Date:

→ Why Ternary Search is less preffered?

↳ Ternary Search has more operations, the number of comparisions are more.
We can even divide the array to 4 or 5 search space where we get $O(\log_4 n)$ which time complexity is even less.
But Search operations are more so Binary Search is most commonly used & preffered.

Insertion Sort

75, 90, 100, 95, 85, 80
0 1 2 3 4 5

key = 90

↓ i

keep at is

j = i-1

75 90
j i
compare

while
j >= 0 & key < arr[j] :
arr[j+1] = arr[j]
j = j-1
arr[j+1] = key

Date:

0 1 2 3 4 5

75, 90, 100, 95, 85, 80

95 100 ↑ 100

→ key = 90

j = i - 1

i = 1

90 < 75 ✗

→ i = 2

key = 100

j = 1

key < arr[j] => 100 < 90 ✗

→ i = 3

key = 95, j = i - 1 = 3 - 1 = 2 index

↳ 95 < 100 → true

So we do,

arr[j+1] = arr[j]

j = j - 1

↳ Interchanging the elements.

→ i = 4 key = arr[i] = 85

85 < 100 ✓

j = i - 1 = 4 - 1 = 3 index = 95

↳ Inter-change.

while j >= 0 & key < arr[j]

arr[j+1] = arr[j]

j = j - 1

Date:

→ i = 5 , key = 80

↳ J = 4 80 < 75 ✗

arr[j+1] = key

75, 80, 85, 90, 95, 100

↳ Sorted Array.

Best case scenario

10, 20, 30, 40, 50 → Increasing order

n = 5

↳ comp → 4

Time complexity = # comp + # swaps

⇒ (n-1) + 0 ⇒ O(n)

* Given an Array, which is almost sorted, which sorting Algorithm is prefered?

↳ MCQ Questions in FAANG

Insertion Sort ⟹ O(n) Best case

* Which array sorting when highly unsorted?

↳ Quick Sort

Date:

If Array is in increasing order,
Best case time complexity → O(n)

If Array is in decreasing order,
Worst case → $O(n^2)$
→ Sum of n natural numbers. # of comparisons.

→ Find the 2nd largest number in the array → use Bubble sort.
↳ Since it places the largest element at the right end.

→ Find the 2nd smallest number in the array → Selection sort.
Coz it will keep smallest at left/start and then you can get in 2nd pass itself.

