Assignment : 2

20DCS103

4DCSE2

Date
Page

(1) Write an algorithm for push and pop operations of stack using an array.

→ Algo for push operation :

Step 1 : Start
Step 2 : int s[size], top ; // top = -1.
Step 3 : if stack is full
then print stack overflow.
Step 4 : else
top ← top + 1 ;
s[top] = data
Step 5 : end

→ Algo for pop operation :

Step 1 : Start
Step 2 : int s[size], top
Step 3 : if stack is empty
then print stack underflow
Step 4 : else
top ← top - 1
Step 5 : end

(2) Explain how stack data structure is used in recursive Fibonacci function.

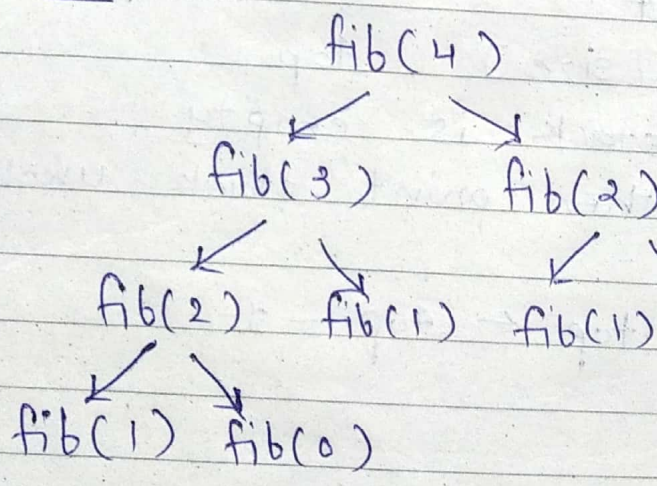
→ When a function calls itself, then it is called recursion.

→ Stack data structure is used in fibonacci function to store the elements on first in last out principle.

→ Using stack, a function can push data into a stack and pop it out when recursive function's values is needed. Hence, using stack to store the fibonacci of each element becomes essential in making efficient program.

→ Ex :- $fib(n) = fib(n-1) + fib(n-2)$

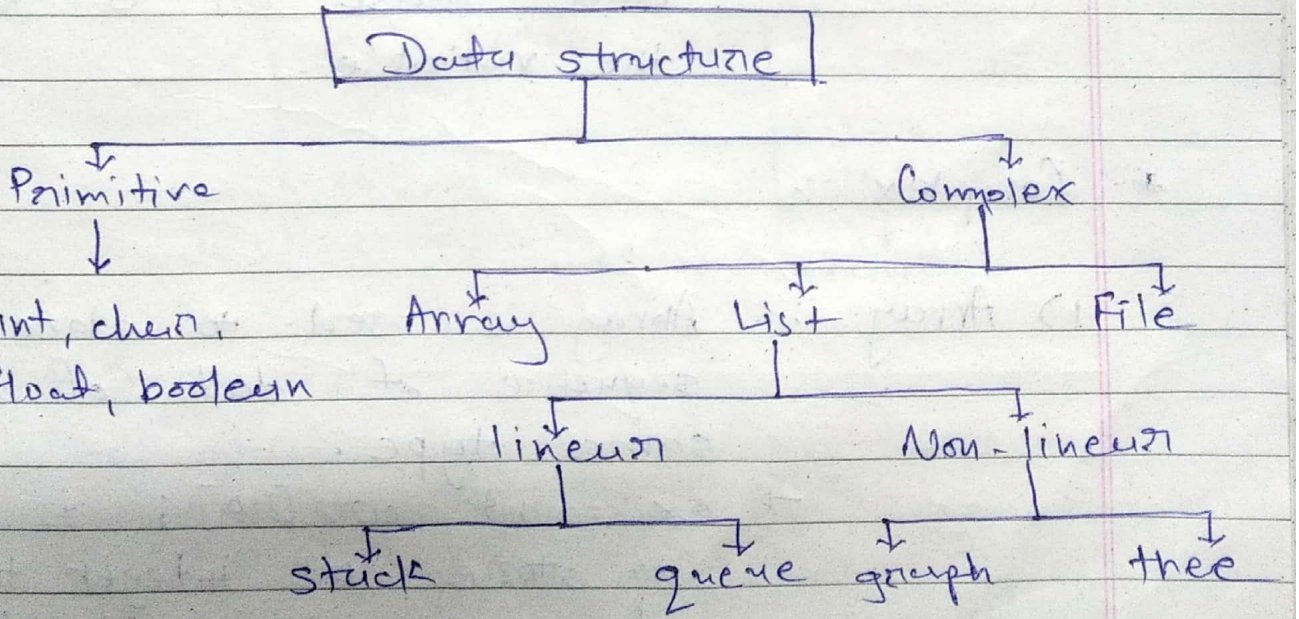
$n = 4$.



→ This is the structure of fibonacci function using stack data structure.

(3) Give the difference between data type and data structure. Also mention types of data structure with an example.

Data type	Data structure
→ The data type is the type of user defined variable to which a value can be assigned which is of the same type.	→ The data structure is the collection of data of different types of data types.
→ It can hold the value but not data.	→ It can hold multiple types of data within a single object.
→ Ex: int, char, float, etc...	→ Ex: stack, queue, tree, graph, etc.



* primitive :

1) int : int types of variable can store the integer type of value
→ ex : 1, 2, -3, ...

2) char : char is used to store the single character to the variable.
→ ex : 'a', 'b', ...

3) float : float is used to store the floating point values to the variable.
→ ex : 1.2, 1.1, 0.1, ...

4) boolean : boolean data types are used to store True or 1 and False or 0 to the variable.

* Complex :

1) Array : Array is used to store the sequence of data of the same type.
→ ex :- int arr[10]; is used to store 10 integer types of values.

