MAHARSHI DAYANAND UNIVERSITY, ROHTAK

NAAC Accredited 'A' GRADE

(To be filled in by the Evaluator at the time of evaluation
To be stapled on the top of online print out of Answer Book)
(Checking Assistant to make sure No column is left Blank)

Roll No. 8017613

Name Bazgha Razi

Exam B.Tech (CSE) Subject Data Structures & Algorithms

Semester 3rd Syllabus Code PCC-CSE-203G

Question Paper ID No. 3128 Date : 24/06/2021

TO BE FILLED BY EXAMINERS ONLY

Q.No	A	B	C	D	E	F	TOTAL
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							

TOTAL IN FIGURES

TOTAL MARKS IN WORDS

Bag ID No.

Examiner ID

Full Signature of Examiner

Full Signature of Checking Assistant

C.O.E.
STAMP

a) University Roll No. (In figures): 8017613 In words: Eight Zero One Seven Six One Three

b) Name of the Student: Bazgha Razi

c) Class/Semester: B.Tech (CSE) / 3rd semester

d) Name of the Paper: Data Structure & Algorithm

e) Question Paper ID: OIE-3128 f) Total No. of Pages Written by Candidate: 2

g) Date of Examination: 24/06/2021

h) Signature of Student: Bazgha Razi

Ans 1 a) Insertion Sort

It is the simple sorting algorithm. In this, we insert each element onto its proper place in the sorted array. It is less sufficient than the other sorting algorithms.

Example: Unsorted Array:

14	33	27	10	35	19	42	44

Insertion sort compares first two element. So, 14 & 33 is already in ascending order. Insertion sort moves ahead & compare 33 with 27. It finds that 27<33 so, we swap them

14	27	33	10	35	19	42	44

Now, compare 14 & 27, so after comparing we move to compare 33 & 10 so, we swap. and again compair 27 & 10 then again swap at last compare 14 & 10 then swap them Hence after applying this, we get.

10	14	27	33	35	19	42	44

Similarly, compare 33 & 35, they are already in ascending order so compare 35 & 19 and after comparing swap then then compare 33 & 19 then swap then compare 27 & 19 and swap then compare 14 & 19 so it is already in ascending order so no swapping is done.

10	14	19	27	33	35	42	44

Hence the array is sorted using insertion sort.

Ans 1 b) Application of binary tree.

- Binary tree is used to represent a non-linear data structure.
- Binary tree plays a vital role in software application.
- Binary tree is also used in the searching algorithms such as heap sort.
- Binary tree is also used in expression evaluation. The leaves of the binary tree are the operands & the internal nodes are the operators.

Ans 2c) Binary search tree

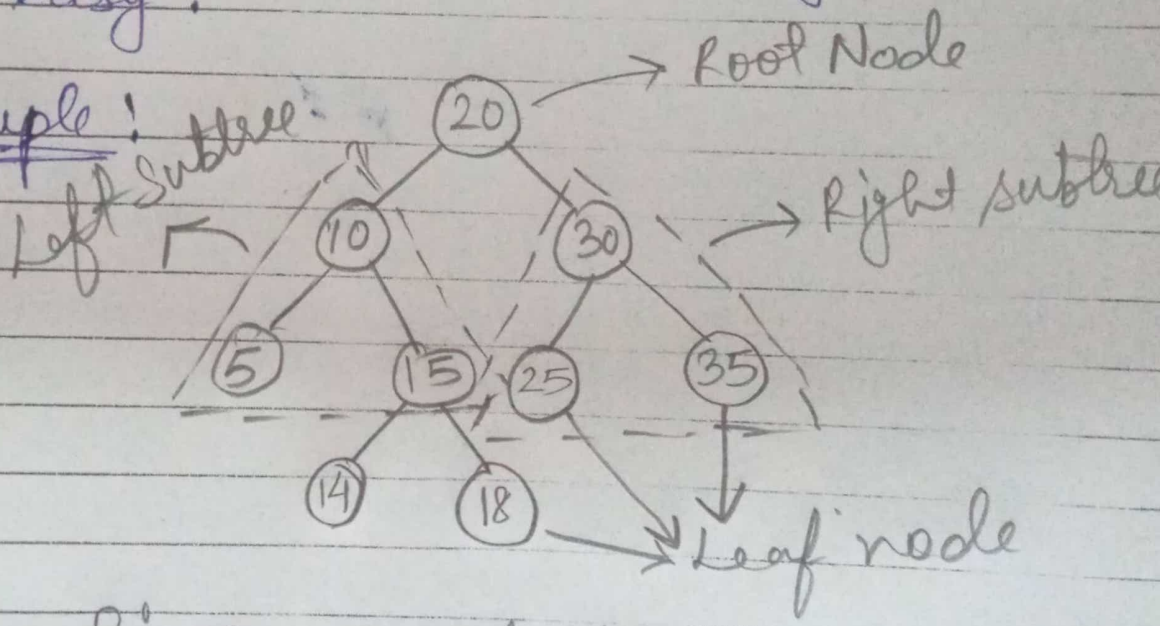
It is a class of binary tree, in which the nodes are arranged in a specific order.

