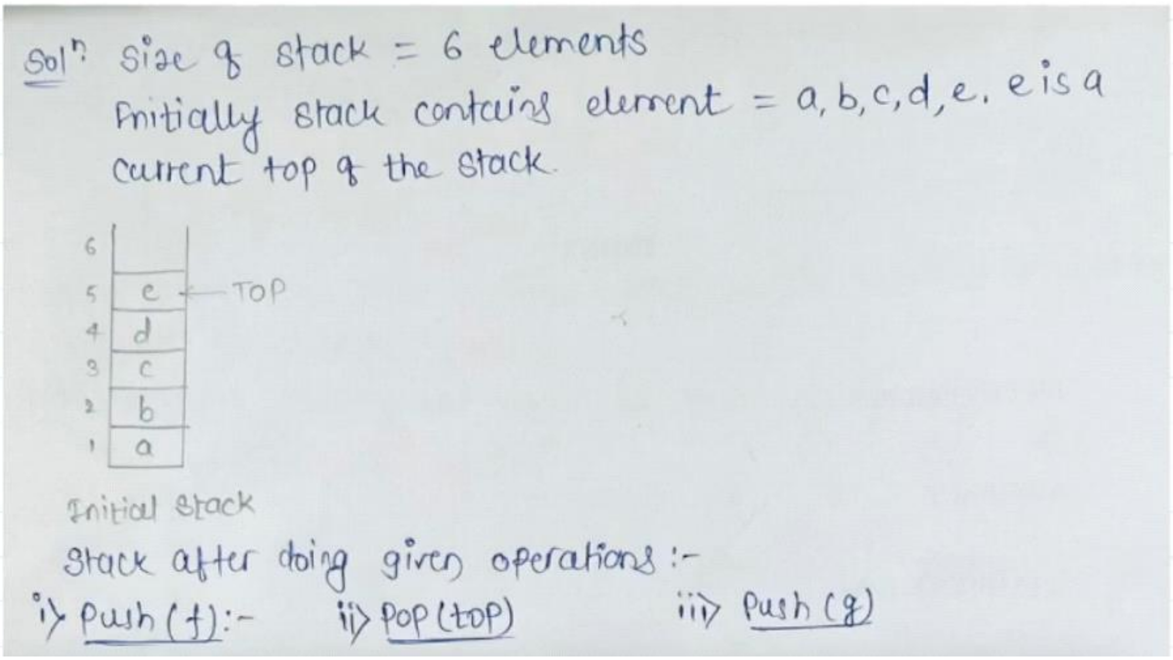
Total **number of comparisons** = (n - 1) + (n - 2) + (n - 3) + ..... + 1 = n(n - 1) / 2 nearly equals to **n²**. So we can say it requires complexity = O(**n²**). Also, we can analyze the complexity by simply observing the number of loops. There are 2 loops so the complexity is n*n = n².

**Time Complexities:**
**Worst Case Complexity:** O($n^2$)
If we want to sort in ascending order and the array is in descending order then, the worst case occurs.
**Best Case Complexity:** O($n^2$)
It occurs when the array is already sorted
**Average Case Complexity:** O($n^2$)
It occurs when the elements of the array are in jumbled order (neither ascending nor descending).

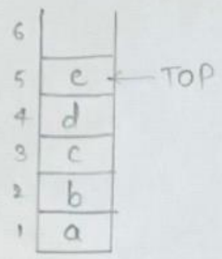
Q. 3) B) Consider the stack of size 6 memory cells. Suppose initially stack contains a, b, c, d, e (Top of stack). Then the following operations are executed in order. Show the stack top and any other situation raised while doing each of the operation.

