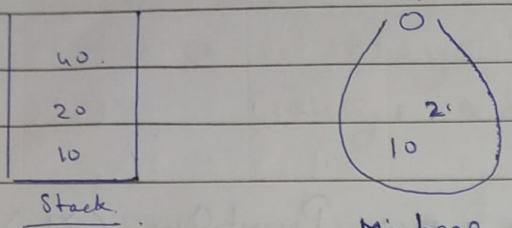
Heaps.

→ Data Structure
→ Priority Queue

M T W T F S S
Page No.:
Date:
YOUVA

→ What & Why ?



add(2)
add(10)
add(1)
remove()
add(0)

push() → O(1) (Add)
pop() → O(1) (Remove)
peek() → O(1) (Get).

In Stack.

Heap => Add :- The ele is added to heap & the minimum ele is at top T.C → O(log n) // n => ele present in heap At that Point.

Remove :- The top(min^m) ele is removed & the next min^m ele comes on top T.C → O(log n) //

peek :- returns the top(min^m) element T.C → O(1) //

Size :- returns size.

Types of Heaps :- → maxHeap. → minHeap.

(1) MinHeap :- Java Collection framework (Builtin) (Default).

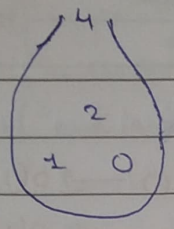
If we are adding 'n' ele one by one in a heap then →

No. of Ops ≈ log(1) + log(2) + log(3) + ... log(n) = log(n!).
≈ n log n.
⇓
T.C = O(n log n) //

=> minHeam V/s ArrayList. → Vala heap isme Add me. TC = O(n log n).

add(2) → arr = {2},
add(10) → arr = {2, 10} → {10, 2},
add(1) → arr = {10, 2, 1} → {10, 2, 1},
remove() → arr = {10, 2},
add(0) → arr = {10, 2, 0} → {10, 2, 0},
add(20). → arr = {10, 2, 0, 20} → {20, 10, 2, 0} //

② Max Heap



10

add (2)
add (10)
add (1).
remove ( ).
add (0)
add (4).

M	T	W	T	F	S	S
Page No.:						YOUVA
Date:						

=> STL :-

Min Heap => PriorityQueue<Integer> pq = new PriorityQueue<>();

Max Heap =>

PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());

* Problem Identification.

① k^{th} Smallest / largest / closest / frequent / k-Sorted Array Out of n, if we are working on 'k' elements.

② $O(n \log n) < O(n \log k) < O(n)$

③ Minimise, Maximise, Continuous Sorting

④ For k smallest ⟶ we use Max Heap.

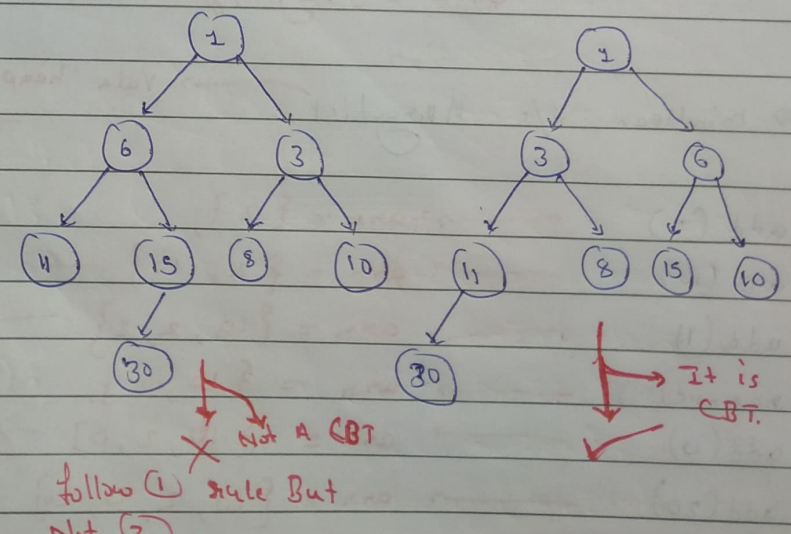
Heaps Visualisation ⟶ binary tree

arr = { 10, 1, 3, 8, 11, 30, 15, 6 },

Min heap :-

↳ ① Ek esa Binary Tree jisme koi bhi node apne Children se chhoti value rakhta hai

↳ ② CBT. [Complete Binary Tree]



⇒ CBT [Complete Binary Tree].

