PW SKILLS

Heaps

Lecture 2

Today's checklist

1. Heaps Visualisation (MaxHeap and MinHeap)
2. Implementation of MinHeap by Array
3. Heapify Algorithm
4. Heap Sort
5. Questions on heaps

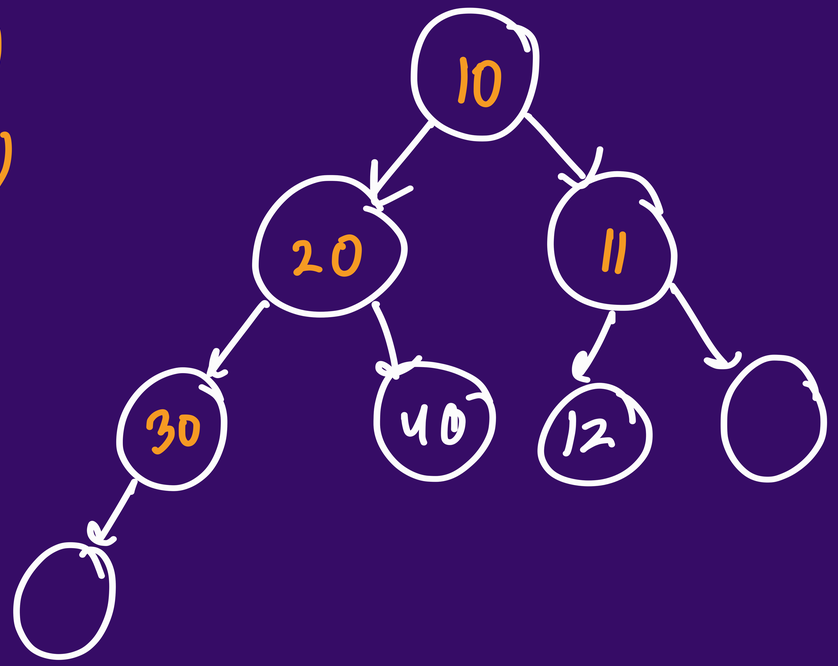
Heaps Visualisation (Binary tree)

PW SKILLS

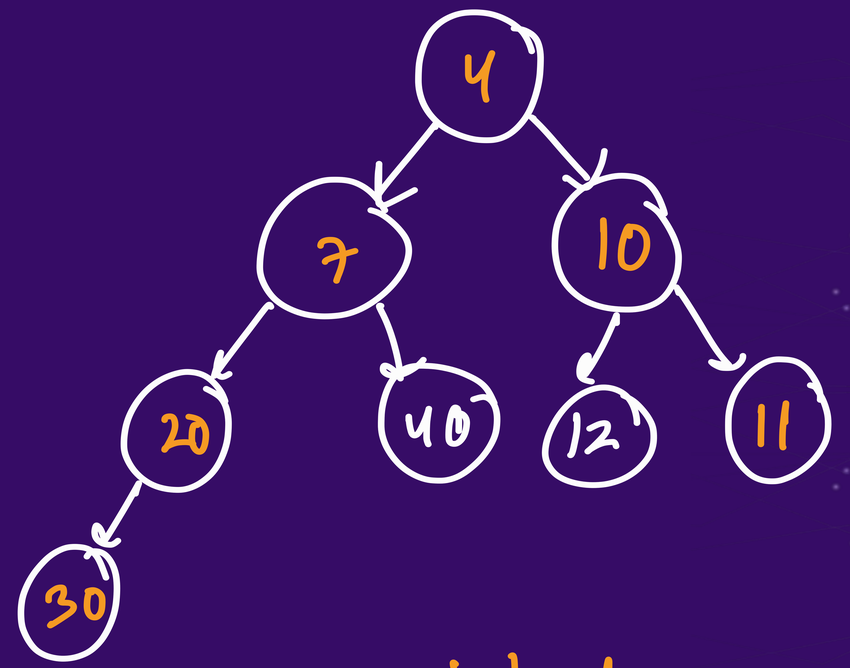
For Ex: Lets do minheap

push(10)
push(20)
push(11)
push(30)
push(40)
push(12)
push(4)
push(7)

pop()
pop()