In binary search tree, the value of all the nodes in the left sub-tree is less than the value of the root.

Value of all the nodes in the right-subtree is greater than or equal to the root.

Using binary search tree, searching becomes very easy.

Example:

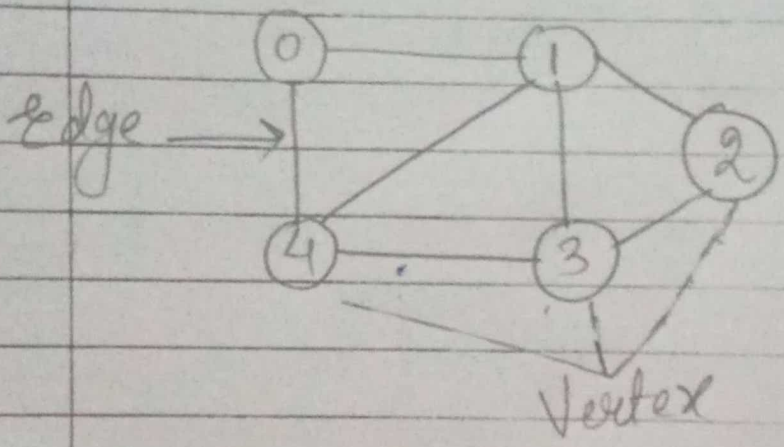


Binary search tree

Ans 1 d) Graph

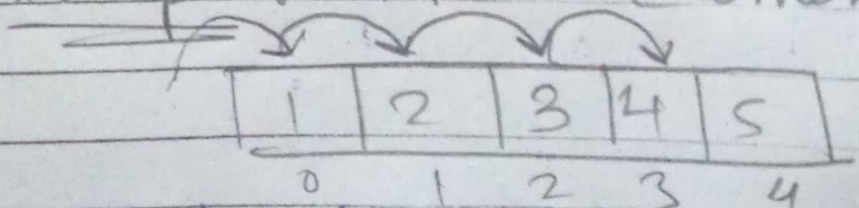
The graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred as vertices and the edges are lines that connect any two nodes in the graph.

Example of graph data structure:



Ans 1 e) Linear Search is also known as simple search. In this we have to search each element without skipping a single element from the list! If the element present in the list then it will return the index of that element. Time Complexity of linear search is O(n).

Example: Find Element 4

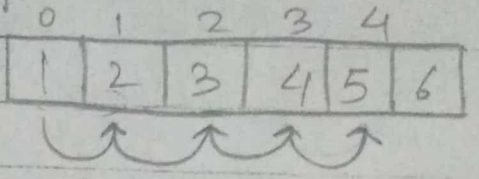


4 is found at position index 3

Ans3 Searching

It is the process of finding element or the position of the element in the given data

For Example: Find 5



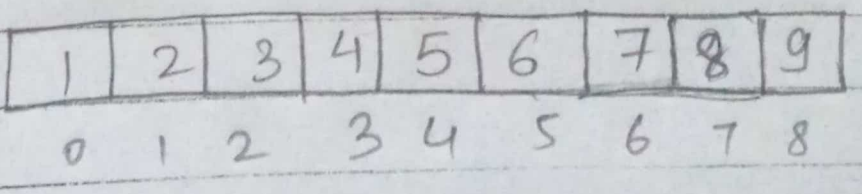
∴ Element 5 is found at position index 4 and this process is known as searching

Binary Search Algorithm

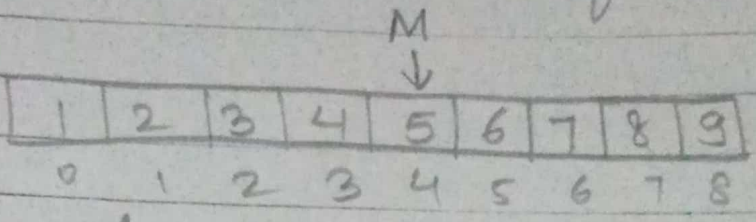
It is an algorithm, its input is sorted list of elements. If an element you're searching in that list, then binary search returns the position where it is located, otherwise it will return null.