2) List :

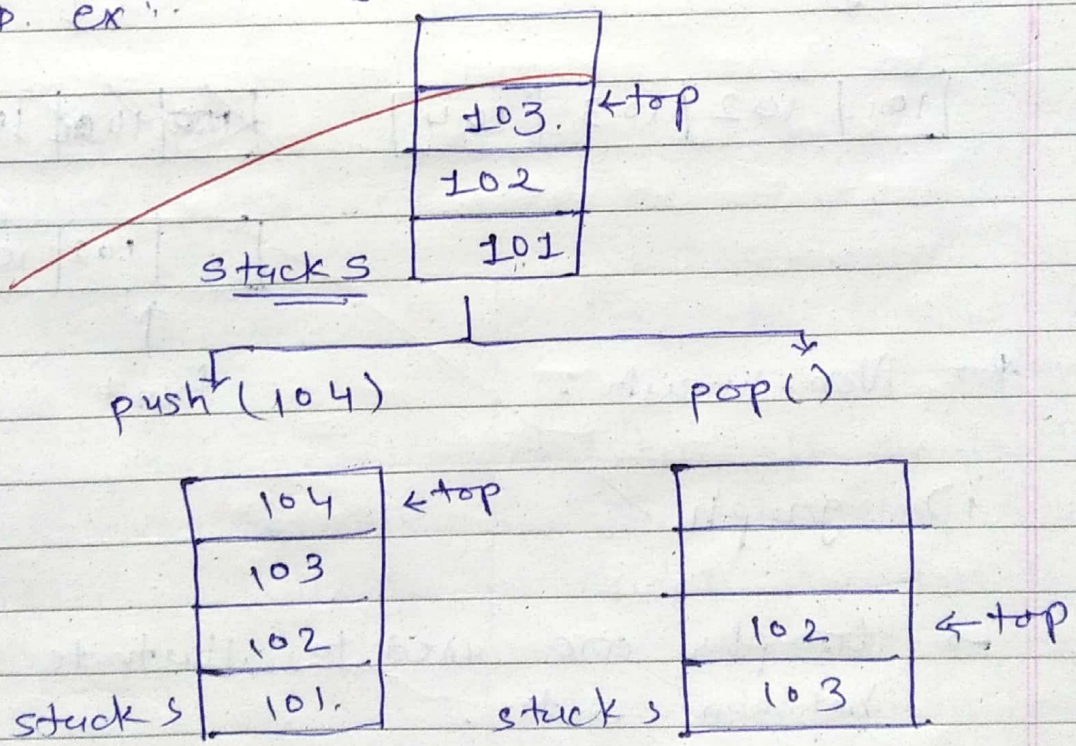
*. Linear :

i) Stack :

→ Stack is used to store data when last first in last out concept is needed.

→ In this data structure, elements can be pushed or poped from the rear only.

→ ex :



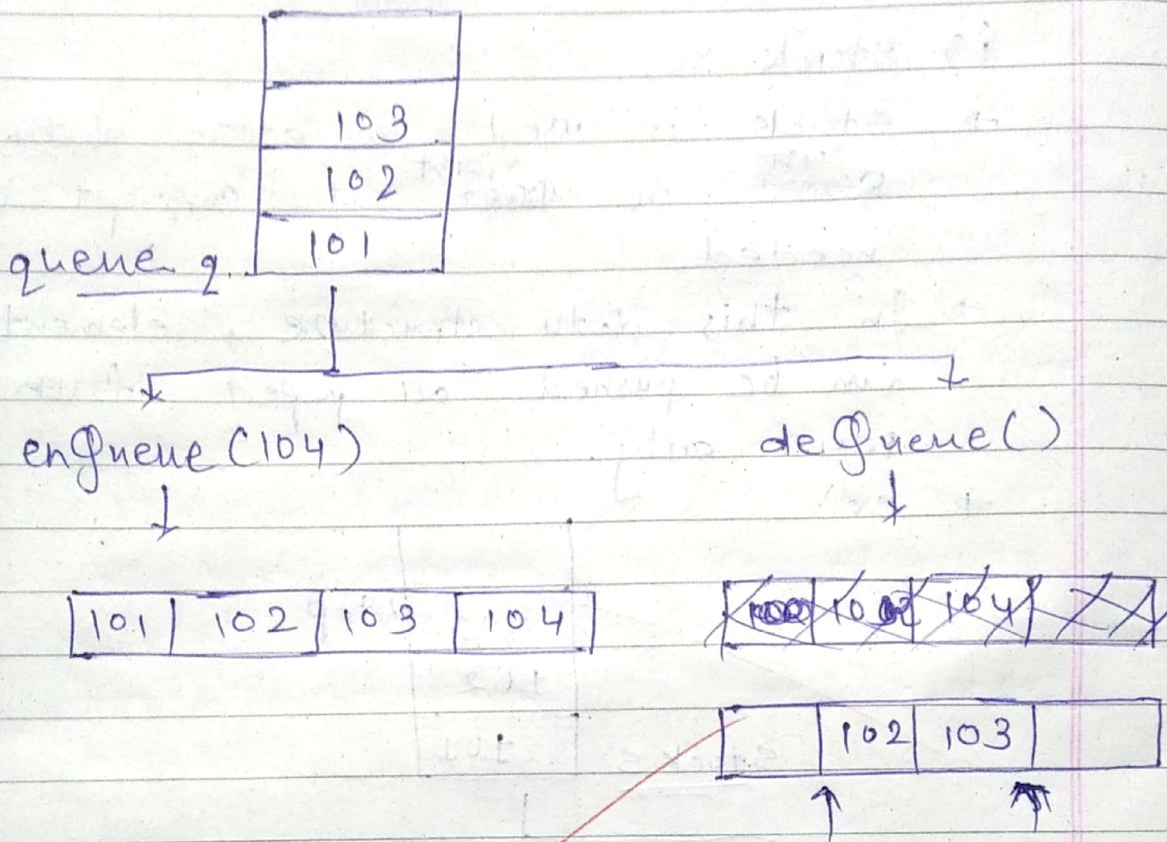
ii) queue.

→ queue is the data structure which is based on the concept of first in first out.

→ In this type of data structure, element can be removed from front

and can be added from rear only.

→ ex :-



** → Non-linear :

1) graph

→ Graphs are used to illustrate relationship between data.

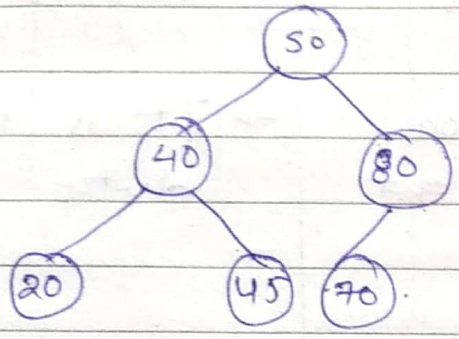
→ Graph is used to present data that are too numerous or complicated to be described.