↳ where 'n-1' levels are completely filled & the last level may or may not be completely filled, but it should be in continous manner from left to right.

↳ It is Always Balanced so height of CBT is always 'log n'.

⇒ Rough Implementation :- [Min Heap]

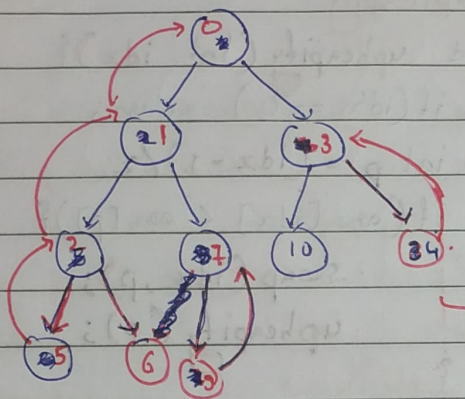
arr = { 1, 2, 4, 5, 9, 10 }

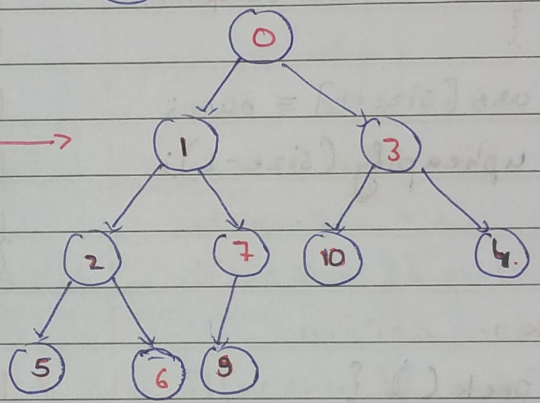
Add (3)

Add (0)

Add (6)

Add (7)





Addition / Insertion.

(1) Add the ele At last
↳ O(1)

(2) Upheapify
↳ O(log n).

Implement A Min Heap by Array :-

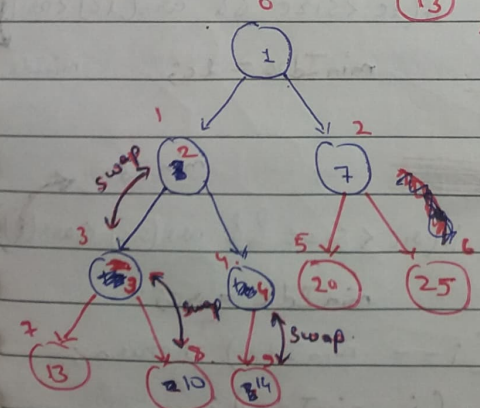
↳ Heap is implemented by Array & Visualised as a CBT with Heap Order Property (HOP).

(1) Addition:-

arr = { 1, 2, 7, 23, 24, 20, 25, 13, 10, 14 }

(1) Add the ele At last
(2) Upheapify.

add (20).
↳ 20 > 7
↳ Nothing will Change.



#lc = 2p + 1.

#rc = 2p + 2.