Binary search divide the list into half as soon as the middle of a sorted list is found.

Example: Find 3



Divide the list into half



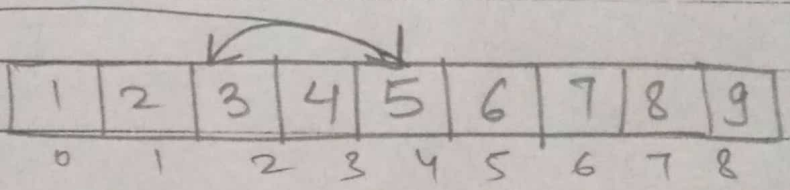
Middle element = M

If the searched no. is = M then return the position of M

If the searched no. is < M then move to the left of list

If the searched no is > M then move to the right of the list

∴ 3 is less than 5



Hence 3 is found at index 2

Binary Search is better than linear search. For large list binary search is better.

Binary search is better than linear search because linear search is time consuming.

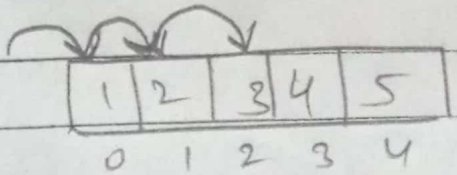
In linear search, we have to check one item at a time without skipping any element, whereas in binary search we have to divide the list into half as soon as the middle of sorted list is found.

Linear search may be implemented on any linear search i.e., singly linked list.
But in binary search, it can be implemented on data structures where two-way traversal is possible.

Time complexity of linear search is O(n) and time complexity of binary search is O(log n)

Example: Find 3

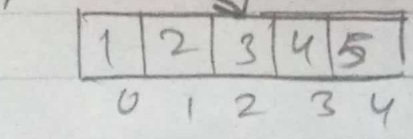
Linear Search



found at index 2

Linear search require 3 steps to find the element

Binary Search



found at index 2

Binary search require only a single step to find the element

Complexity of binary search

Time Complexity = O(log n)

Worst Case Complexity is . O(log n)

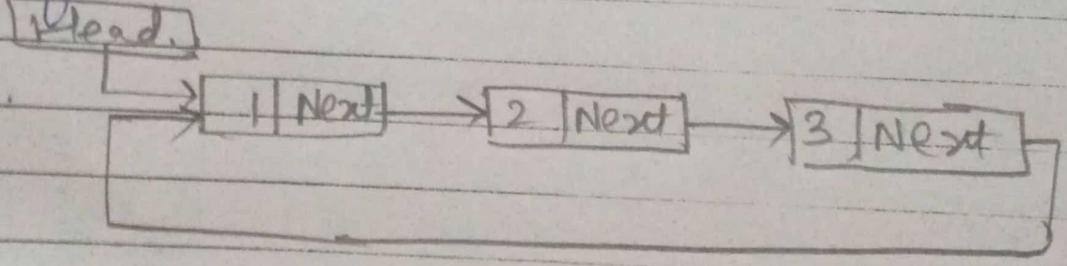
Best Case Complexity O(1)

The best case complexity of binary search is to find the element in the middle position.

Ans 6: Circular linked list.

In a circular singly linked list, the last node of the list contains a pointer to the first node of the list.

We can traverse a circular singly linked list until we reach the same node where we started. The circular linked list has no begening & no ending. There is no null value present in the next part of any of the node.



Algorithm for insertion

1. If PTR = NULL
   write overflow
2. Set New_Node = PTR
3. Set PTR = PTR → Next
4. Set New_Node → Data = Val
5. Set Temp = Head
6. Repeat step 8 while Temp → Next != Head
7. Set Temp = Temp → Next (End of loop)

8. Set New_Node → Next = Head
9. Set Temp → Next = New_Node
10. Set Head = New_Node
11. Exit

Algorithm for Deletion

1. If Head = NULL
   Writes underflow (End of loop)
2. Set PTR = Head
3. Repeat step 4 & step 5 while PTR → ~~Next!!!!~~ Next != He
4. Set PREPTR = PTR
5. Set PTR = PTR → Next
6. Set PREPTR → Next = Head
7. Free PTR
8. Exit

Roll No. 2017613 Page No. 11 Date 24/06/21 Signature

Ans7 a) AVL trees are balanced trees whose worst case time complexity is equivalent to binary search i.e., O(log n).