HeapSort

↳ Heap Data Structure

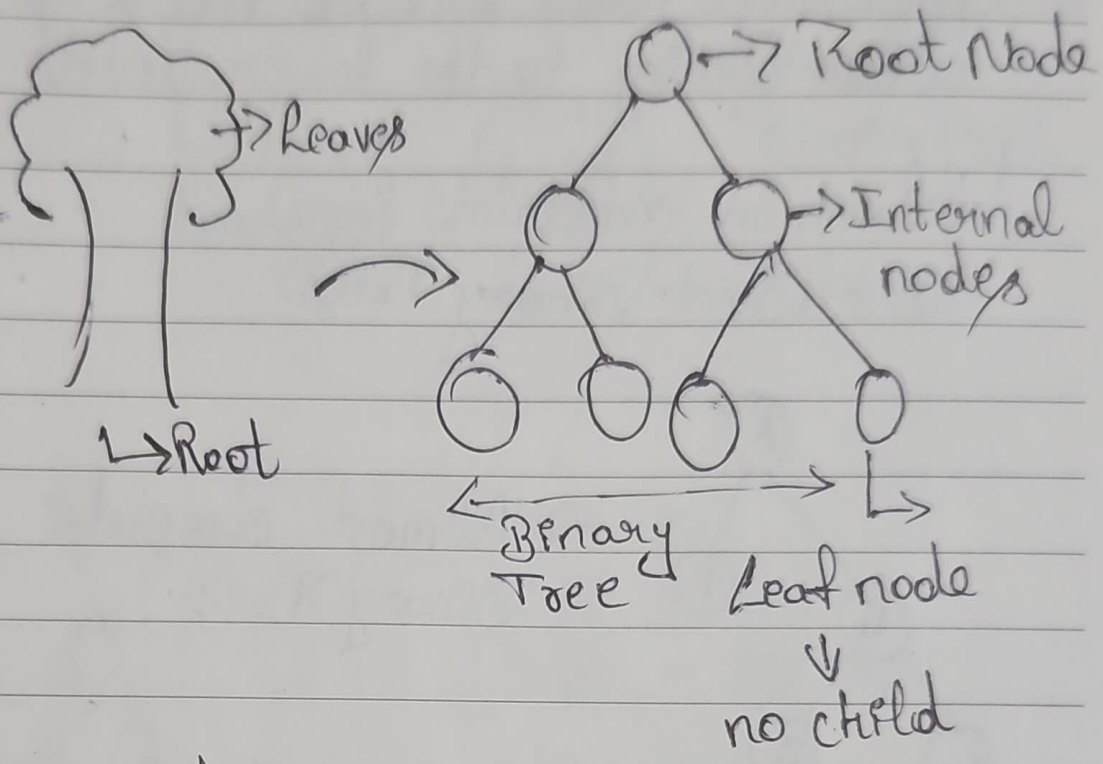
← MaxHeap MinHeap

Fundamentals & Applications

Date:

Tree Data Structure → Basic knowledge to understand Heap

↳ Non-linear Data Structure.



Fundementals:-

1) Binary Tree :- 0, 1, 2 → children
   ↳ Atmost 2 child nodes only then it's considered as Binary Tree.

2) Full Binary Tree vs Almost complete Binary Tree vs Complete Binary Tree

   ⇓

   or

   Perfect Binary Tree

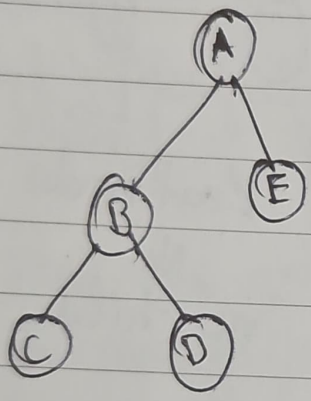
   ↳ Heap DS is based on CBT.

Date:

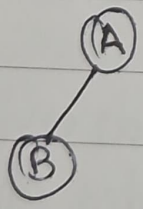
1) Insertion of nodes happens from left to right only

2) Upper level nodes will be filled up before coming to the lower levels

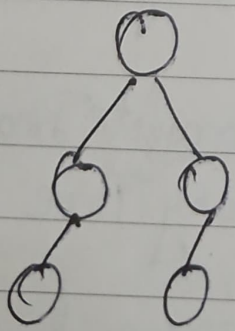
↳ Common constraints for above 3 different Binary Trees.



→ Almost complete Binary Tree.



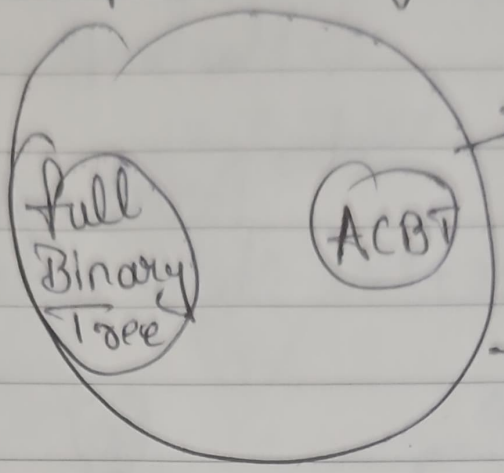
→ Valid ACBT.

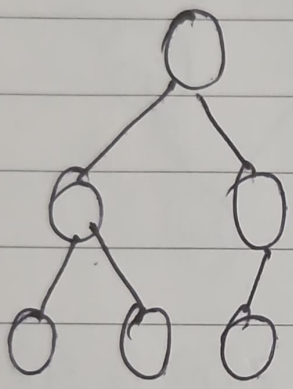


→ Not a valid ACBT

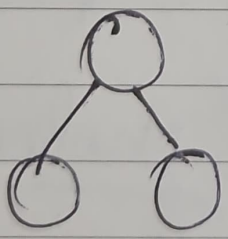
Date:

Complete Binary Tree
↳ Full Binary Tree && Almost CBT
It accepts Both of them





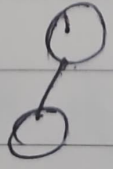
CBT - Yes
ACBT - Yes
FBT - ~~Yes~~ No



CBT - Yes
ACBT - No
FBT - Yes

↳ This is not ACBT coz the last leaf node is complete & it should be there such that last leaf should be empty:

Eg:



→ ACBT

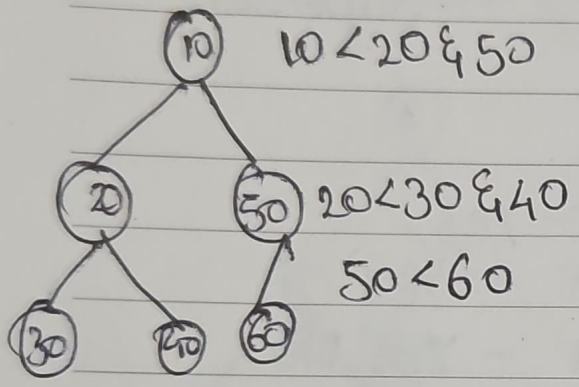
Date:

Heap

↳ Complete Binary Tree

1) Minheap

↳ Parent node data < child node



↳ Valid Minheap

⇓

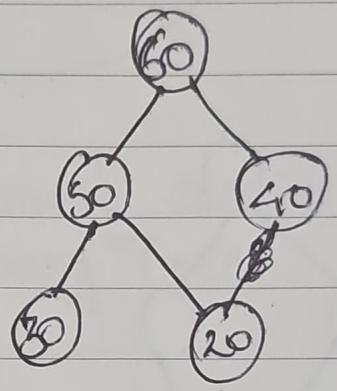
Minimum element from the given array

⇓

O(1)

2) Maxheap

↳ Parent node Data > child node



60 > 50 & 40

50 > 30 & 20

⇓

Valid Maxheap

⇓

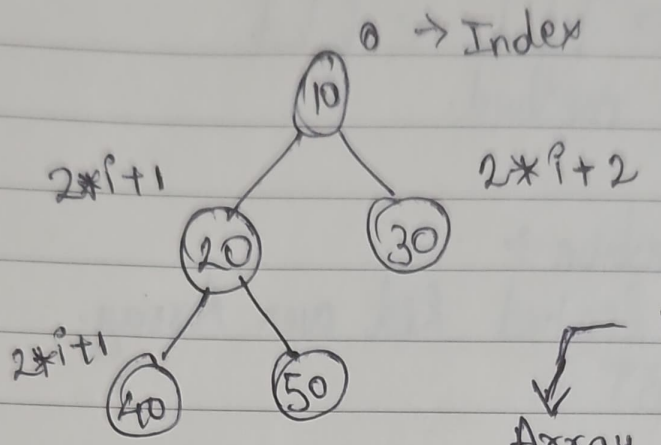
max. element from given array

⇓

O(1)

Date:

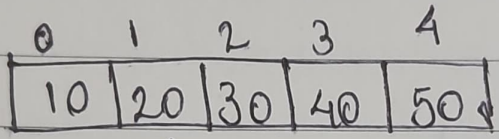
In minheap to find minimum element if we create a minheap & delete the root node we get least element. Since delation always happens from the root node.



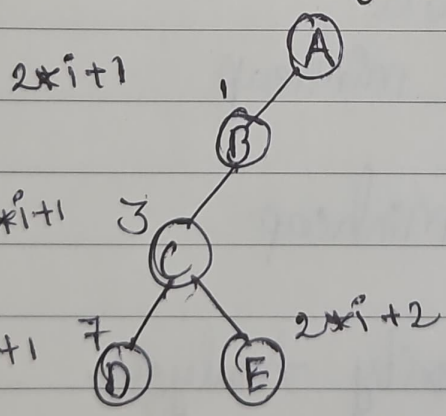
CBT

No wastage of space

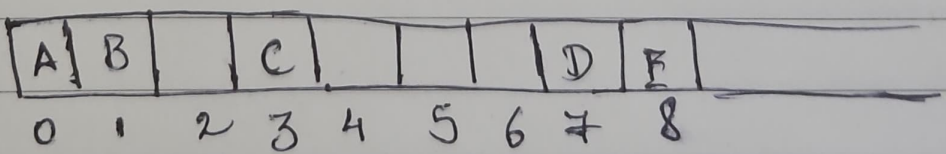
Array



Skewed Binary Tree



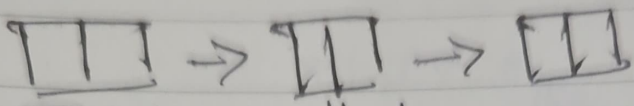
Preferred
↳ Linked List



Array → Lot of wastage of space.

Date:

Linked list → Skewed BT
↳ Not a contiguous allocation of memory.



Pointer method.

Interview Question →
Why prefer Linked list over Array in Skewed BT ?

↳ Operations

1) Insertion
2) Deletion
3) Build minheap

↳ of minheap.

And Time complexity Analysis.