→ ex :- simple graphs, ---

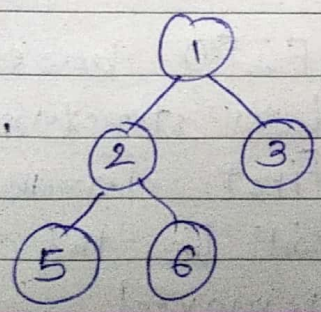
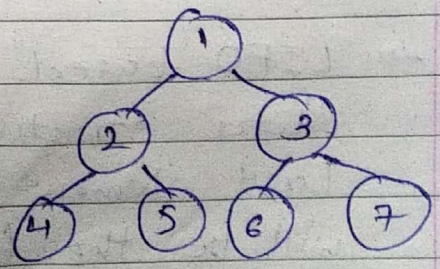
ii) Tree

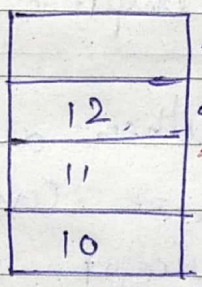
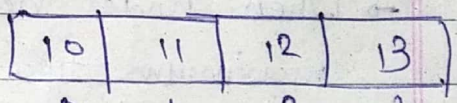
→ Tree is used to structre the data in an efficient way such the operations like searching, deletion can be done in moderate time.

→ ex :-



(4) Differentiate :-

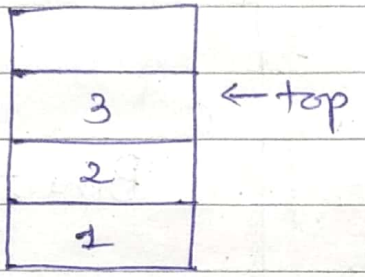
i) Binary tree	Complete binary tree
→ When nodes have maximum two children then it is called binary tree.	→ When all the levels are completely filled except the last level, which is filled from the left, then it is called complete binary tree
→ ex :- 	→ 

ii) Stack	Array
→ It's a linear data structure represented by a sequential collection of elements in fixed order.	→ It's a collection of related data values called elements each identified by an indexed array.
→ It is based on last in first out principal.	→ It is index based data structure.
→ The last element inserted will be the first element removed.	→ The elements can be accessed by their indices.
→ ex:  	→ ex:- int arr[4] = {1,2,3,4} 

iii) Stack	Queue
→ LIFO based linear data structure. Last element inserted will be the first element removed.	→ FIFO base linear data structure. First element inserted will be the first element removed.

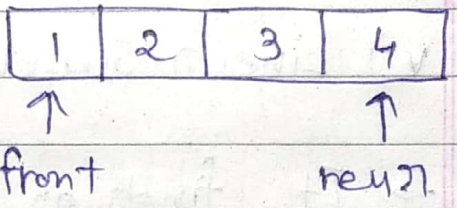
→ Insertion and deletion is performed from the end only.

→ ex:-



→ Insertion will be performed from rear and deletion will be performed from front only.

→ ex:- Queue Q



iv) Singly LL.

→ One single node contains the data of itself and the address of the next node.

→ It occupies less memory, as it has 2 fields.

→ Complexity of insertion and deletion is O(n)

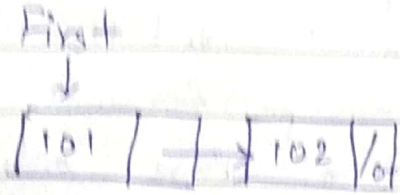
Doubly LL.

→ One node contains the address of the previous node and next node as well. and the data.

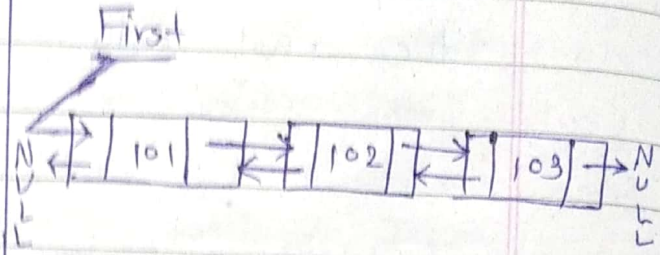
→ It occupies more memory as it has 3 fields.

→ Complexity of insertion and deletion is O(1).

→ ex:



→ ex:



v) Linear Search

→ It finds an element in the list by searching the element sequentially until the last element is found in the list.

→ It is not mandatory that elements should be in sorted order

→ It is based on sequential approach.

→ Time complexity is $O(n)$.

Binary Search

→ It finds the middle elements in the list recursively until the middle element is matched with a searched element.

→ It is mandatory that elements should be in sorted order.

→ It is based on divide and conquer approach.

→ Time complexity is $O(\log_2 n)$.

(5) Create a doubly linked list using following data that arranges data in descending order. Show list after each insertion.
Addresses : 9020, 6110, 8990, 5011, 6229.
Elements : 15, 11, 09, 13, 18.

→ data arrangement in descending order :-

1) Head → NULL

2) Head → 15 → NULL
NULL ← 15 → NULL
9020.

3) Head → 15 → 11 → NULL
NULL ← 15 ⇄ 11 → NULL
9020 6110.

4) Head → 15 → 11 → 9 → NULL
NULL ← 15 ⇄ 11 ⇄ 9 → NULL
9020 6110 8990.

5) Head → 15 → 13 → 11 → 9 → NULL
NULL ← 15 ⇄ 13 ⇄ 11 ⇄ 9 → NULL
9020 5011 6110 8990

6) Head → 18 → 15 → 13 → 11 → 9 → NULL
NULL ← 18 ⇄ 15 ⇄ 13 ⇄ 11 ⇄ 9 → NULL
6229 9020 5011 6110 8990.

(6) A faculty stored data of 50 students of a class having ID from 1 to 50 in ascending order and the faculty allows all students to sit in the class randomly. Each student have two information : ID, and the location of next present student of the class.

1) Which data structure is useful for this application ?

→ Singly linked list is appropriate for the given application as each student has two information : ID and the location of next present student.

2) In the class, all the students are present except ID 12 and 25. After 10 minutes, ID 12 arrives. He sits at random in the class. Write an algorithm such that data of ID 12 is arranged in ascending order.

→ Algorithm :-

Step 1 : start
Step 2 : Node *t ← first
Step 3 : Node *n ← new Node(12)
Step 4 : while (t → next is NOT NULL)
do
if t → data < n → data
t ← t → next.

else

$n \rightarrow next \leftrightarrow t \rightarrow next$

$t \rightarrow next \leftarrow n$

Step 5 : end.

(7) What is the maximum and minimum height of binary tree with 12 nodes ?
What is the drawback of binary search tree

→ The maximum height of a binary tree with n Nodes is given by : $n-1$

∴ maximum height of a tree with 12 nodes is 11

→ The minimum height of a binary tree with n nodes is given by : $floor(\log_2 n)$

∴ minimum height of a tree with 12 nodes in : $floor(\log_2 12)$

$= floor(3.58)$

$= 3$

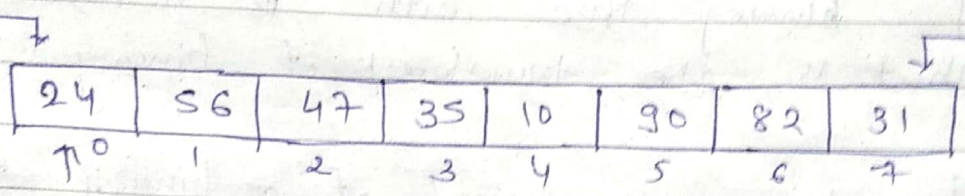
(8) What is quick sort and what is it's worst case time complexity ?
Sort the following using quick sort :
24, 56, 47, 35, 10, 90, 82, 31.