i) Push(f) ii) Pop(top) iii) Push(g) iv) Push(h) v) Pop(top) vi) Push(I

**Answer:**



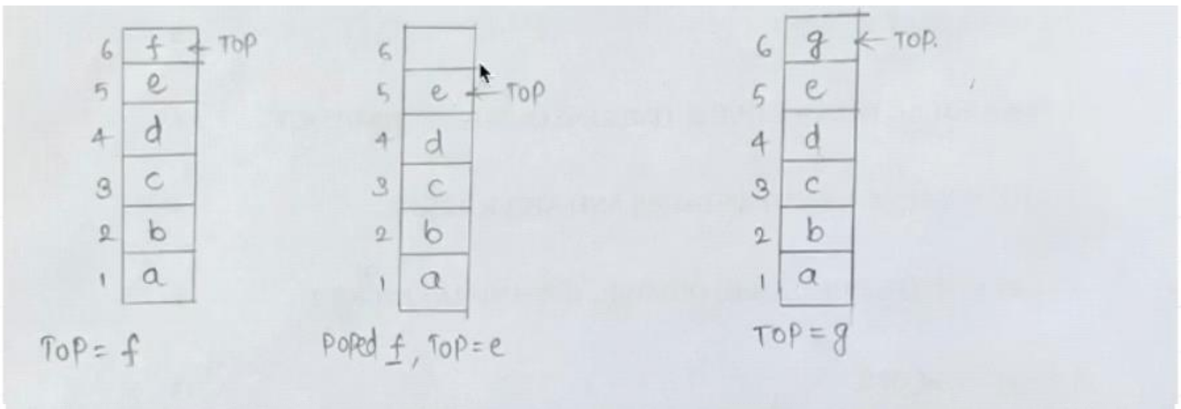
Sol$^n$: size of stack = 6 elements
Initially stack contains element = a, b, c, d, e. e is a current top of the stack.



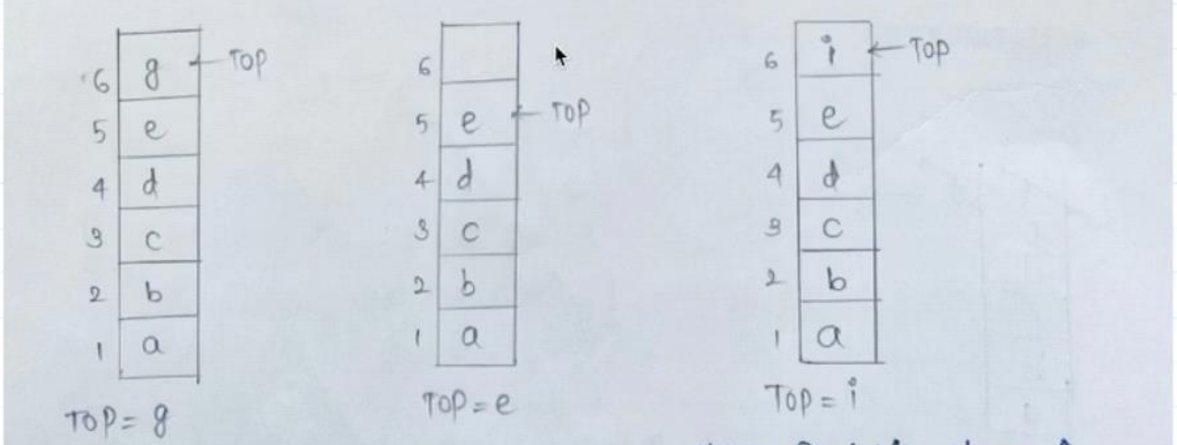
Initial stack

Stack after doing given operations :-

i> Push(f) :-   ii> Pop(top)   iii> Push(g)



iv) Push(h):- v) POP(top):- vi) Push(i):-



cannot push 'h', stack prints 'overflow'

After Push(i), stack is fulled completely & TOP element is 'i'

C) Explain how to implement two stacks in one array A[1 . . N] in such a way that neither stack overflow unless the total number of elements in both the stacks together is N. Note that, Push() and Pop() operations should be run in O (1) time.

**Answer:**

**Method 1 (Divide the space in two halves)**

A simple way to implement two stacks is to divide the array in two halves and assign the half half space to two stacks, i.e., use arr[0] to arr[n/2] for stack1, and arr[(n/2) + 1] to arr[n-1] for stack2 where arr[] is the array to be used to implement two stacks and size of array be n.