minheap

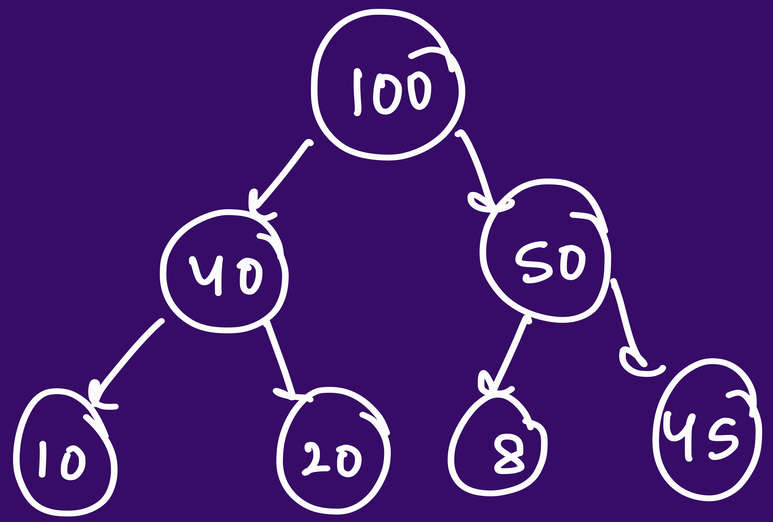


minheap

Heaps Visualisation

PW SKILLS

Maxheap



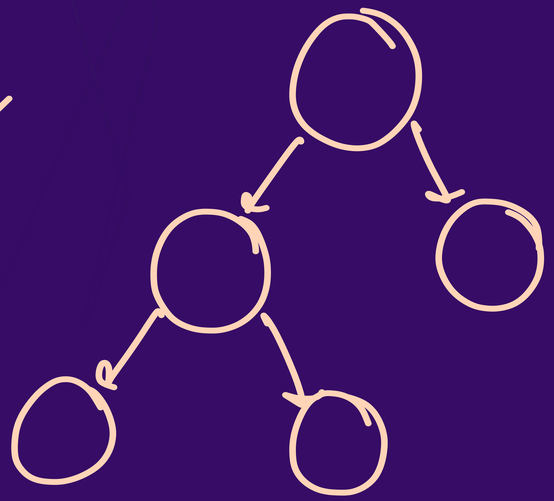
H.W. Visualise & draw maxheap via a CBT.

Heaps Visualisation

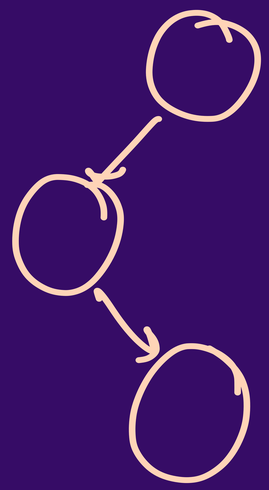
PW SKILLS

What is a complete Binary Tree ?