$\# p = \frac{c-1}{2}$.

add (25).
↳ 25 > 7
↳ Nothing will Change.

add (13).

add (4)
↳ 4 < 14.
↳ upheapify.

add (2).
↳ 2 < 10
↳ upheapify
→ 2 < 3

Code:-

MinHeap

```
Class MinHeap {
    private int[] arr;
    private int size;

    MinHeap (int capacity) {
        this.arr = new int[capacity];
        size = 0;
    }
```

// Add:-

```
void add (int num) {
    if (size == arr.length) {
        (Heap full)
        throw exception;
    }
    arr[size++] = num;
    upheapify (size-1);
}
```

// peek :-

```
int peek () {
    if (size == 0) {
        (Heap empty)
        throw Exception;
    }
    return arr[0];
}
```

// Remove :-

```
int remove () {
    if (size == 0)
        (Heap empty)
        throw Exception;
    int peek = arr[0];
    swap (0, size-1);
    size--;
    downHeapify (0);
    return peek;
}
```

// size.

```
int size() {
    return size;
}
```

// swap :-

```
Void swap (int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
```

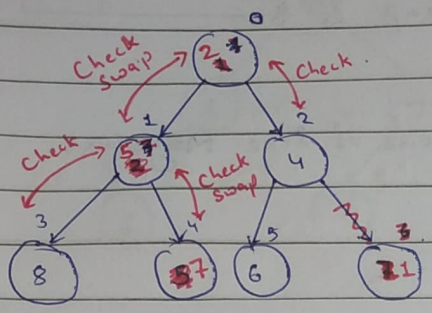
// UpHeapify :

```
Void upheapify (int idx) {
    if (idx == 0) return;
    int p = (idx-1)/2
    if (arr[idx] < arr[p]) {
        swap (idx, p);
        upheapify (p);
    }
}
```

// Down Heapify :-

```
void downHeapify (int i) {
    if (i >= size-1) return;
    int lc = 2*i + 1;
    int rc = 2*i + 2;
    int minIdx = i;
    if (lc < size && arr[lc] < arr[i])
        minIdx = lc;
    if (rc < size && arr[rc] < arr[minIdx])
        minIdx = rc;
    if (i == minIdx) return;
    swap (i, minIdx);
    downHeapify (minIdx);
}
```

(2) Remove in MinHeap:



pq.remove().

1. Swap arr[0] & arr[size-1]
2. Size --
3. DownHeapify

#Code → MAX HEAP. → Changes are only in UpHeapify & Downheapify Methods.

```
Void upheapify(int idx){
    if(idx==0)
        return;
    int p = (idx-1)/2;
    if(arr[idx] > arr[p]){
        swap(idx, p);
        upheapify(p);
    }
}
```

```
Void downHeapify(int i){
    if(i >= size-1)
        return;
    int lc = 2*i+1;
    int rc = 2*i+2;
    int minIdx = i;
    if(lc < size && arr[lc] > arr[minIdx])
        minIdx = lc;
    if(rc < size && arr[rc] > arr[minIdx])
        minIdx = rc;
    if(i == minIdx)
        return;
    swap(i, minIdx);
    downHeapify(minIdx);
}
```

Heapify Alogorithm.

→ upheapify & downheapify.

upheapify → Add

downheapify → remove

Note :-

→ A sorted array is Always a minheap. Vice-versa is Not True.

→ A sorted array in decreasing Order is Always A Max Heap.

Heap Sort → Mazak.

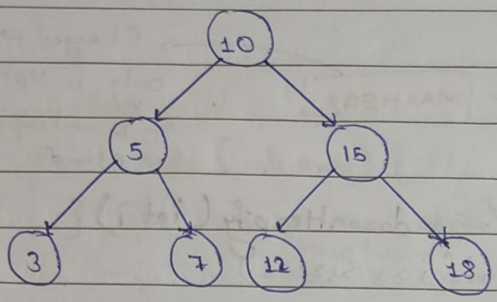
→ Adding the 'n' elements to heap & then remove them One by One.

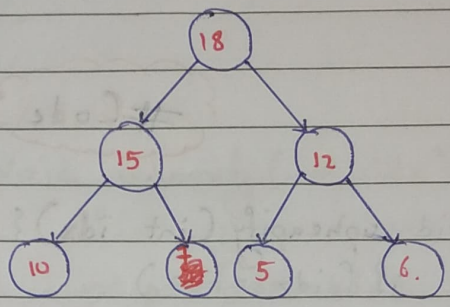
→ T.C = O(n log n).