→ Quick sort algorithm is based on divide and conquer principal.

→ It picks an element as pivot and partitions the given array around picked pivot.

→ Here, we are taking first element of an array as pivot.

→ 24, 56, 47, 35, 10, 90, 82, 31



pivot

i j

i = 0 → 24, 56, 47, 35, 10, 90, 82, 31
j = 7 P

i j

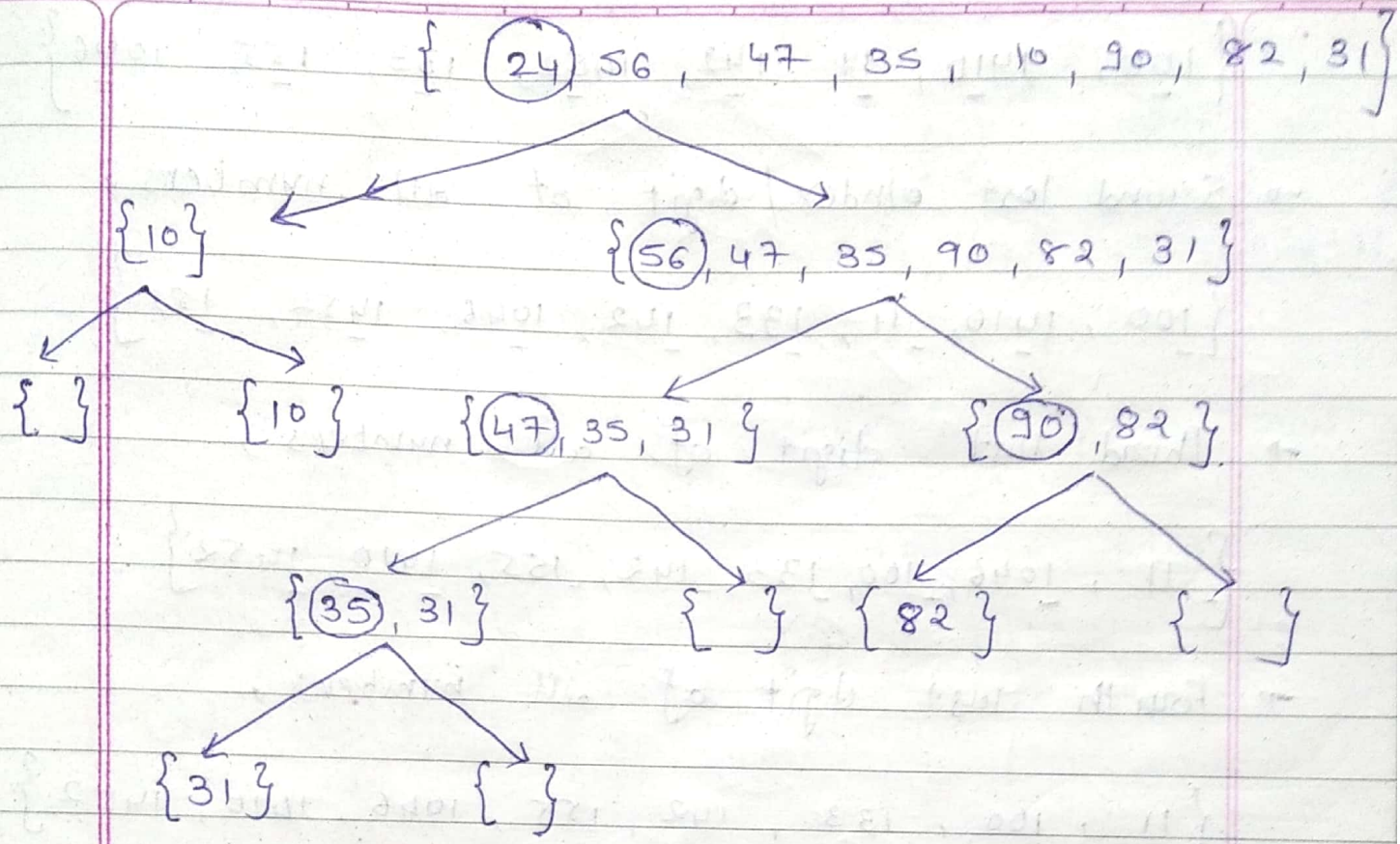
i = 1 → 24, 56, 47, 35, 10, 90, 82, 31
j = 6

likewise we can perform quick sort algorithm to get the sorted array.

→ To make it simple, let us draw the diagram which easily depicts how quick sort algorithm works taking first element as pivot.

→ Time complexity:

Best case : $O(n \cdot \log n)$
Average case : $O(n \cdot \log n)$
Worst case : $O(n^2)$



→ This is how partition algorithm of quick sort works.

→ As a result we will get the sorted array as :

10	24	31	35	47	56	82	90
0	1	2	3	4	5	6	7

(9) Sort the following data using Radix sort. : 100, 11, 155, 1410, 142, 1452, 1331046.

→ {100, 11, 155, 1410, 142, 1452, 1331046}

→ Taking last digit of all numbers in the array.

{100, 1410, 11, 142, 1452, 133, 155, 1046}

→ Second last index / digit of all numbers,

{100, 1410, 11, 133, 142, 1046, 1452, 155}

→ Third last digit of all numbers,

{11, 1046, 100, 133, 142, 155, 1410, 1452}

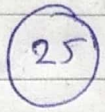
→ Fourth last digit of all numbers,

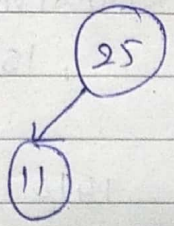
{11, 100, 133, 142, 155, 1046, 1410, 1452}

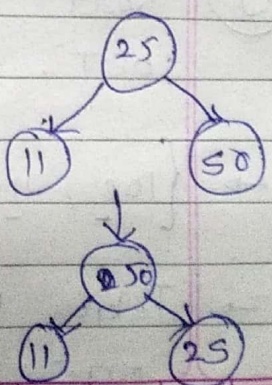
which is a sorted array.