The problem with this method is inefficient use of array space. A stack push operation may result in stack overflow even if there is space available in arr[]. For example, say the array size is 6 and we push 3 elements to stack1 and do not push anything to second stack2. When we push 4th element to stack1, there will be overflow even if we have space for 3 more elements in array.

**Method 2 (A space efficient implementation)**

This method efficiently utilizes the available space. It doesn't cause an overflow if there is space available in arr[]. The idea is to start two stacks from two extreme corners of arr[]. stack1 starts from the leftmost element, the first element in stack1 is pushed at index 0. The stack2 starts from the rightmost corner, the first element in stack2 is pushed at index (n-1). Both stacks grow (or shrink) in opposite direction. To check for overflow, all we need to check is for space between top elements of both stacks.

Q.4) A) What are the different types of the linked list? Give advantages and disadvantages each of the linked list over another.

**Answer:**
A linked list is a sequential data structure. A Linked List is a sequence of links which contains items. Each link contains a connection to another link.

**Types of Linked List:**
Following are the various types of linked list.

1. **Singly Linked List** – It is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type.

2. **Doubly Linked List** – A doubly linked list or a two-way linked list is a more complex type of linked list that contains a pointer to the next as well as the previous node in sequence.

3. **Circular Linked List** – A circular linked list is one in which the last node contains the pointer to the first node of the list. A circular linked list has no beginning and no end.

**Advantages of Singly Linked list:**

1. It requires only one list pointer variable, i.e., the head pointer pointing to the first node.
2. It utilizes less memory space
3. It can be implemented on the stack.

**Disadvantages of Singly Linked list:**

1. It is less efficient as compared to a doubly-linked list
2. The singly linked list can be traversed only in the forward direction.
3. In a singly linked list, the time complexity for inserting and deleting an element from the list is **O(n)**.

**Advantages of doubly linked list:**

1. The doubly linked list can be accessed in both directions.
2. It is more efficient.
3. In a doubly-linked list, the time complexity for inserting and deleting an element is **O(1)**.
4. It can be implemented on stack, heap and binary tree.

**Disadvantages of doubly linked list:**

1. It utilizes more memory space.
2. It requires two list pointer variables, **head** and **last**.

**Advantages of Circular linked list:**

1. List can be traversed from both directions i.e. from head to tail or from tail to head.
2. Easy for data manipulation.
3. Jumping from head to tail or vice versa takes O(1) time.
4. Efficient insertion and deletion.
5. Ability to handle circular buffers.

**Disadvantages of Circular linked list:**

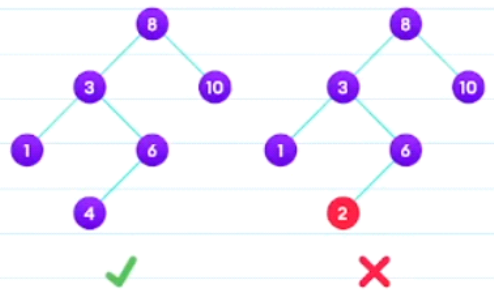
1. Requires additional memory
2. More complex than singly linked list
3. If not used properly, then the problem of infinite loop can occur
4. Have a slight overhead of maintaining both the next and previous pointers at each node

B) Assume, the following letters are inserted into an empty binary search tree in given order. J, B, D, F, N, K, O. Construct binary search tree and also give height of the tree.