AVL tree was invented by Adelson Velsky and Landis in 1962. So, tree was named in the honour of their inventors.

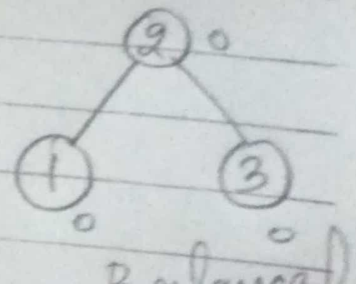
AVL tree depends on two main factors:
i) It should be a binary tree.
ii) It's balanced factor should be 0, 1, -1

Balanced factor mean,
Balanced factor = $H_L - H_R$

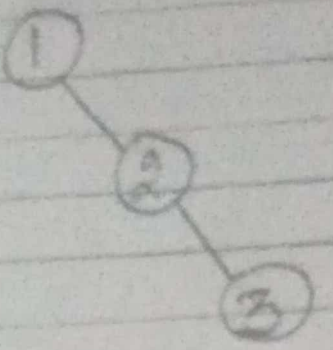
where, H_L = Height of left subtree
H_R = Height of right subtree

Example:

In this tree node 3 and 1 has zero balanced factor because they are leaf node and balance factor of 2 is also zero because the height of the left subtree is also 1 & height of the right subtree is also one so difference is zero. Hence, this tree is balance or we can say it is an AVL tree.

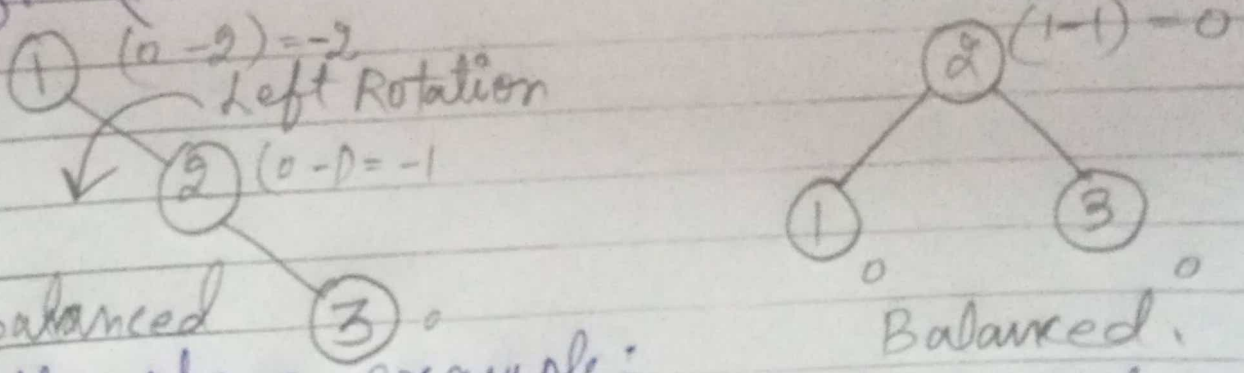


In this figure the balanced factor of 3 is zero as 3 is a leaf node. Balanced factor of 2 is -1 because the difference of height of left tree and height of right subtree is -1. But the balanced factor of 1 is -2 because the difference of height of left subtree and height of right subtree is -2. Hence this tree is unbalanced.



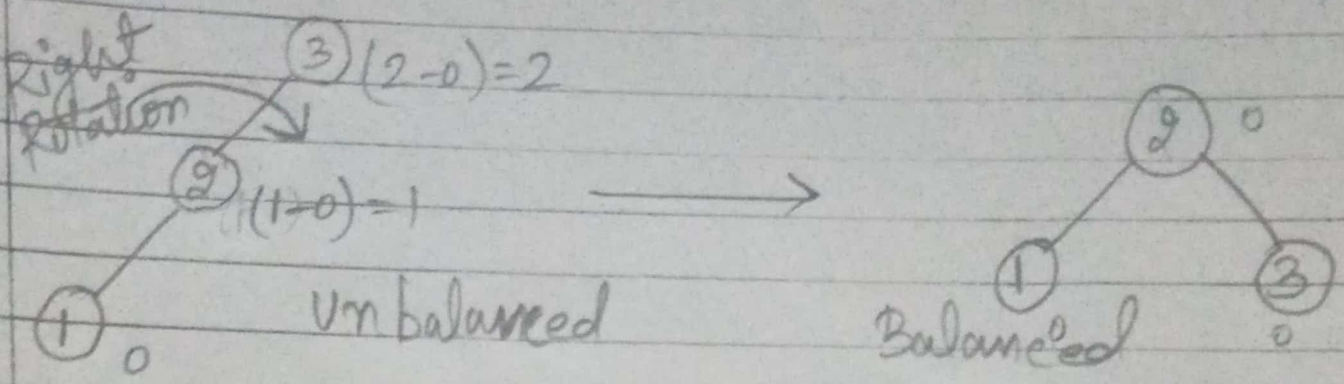
To make these type of unbalanced tree as balanced tree then the following rotation is used