(10) Create the Max Heap tree for the following data. Draw the tree after each insertion : 25, 11, 50, 9, 71, 35, 95, 30.

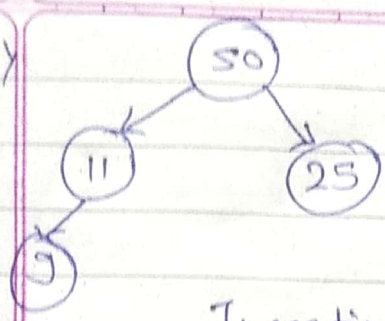
→ Given : 25, 11, 50, 9, 71, 35, 95, 30.

1) 

2) 

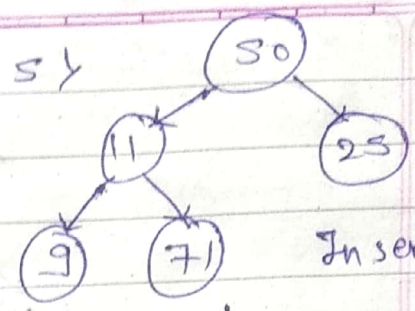
3) 

4)

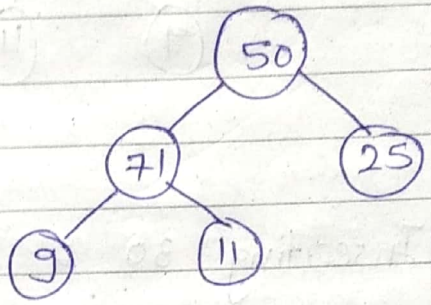


Inserting : 9.

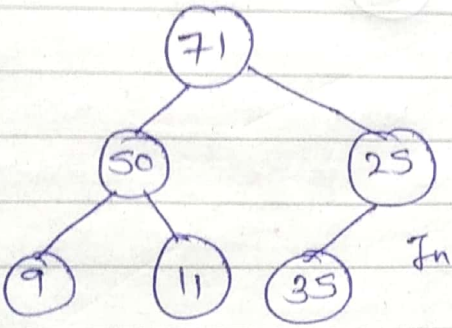
5)



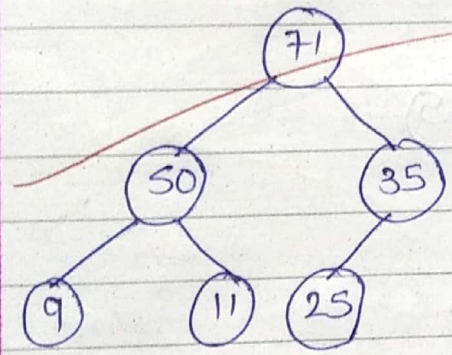
Inserting : 71.

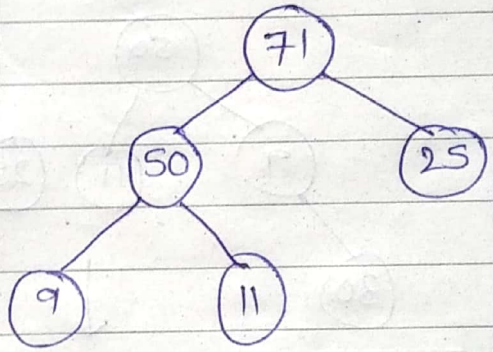


6)

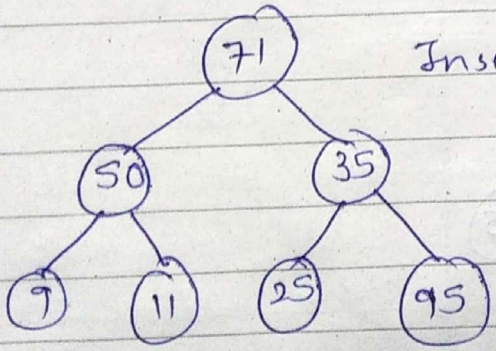


Inserting 35



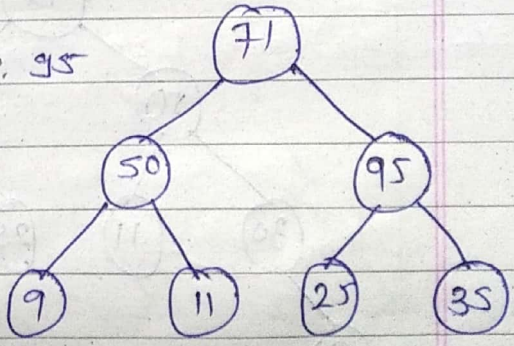


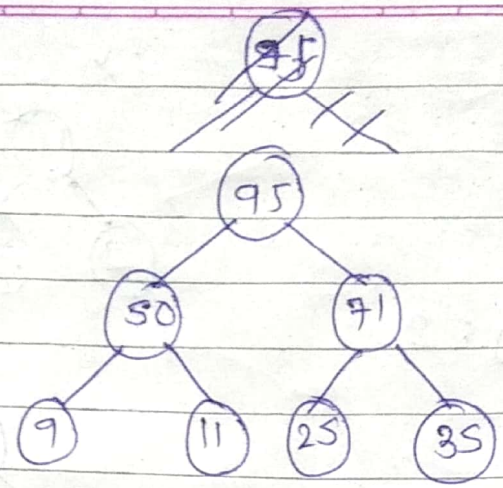
7)