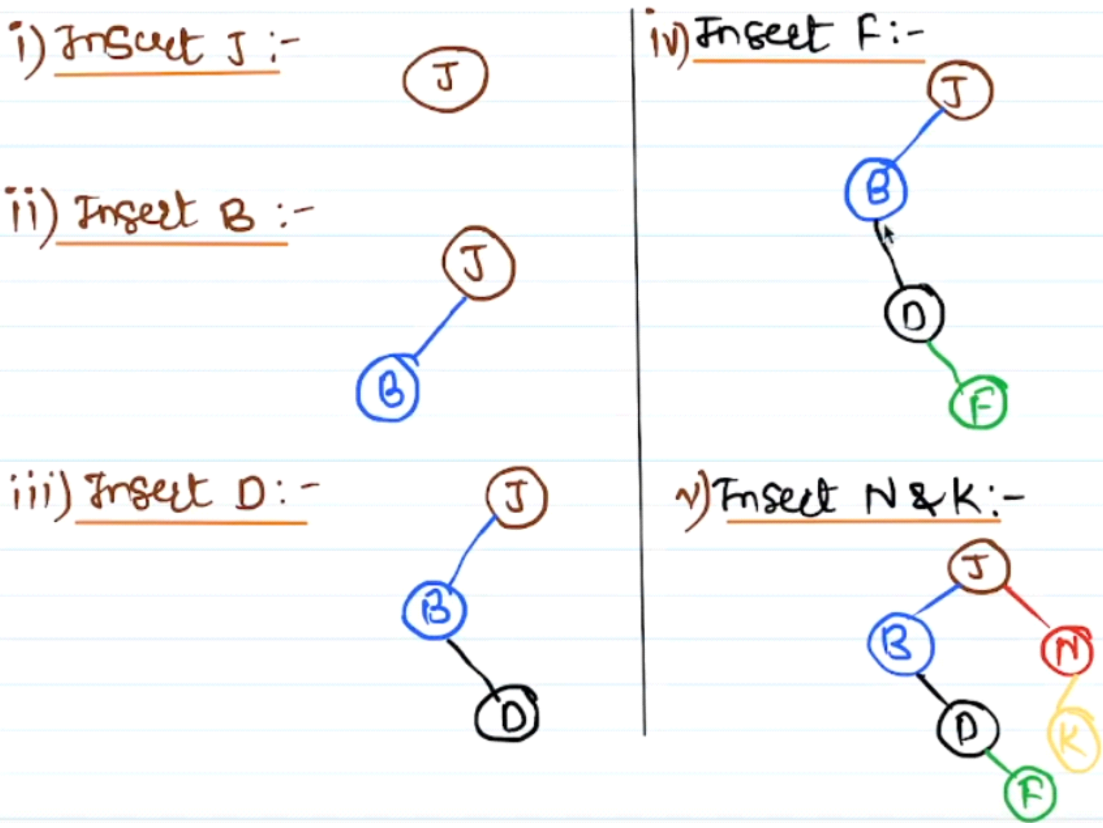
**Answer:**

**Binary Search Tree:** Binary search tree is a data structure that quickly allows us to maintain a sorted list of numbers. In a Binary search tree, the value of left node must be smaller than the parent node, and the value of right node must be greater than the parent node. This rule is applied recursively to the left and right subtrees of the root.

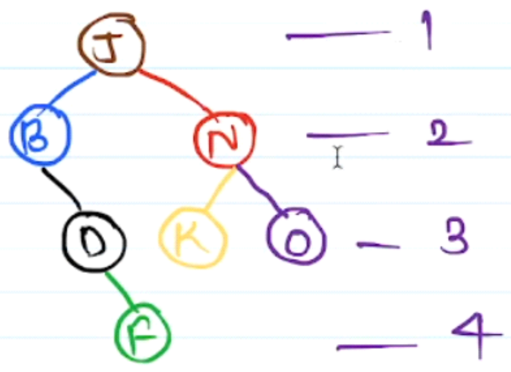
**Ex:**



**BST for J, B, D, F, N, K, O:**



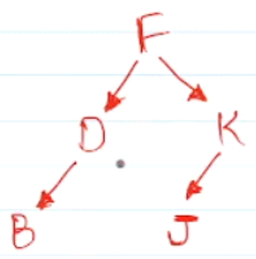
vi) Insert O :-



Binary Search Tree

$\boxed{\therefore \text{ Height of the BST} = 4}$

C) What is threaded binary trees? Give the threaded binary tree of the following binary tree