i) Left Rotation



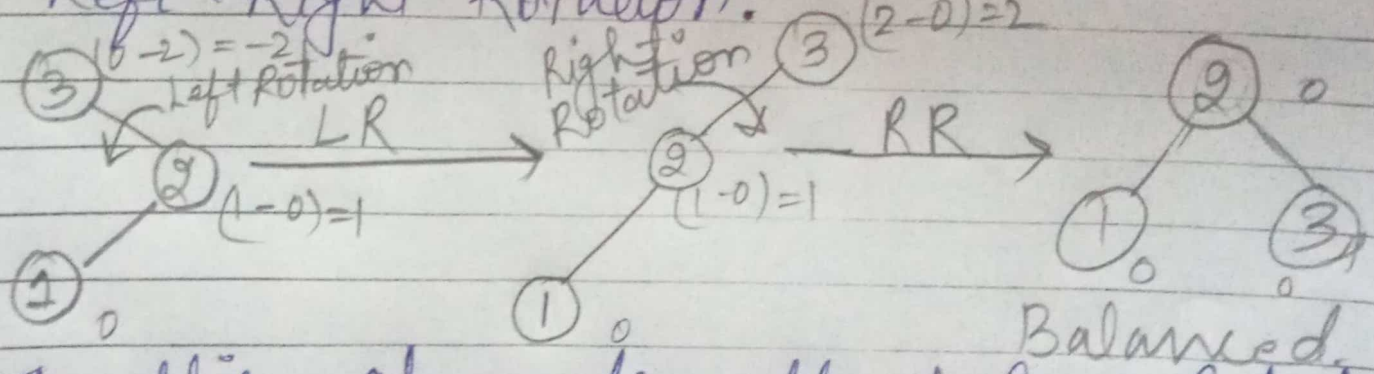
In the above example:
The balanced factor of 3 is 0, 2 is -1 and 1 is -2, so it is unbalanced because the balanced factor of 1 is -2. Hence we have to do a rotation i.e., left rotation to balanced this tree. After left rotation the balanced factor of 1 is 0, 2 is 0 and 3 is also zero. Hence, the tree is balanced.

ii) Right Rotation



In the first figure, we can see that the balanced factor of 1 is 0, 2 is 1 and 3 is 2. So, balanced factor of 3 is 2 therefore it is unbalanced tree. So, we have to do right rotation so that tree becomes balanced. Hence after applying right rotation, the balanced factor of 1 is 0, 2 is 0 & 3 is 0 Hence, the tree is balanced.

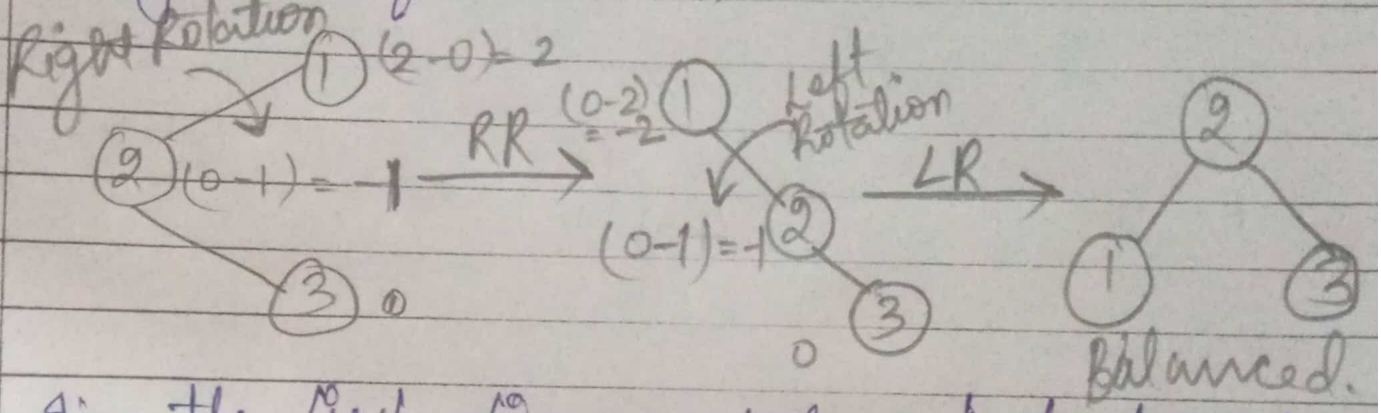
iii) Left-Right Rotation.