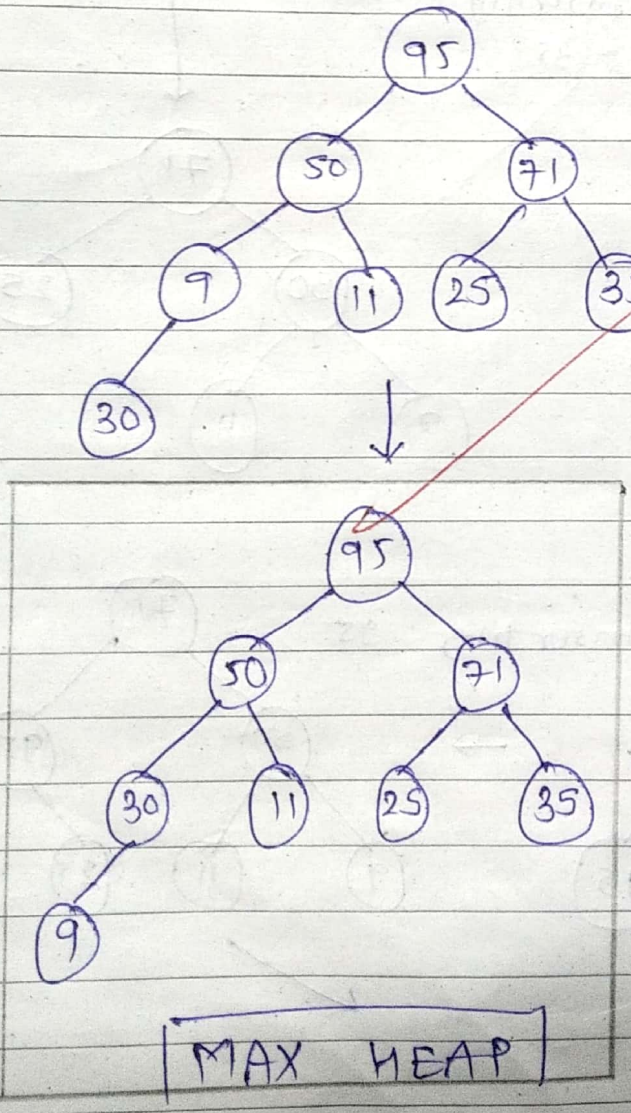
Inserting : 95

→



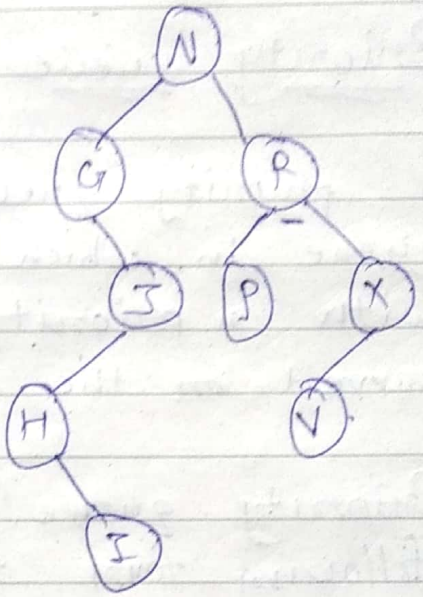


8) Inserting 30 :



(11) Create the binary search tree for the following data : N, R, X, G, P, J, H, V, I. After creation of entire tree, delete node 'R' and draw final tree.

→ Creating BST :-

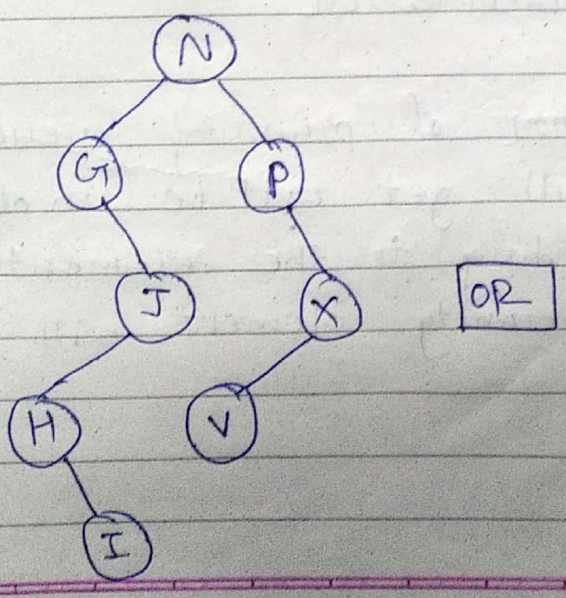


→ in order traversal is : G, H, I, J, N, P, R, V, X

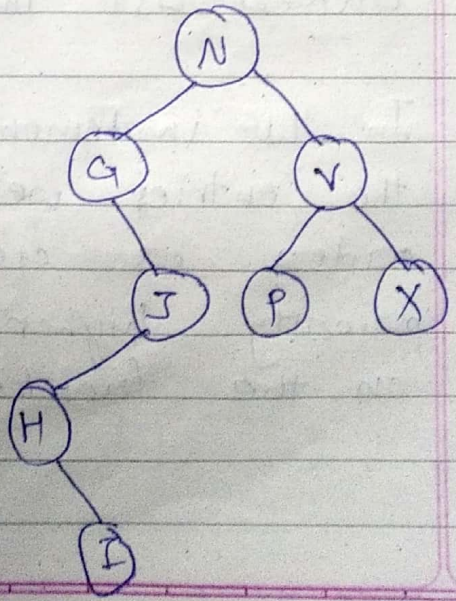
∴ which is sorted.

→ Now, removing node (R) :-

when we remove node (R) then it's inorder predecessor (P) or inorder successor (V) will take place of (R). ∴ Resultant bst is...



OR



(12) Define priority queue. List out different techniques to implement priority queue. Explain one of the techniques with suitable example.

→ Priority queue :-

A priority queue is a special type of queue in which each element is associated with a priority value and elements are served on the basis of their priority.

→ Priority queue can be implemented using following four techniques :

1) Arrays
2) Linked List
3) Heap
4) BST : Binary Search Tree.

→ Linked List implementation :-

→ In LL implementation of priority queue, the entries we will get will be in descending order as according to the element's priority higher priority element will be on the front.

→ Example :-

```
#include <bits/stdc++.h>
using namespace std;

class Node
{
  public :
      int data ;
      int pry ;
      int * next ;

      Node * Node ( int data , int p )
      {
          Node * temp = new Node ( data ) ;
          temp -> priority = p ;
          temp -> next = NULL ;
          return temp ;
      }

      int peek ( Node ** head )
      {
          return ( * head -> data ) ;
      }
      void pop ( Node ** head )
      {
          Node * temp = * head ;
          ( * head ) = ( * head ) -> next ;
          free ( temp ) ;
      }
```

```
void push ( Node ** head , int d, int p )
{
  Node * start = (*head);
  Node * temp = new Node(d, p);

  if ( (*head) -> pry > p )
  {
    temp -> next = *head;
    (*head) = temp;
  }
  else
  {
    while ( start -> next != NULL
            && start -> next -> pry < p)
    {
      start = start -> next;
    }
    temp -> next = start -> next;
    start -> next = temp;
  }
}

int isEmpty ( Node ** head)
{
  return ((*head) == NULL);
}
```

```
int main()
{
    Node *qu = new Node(4, 1);
    Node t;
    t.push(&qu, 5, 2);
    t.push(&qu, 6, 3);
    t.push(&qu, 8, 0);

    while ( ! isEmpty(&qu) )
    {
        cout << " " << peek(&qu);
    } pop(&qu);

    return 0;
}
```

→ Output :-

8 4 5 6

→ Time complexity for pushing an element into a queue is : O(n)

→ This is how we can implement priority queue using linked list data structure.

(13) Explain the impact of data structure on performance of algorithm with the help of suitable example.

→ Data structure is about organizing and storing data in a computer in such a way that we can perform certain operations using certain algorithm in a very efficient way.