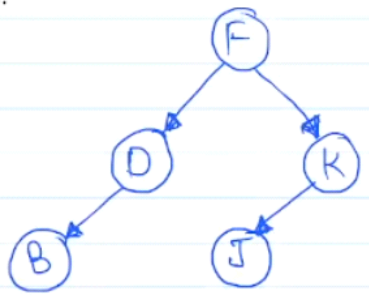


Answer:-

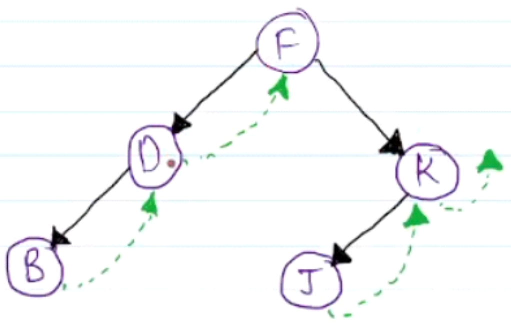
Threaded binary tree (TBT):

A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its INORDER successor. By doing this threading we avoid the recursive method of traversing a Tree, which makes use of stacks and consumes a lot of memory and time.

Given binary tree is:

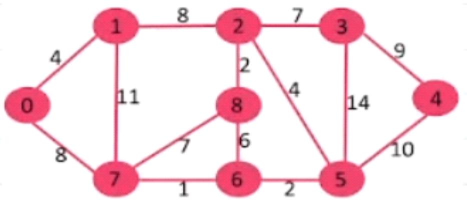


The Inorder (Left-Root-Right) traversal for the above tree is -BDFJK
So the TBT for given tree is -



D has no right child and its inorder successor is F and so a thread has been made in between them. Similarly, for B and J.
K has no right child but it has no inorder successor even, so it has a hanging thread.

Q.5) A) What is graph? Find the shortest path using Dijkstra algorithm. Assume starting node is 0.