CBT ✓

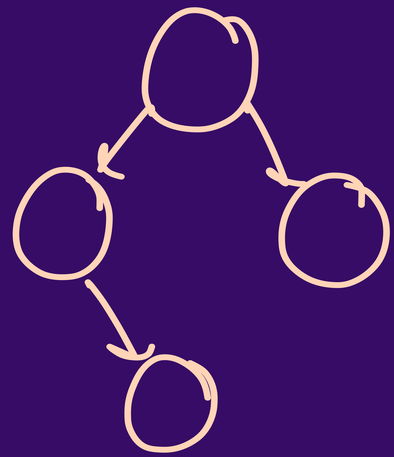


levels = 2

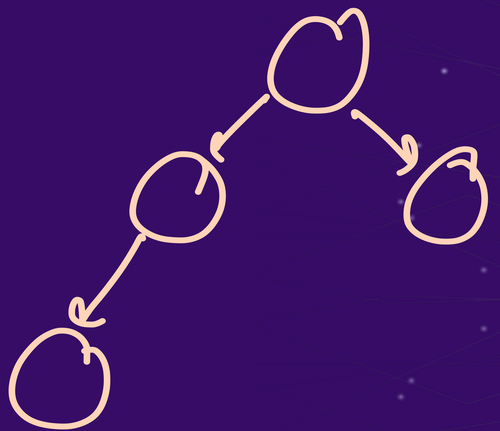


levels = 3

✗



✗



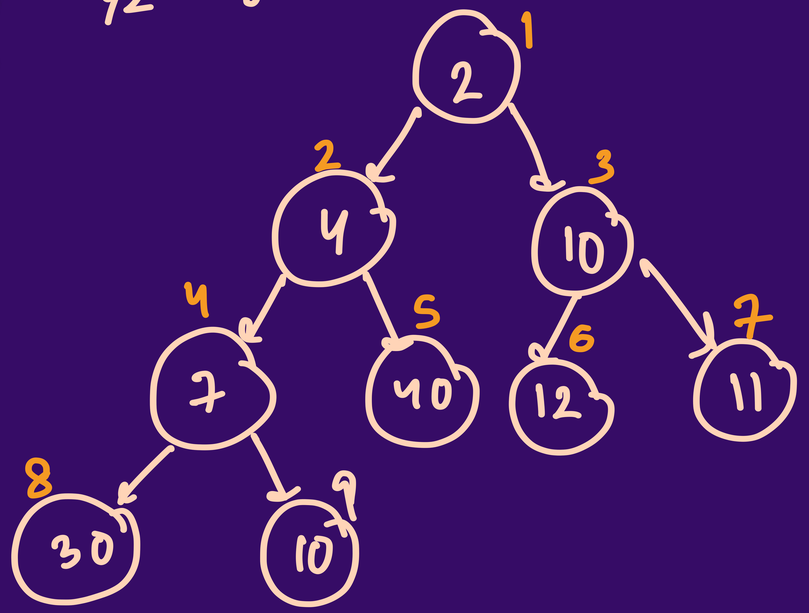
✓

Ques:

Q1 : Implement a MinHeap by Array (Visualise it with a CBT)

push(10) ✓
push(20) ✓
push(11) ✓
push(30) ✓
push(40) ✓
push(12) ✓
push(4) ✓
push(7) ✓
push(2) ✓

0	1	2	3	4	5	6	7	8	9	10	11	12	...
⊠	2	4	10	7	40	12	11	30	20				
	i/2	i											

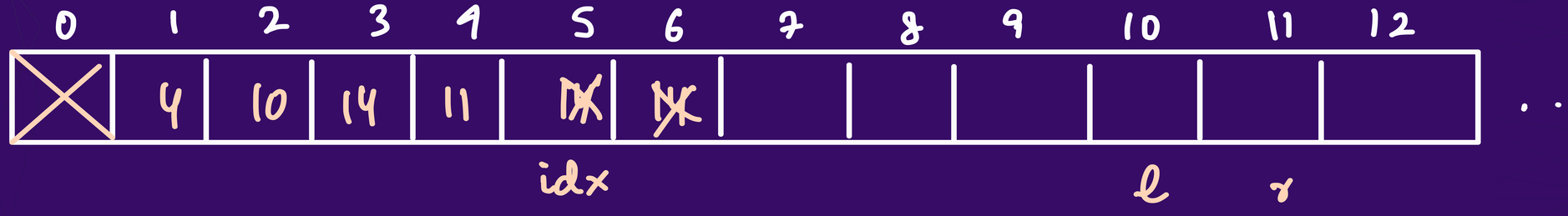


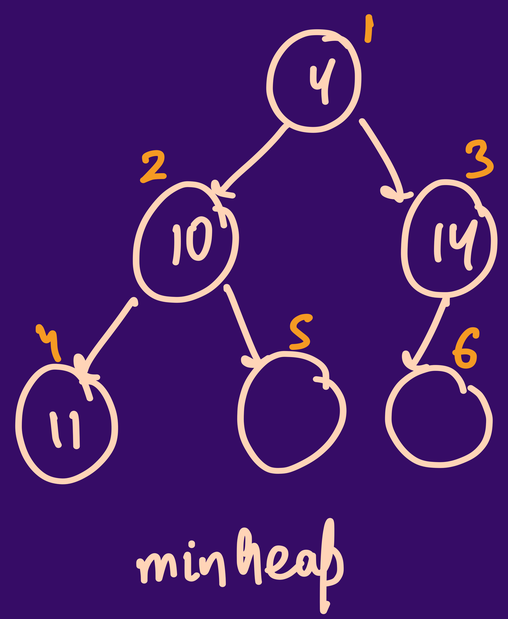
for a node at i,
then,
left child = 2*i
right child = 2*i + 1
parent = i/2

Ques:

Q1 : Implement a MinHeap by Array

push(10)
push(2)
push(14)
push(11)
push(1)
push(4)
pop()
pop()





minheap

Homework :