→ We can have certain types of algorithms such as ... for, sorting, deleting, searching, to update, etc ...

→ Let us have an example of storing & searching an element in a particular data structures.

1) For storing elements into an array, one has to perform insert operation which takes O(1) time.

→ Now for the same array if we want to perform delete operation then to search an element into an array the worst case time consumed is O(n).

13) For storing elements into a binary search tree, we need to perform insert operation frequently and ~~for~~ ~~that~~ for that it takes O(log n) time.

→ Now for the same binary search tree, to perform search operation, O(log n) time is consumed.

→ In conclusion, storing data and searching data in array data structure is costly in time while in case of binary search trees, it takes less time for the same operations.

→ At the end, selecting a right data structure for storing data is essential in terms of time complexity and space complexity as well.

(14) Explain merge sort with detailed algorithm.

→ Merge sort works on the principle of divide and conquer algorithm. It breaks down a list into several sublists until each sublist consists of a single element, and merging those sublists in a manner that

results into a sorted manner.

→ Algorithm :

→ Merge :-

Steps

Merge (A , p , q , r)

1 $n_1 \leftarrow q - p + 1$.
2 $n_2 \leftarrow r - q$
3 Creation of arrays
$L[1 \ldots n_1 + 1]$ and $R[1 \ldots n_2 + 1]$
4 for $i \leftarrow 1$ up to n_1
5 do $L[i] \leftarrow A[p + i - 1]$
6 for $j \leftarrow 1$ up to n_2
7 do $R[i] \leftarrow A[q + j]$
8 $L[n_1 + 1] \leftarrow \infty$
9 $R[n_2 + 1] \leftarrow \infty$
10 $i \leftarrow 1$
11 $j \leftarrow 1$
12 for $k \leftarrow p$ up to r
13 do if $L[i] \leq R[j]$
14 then $A[k] \leftarrow L[i]$
15 $i \leftarrow i + 1$
16 else $A[k] \leftarrow R[j]$
17. $j \leftarrow j + 1$

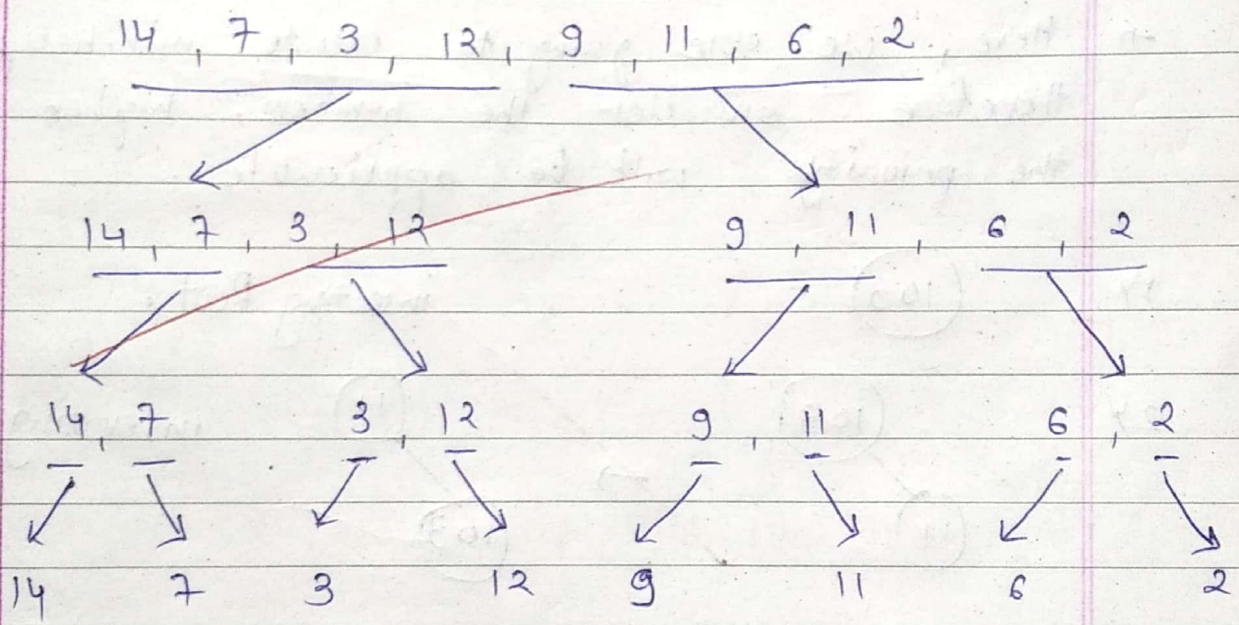
Ans. Merge sort :-

Steps

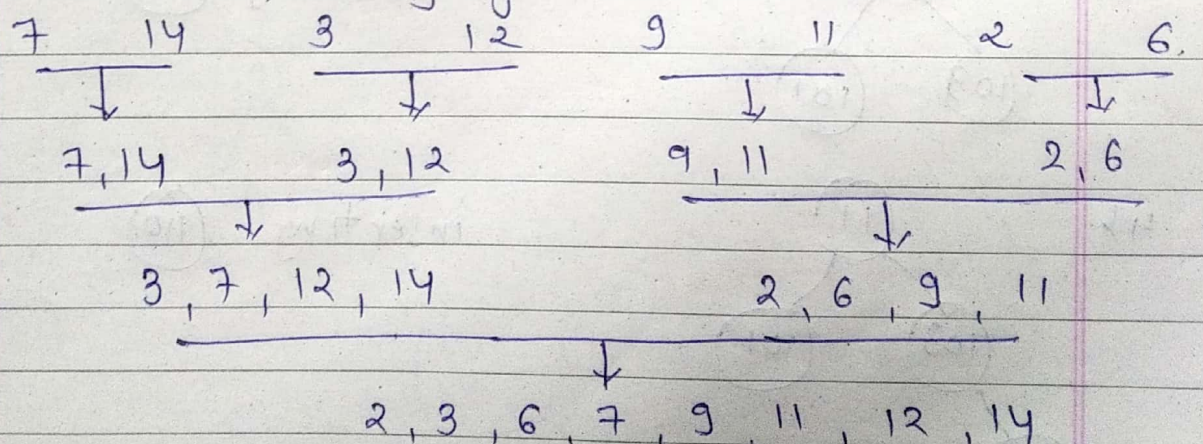
MergeSort (A , p , r)

1. if $p < r$
2. then $q \leftarrow \lfloor (p+r)/2 \rfloor$
3. MergeSort (A , p , q)
4. MergeSort (A , q+1 , r)
5. Merge ~~Sort~~ (A , p , q , r)

Ex:



Now, Merging sublists ...