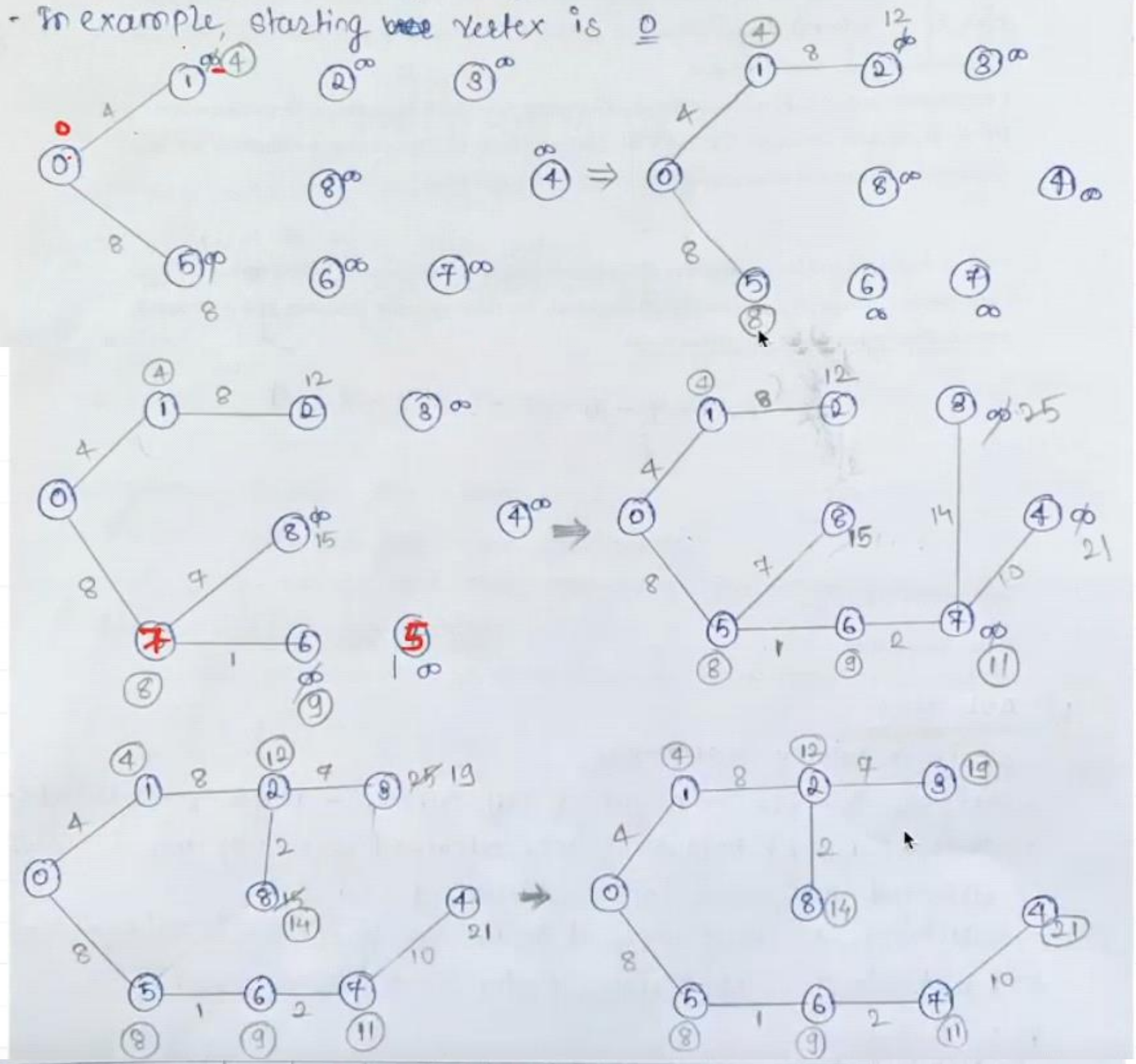


**Answer:**

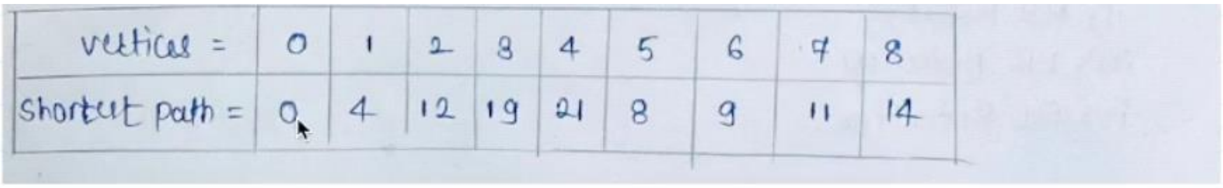
- Dijkstra's algorithm is used to find/solve single source shortest path prob.
- Distance updation =

$$\boxed{\begin{array}{l} \text{if } d[u] + c[u,v] < d[v] \text{ then,} \\ \quad d[v] = d[u] + c[u,v] \end{array}}$$

- In this we have relax an edge. Once we do relaxing we cannot update the distance further
- In example, starting ~~vee~~ vertex is $\underline{0}$



|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| vertices = | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |



| vertices = | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Shortest path = | 0 | 4 | 12 | 19 | 21 | 8 | 9 | 11 | 14 |

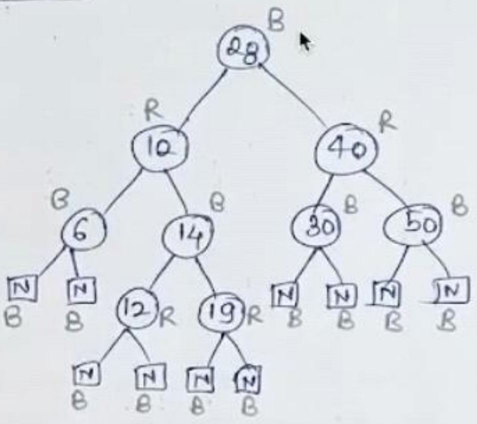
B) Explain the in brief the following
i) red black tree
ii) m-way search tree
iii) b tree
iv) b+ tree
v) sparse matrix
vi) AVL tree