S.C = O(n).

} → kind of like MergeSort.

Convert BST to MaxHeap :-



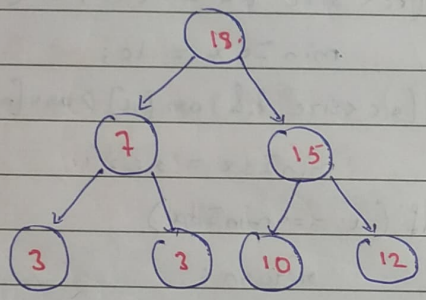


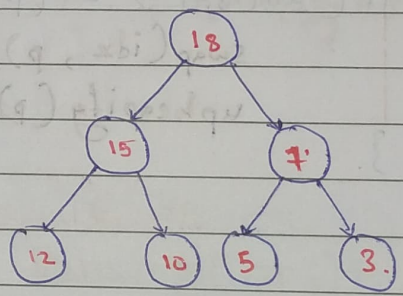
Reverse Inorder → {18, 15, 12, 10, 7, 5, 3}.

Inorder → {3, 5, 7, 10, 12, 15, 18.}

Remove ele's from Back.

Inorder To Level Order Traversal.

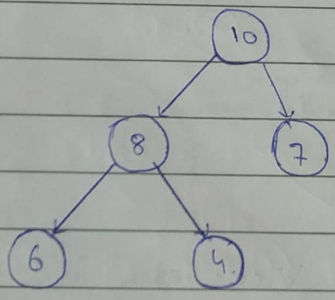




Inorder To Preorder Traversal.

Reverse Inorder To Post Order Traversal.

Check if Given Binary Tree is a MaxHeap or Not :-

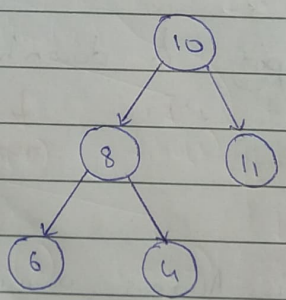


True

HOP ✓ CBT ✓

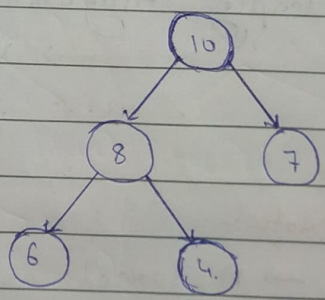
↓

P > C



False

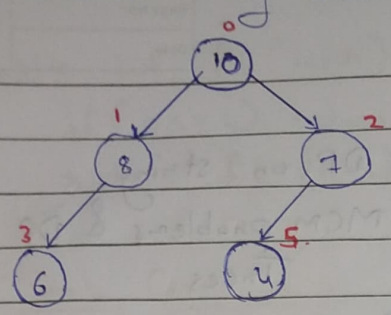
HOP ✗ CBT ✓



False

HOP ✓ CBT ✗.

How to Check if any tree is CBT or Not :-



Size ⇒ 5

lc = 2i + 1.

rc = 2i + 2.

if (lc >= size) ⟶ return false.

if (rc >= size) ⟶ return false.

=> Code :-

```
ps boolean isMaxHeap(Node root){
    int size = size(root);
    return isHeap(root) && isCBT(root, 0, size);
}

ps int size(Node root){
    if(root == null) return 0;
    return 1 + size(root.left) + size(root.right);
}

ps boolean isHeap(Node root){
    if(root == null) return true;
    if(root.left != null && root.left.val > root.val) return false;
    if(root.right != null && root.right.val > root.val) return false;
    return isHeap(root.left) && isHeap(root.right);
}

ps boolean isCBT(Node root, int i, Integer n){
    if(root == null) return true;
    if(i >= n) return false;
    return isCBT(root.left, 2*i+1, n) && isCBT(root.right, 2*i+2, n);
}
```

int, double, char, String ⟶ pass by Value.

↓ ↓ ↓

Integer, Double, Character. ⟹ Pass by Reference. [Wrapper Class]