Implement a MaxHeap using Array

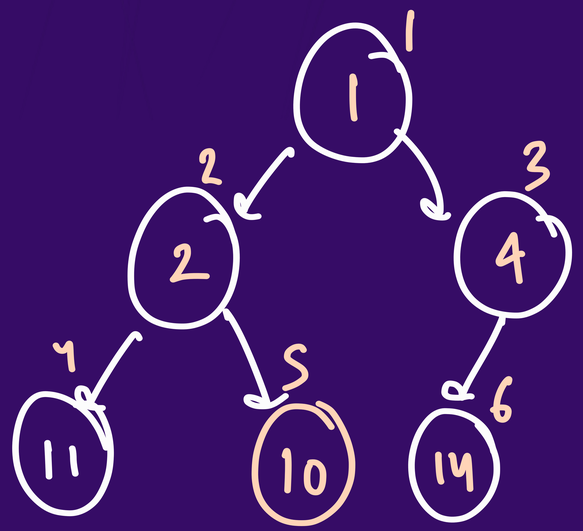
Heapify Algorithm

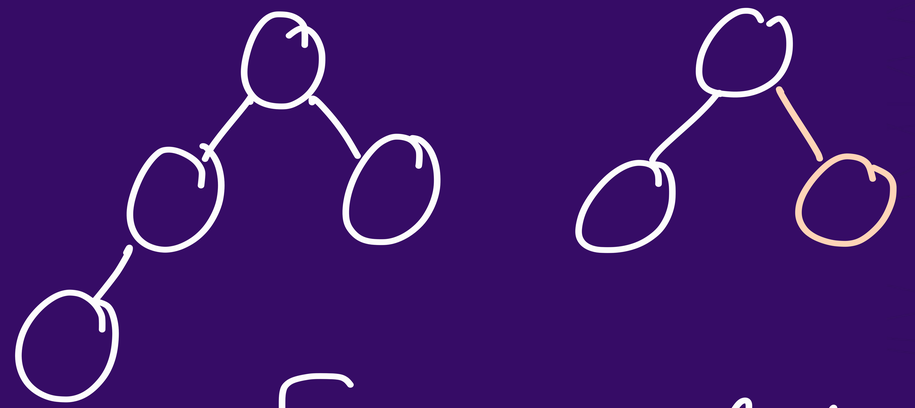
PW SKILLS

↳ Convert given array to Heap

	1	2	3	4	5	6
arr =	1	2	4	11	10	14

Convert it into minheap





$\left[\frac{n}{2}, \frac{n}{2}+1 \text{ leaf nodes}\right]$

even odd

Heapify Algorithm

PW SKILLS

```
for (int i = n/2 ; i>=1 ; i--){
    heapify (i, arr, n);
}
```

3

↓

pop()'s rearrangement

Heapify Algorithm

```
void heapify(int i, int arr[], int n){
    while (true){
        int left = 2 * i, right = 2 * i + 1;
        if (left >= n) break;
        if (right >= n){
            if (arr[i] > arr[left]){
                swap(arr[i], arr[left]);
                i = left;
            }
            break;
        }
        if (arr[left] < arr[right]){
            if (arr[i] > arr[left]){
                swap(arr[i], arr[left]);
                i = left;
            }
            else break;
        } else{
            if (arr[i] > arr[right]){
                swap(arr[i], arr[right]);
                i = right;
            }
            else break;
        }
    }
}
```

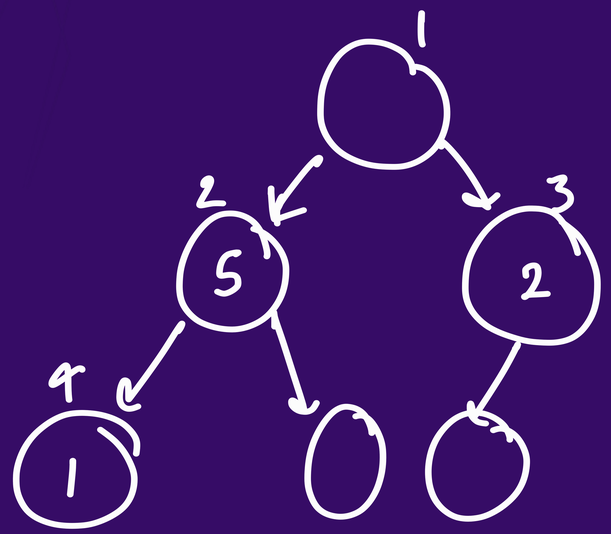
Heap Sort (A joke) (use pq STL)