**Answer:**

i) Red Black Tree:-
Red black tree is a binary search tree with one extra bit of storage per node

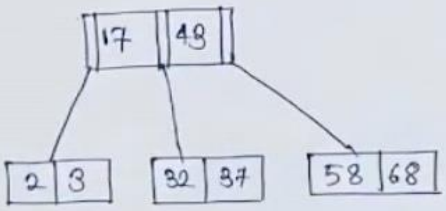
| left | color | key | right |
|---|---|---|---|

* Properties:-
1. Every node is red or black
2. The root is black
3. Every leaf is black
4. If node is red then both its children's are black
5. For each node, all paths from nodes to descendent leaves contain the same number of black nodes



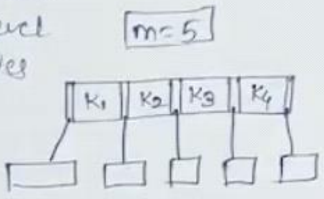
ii) m-way search Tree:-
- m-way search tree is multi way and similar to binary search tree
- A search tree in which node can have m pointers and (m-1) keys.
- ex:- A 3-way search tree have 2 keys & 3 pointers

| $P_0$ | $K_1$ | $P_1$ | $K_2$ | $P_2$ |
|---|---|---|---|---|



iii) B-tree:-

- It is a balanced m-way tree
- It is a generalisation of BST in which a node can have more than one key & more than 2 children
- All leaf node must be at the same level
- B. Tree of order 'm' has following properties
  - a) Every node has 'm' children
  - b) min children: leaf → 0
    - root → 2
    - internal node → $\lceil \frac{m}{2} \rceil$

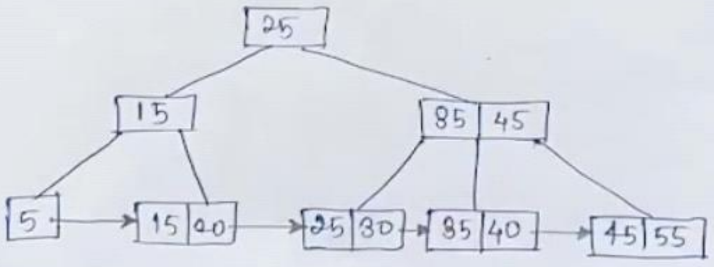


- Every node has (m-1) keys.
- min keys ⇒ root node = 1
  - all other nodes = $\lceil \frac{m}{2} \rceil - 1$

iv) B+- tree:-

- It is an extension of B. Tree which allows insertion, deletion & search operations.
- all leaves are at the same level
- The root has atleast two children
- Each node except root node can have maximum of 'm' children and atleast $\lceil m/2 \rceil$ children
- Each node can contain a maximum of m-1 keys and minimum of $\lceil \frac{m}{2} \rceil - 1$ keys

Ex:-



Applications:-

1) multilevel indexing
2) Database indexing.

v) Sparx matrix :- The situation in which a matrix contain more number of zero elements then non-zero elements, such matrix is called sparx matrix.

Sparx matrix Representation :-

1> Array Representation:- will represent in three field row, column, value

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 2 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 3 \end{bmatrix}$$

2> Linked List Representation :- will store in four fields row, column, value, next node.

vi) AVL Tree :-

- It is a binary search tree
- Height of Tree = Height of Left subtree – Height of right subtree
- Only {-1, 0, 1} heights of tree allowed then only tree is called balanced otherwise it is unbalanced.
- Searching, insertion & deletion in AVL tree done is O(logn) time

AVL Rotations :- (If Balance factor is other than -1,0,1)

i> LL Rotation

ii) RR Rotation

iii> LR Rotation

iv> RL Rotation