In this above tree the balanced factor of 1 is 0, 2 is 1 & 3 is -2 in the first figure. So, we have to do left rotation after left rotation the balanced factor of 1 is 0, 2 is 1 and 3 is 2. It is still unbalanced. So, we have to

do right rotation, after right rotation, balanced factor of 1 is 0, 2 is 0 & 3 is 0. Hence the tree is balanced.
∴ After applying left-right rotation, tree is balanced.

iv) Right - Left Rotation



In the first figure, balanced factor of 1 is 2, 2 is -1, 3 is 0. It is unbalanced because balanced factor of 1 is 2. So, we have to do right rotation. After right rotation balanced factor of 1 is -2, 2 is -1, 3 is 0. It is still unbalanced so we do left rotation. After rotation the balanced factor of 1 is 0, 2 is 0 & 3 is 0. Hence after applying Right left rotation tree is balanced.

Roll No. 8017613 Page No. 15 Date 24/06/21 Signature Reza Rafi

Ans 7 b) B+ Tree

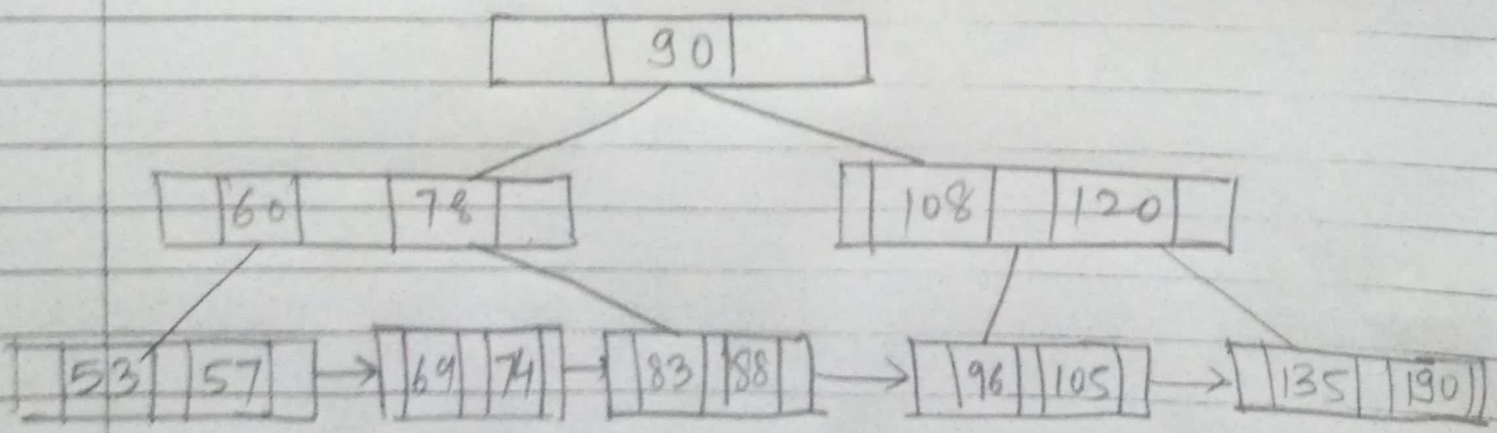
B+ tree is an extension of B Tree which allows efficient insertion, deletion & search operations.

In B+ tree, data can only be stored on the leaf node while internal nodes can only store the key values.

The leaf nodes of B+ tree are linked together in the form of a singly linked list to make the search more efficient.

It is used to store the large amount of data which can not be stored in the main memory.