PW SKILLS

For Ex: an array is given, sort it using heap.

arr = { 10, 1, 2, 20, 5, 8 }

8 10 20



Can be done by minheap & maxheap

T.C. = $O(n * \log n)$

↓

Merge & Quick Sort

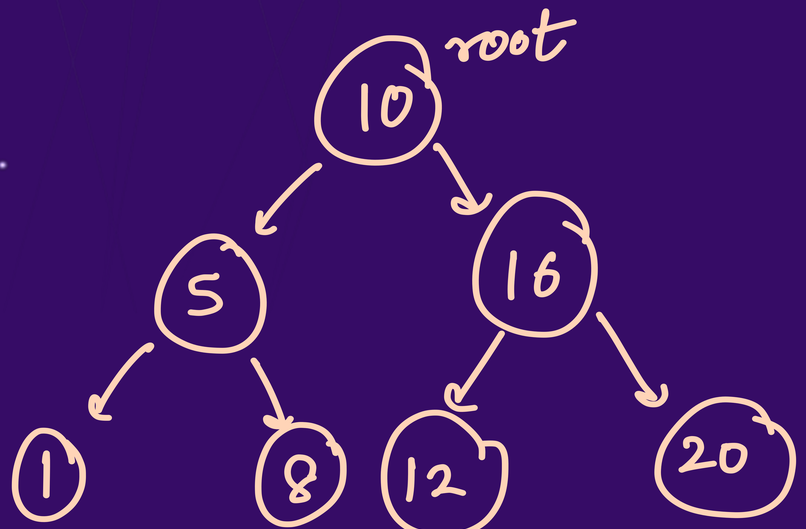
S.C. = $O(n)$

Ques:

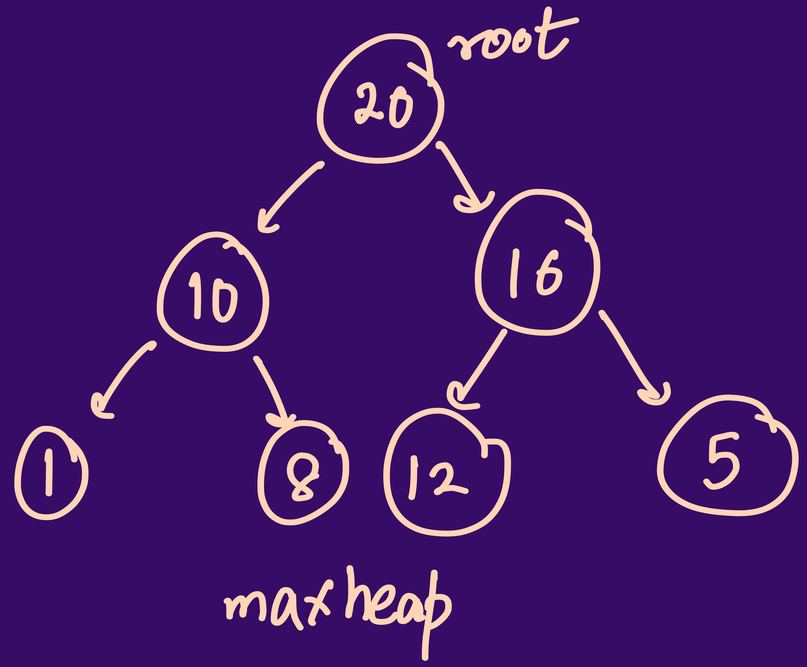
Q2 : Convert BST to MaxHeap

inorder

BST → sorted array



1 5 8 10 12 16 20



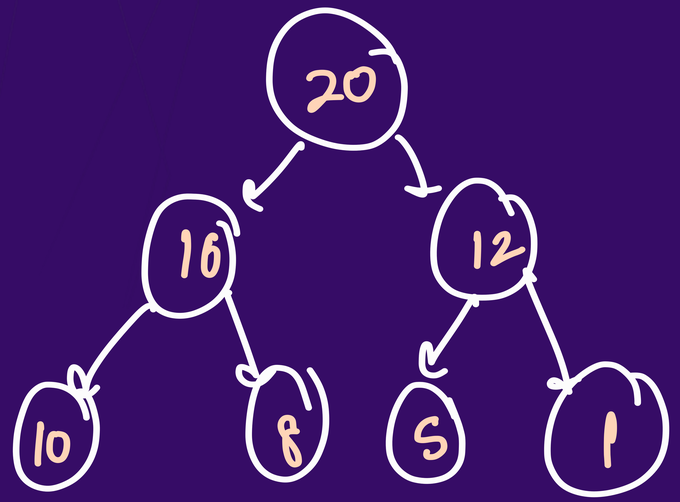
max heap

Ques:

ek LST > RST → O(N-2)