→ Hence, we get the sorted array by using merge sort algorithm.

(15) A heap tree stores priorities (or priority element) pairs at nodes. Demonstrate step by step, the operation of building min-heap on the array. [Here only priorities of elements are given.]

[103, 11, 101, 110, 111, 119, 19, 91]

→ Given : 103, 11, 101, 110, 111, 119, 19, 91.

→ Here, we are going to create min-heap therefore smaller the number, higher the priority will be applicable.

1) 103

inserting Root :

2) 103, 11

11, 103

inserting 11

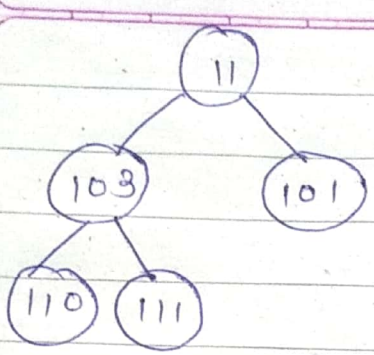
3) 11, 103, 101

inserting 101

4) 11, 103, 101, 110

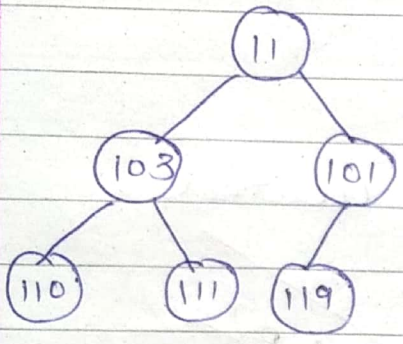
inserting 110

5)



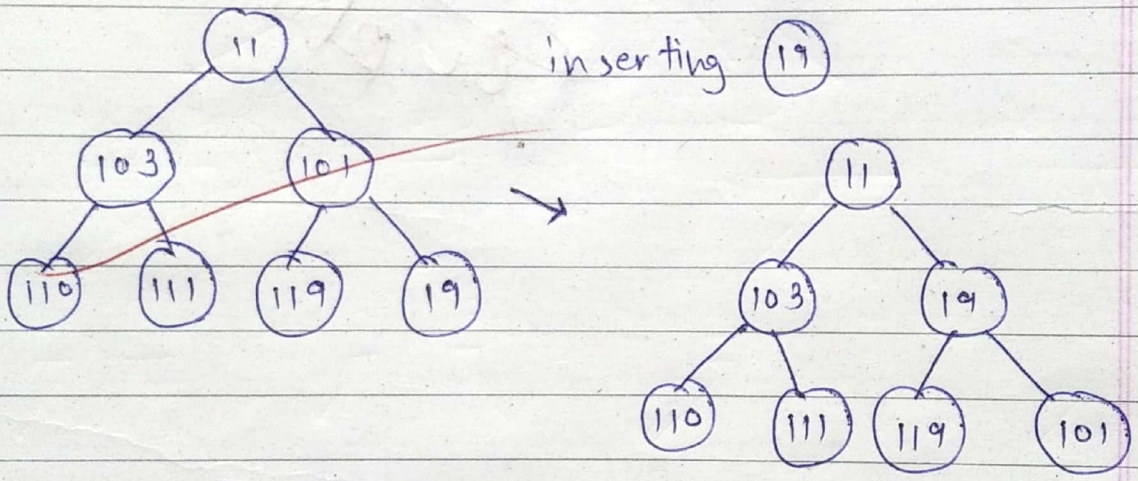
inserting (111)

6)



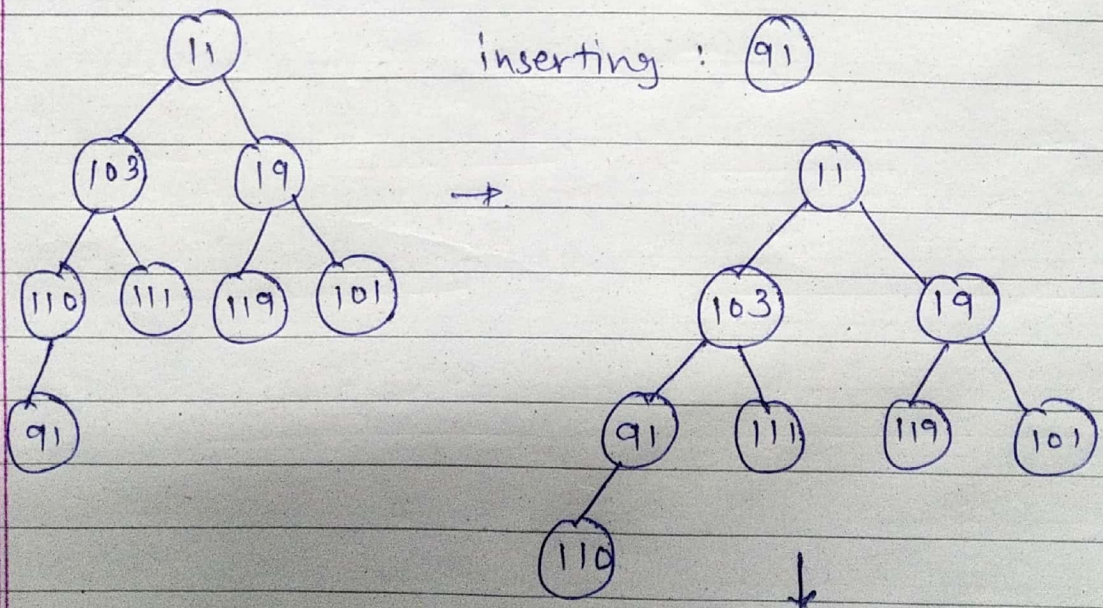
inserting (119)

7)

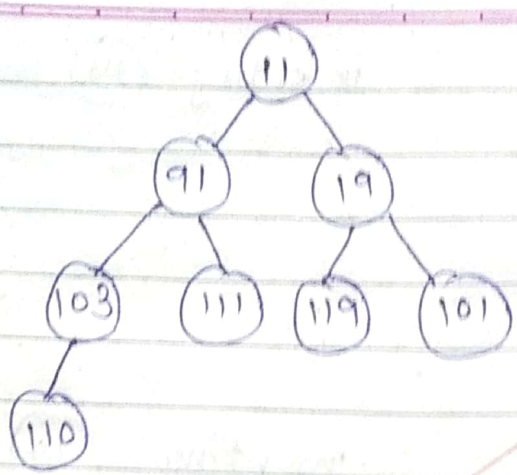


inserting (19)

8)



inserting : (91)



∴ Resultant min-heap is created.

× Put 22