The internal nodes of B+ tree are often called index nodes. B+ tree of order 3 is shown below:



Insertion in B+ tree

1. Insert the new node as a leaf node.
2. If the leaf node doesn't have required space, split the node & copy the middle node to the next index node.
3. If the index node doesn't have required space, split the node & copy the middle element to the next index page.

Deletion in B+ tree

1. Delete the key & data from the leaves.
2. If the leaf node contains less than min no. of elements, merge down the node with its siblings and delete the key in b/w them.
3. If the index node contains less than minimum no. of elements, merge the node with the sibling & move down the key in b/w them.

Ans: Heap Sort

Heap sort is the improved version of the binary search tree. It does not create a node as in case of binary search tree instead it builds the heap by adjusting the position of elements within the array.

An ordered balanced binary tree is called a Min-heap, where the value at the root of any subtree is less than or equal to the value of either of its children.

An ordered balanced binary tree is called a max-heap, where the value at the root of any subtree is more than or equal to the value of either its children.

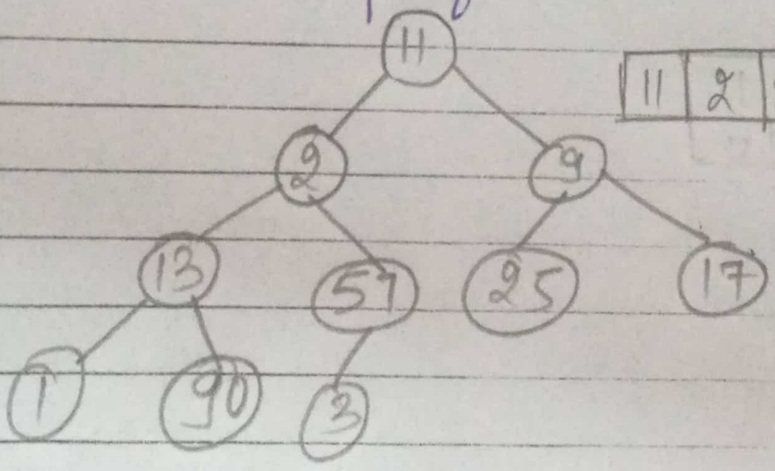
It is not necessary that the two children must be in some order, sometimes the value in the at the left child may be more than the right child or vice-versa.

Basically, there are two phases involved in the sorting of elements using heap sort algorithm are as follows:

- First, start with the construction of a heap by adjusting the array elements.
- Once the heap is created repeatedly eliminate the root element of the heap by shifting it to the end of the array and then store the heap structure with the remaining elements.

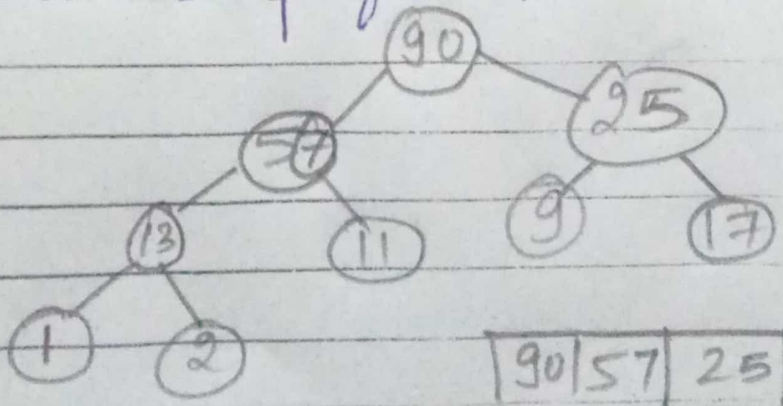
Example: Array: 11, 2, 9, 13, 57, 25, 17, 1, 90, 3
sort this array using heap sort

Create a heap from the array elements.



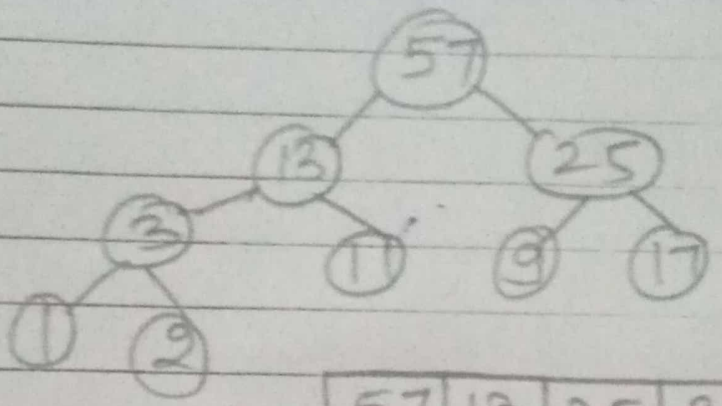
11	2	9	13	57	25	17	1	90	3

The above diagram depicts the binary tree version the list.
Build heap from the above binary tree.



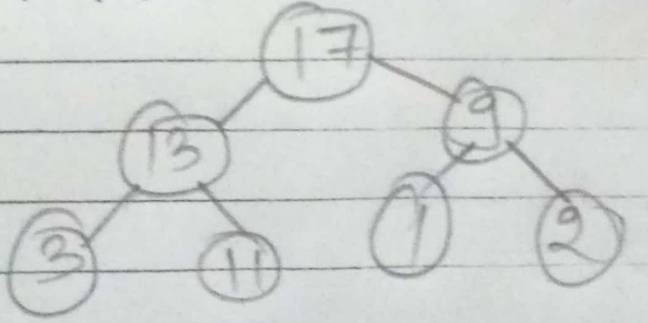
90	57	25	13	11	9	17	1	2	3

Now, the root 90 is moved to the last location by exchanging it with 3. Finally 90 is eliminated from the heap by reducing the maximum index value of the array by 1. The balance elements are then rearranged into the heap. The heap the array look like as shown in the below diagram:



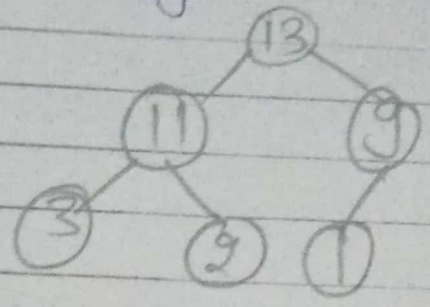
57	13	25	3	11	9	17	1	2	90

Similarly when the current root 57 is exchanged with 2, and eliminated from the heap by reducing the maximum index value of the array by 1. The balance elements are then rearranged into the heap. The heap & array look like as shown below:



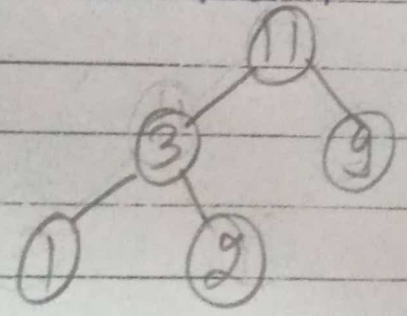
17	13	9	3	11	1	2	25	57	90

Following the same approach, the following phases are followed until the fully sorted array is achieved.



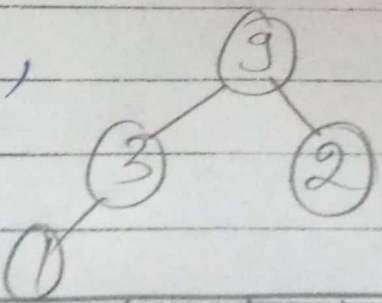
13	11	9	3	2	1	17	25	57	90

following the same approach, the following phases are followed until the fully sorted array is achieved.



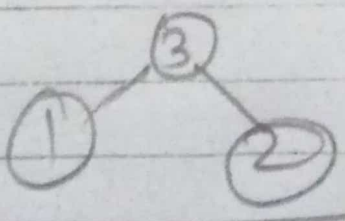
11	3	9	1	2	13	17	25	57	90

Similarly,

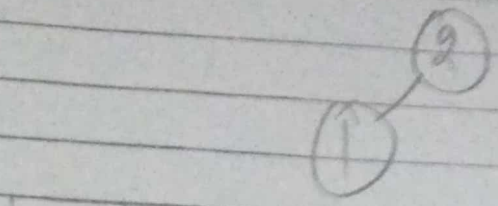


9	3	2	1	11	13	17	25	57	90

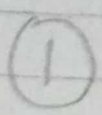
Similarly,



3	1	2	9	11	13	17	25	57	90



2	1	3	9	11	13	17	25	57	90



1	2	3	9	11	13	17	25	57	90

The array is fully sorted using the heap, thus this process is called heapsort.

Algorithm for Heap Sort.

1. Construct a binary tree with given list of elements.
2. Transform the Binary tree into Min heap.
3. Delete the root element from Min heap using heapify method.
4. Put the deleted element into the sorted list.
5. Repeat the same until Min Heap becomes empty.
6. Display the sorted list.