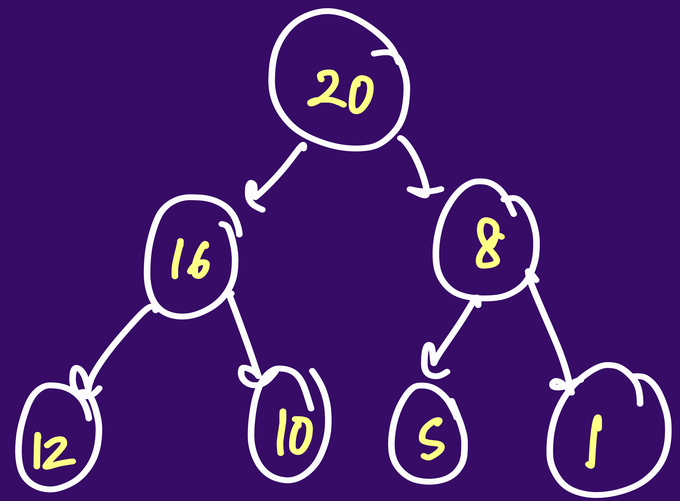
Q2 : Convert BST to MaxHeap

20 16 12 10 8 5 1



1) level wise → ek array ke elements (1 sorted)

20 16 12 10 8 5 1



2) Pre Order wise ✓

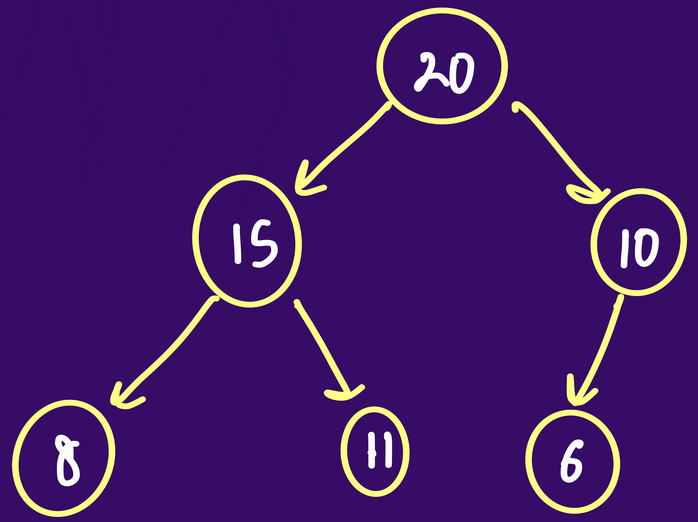
Ques:

PW SKILLS

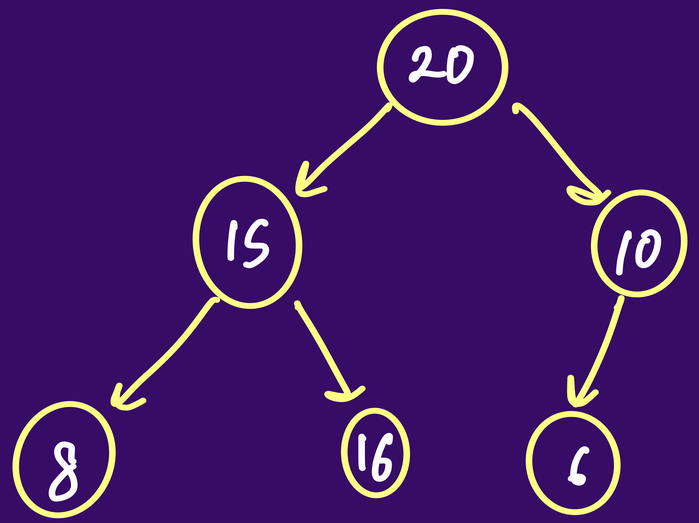
Q3 : Check if given Binary Tree is a MaxHeap or not

Condition-1 : all decendants of any node should be smaller

⇒ root->left->val < root->val > root->right->val



✓



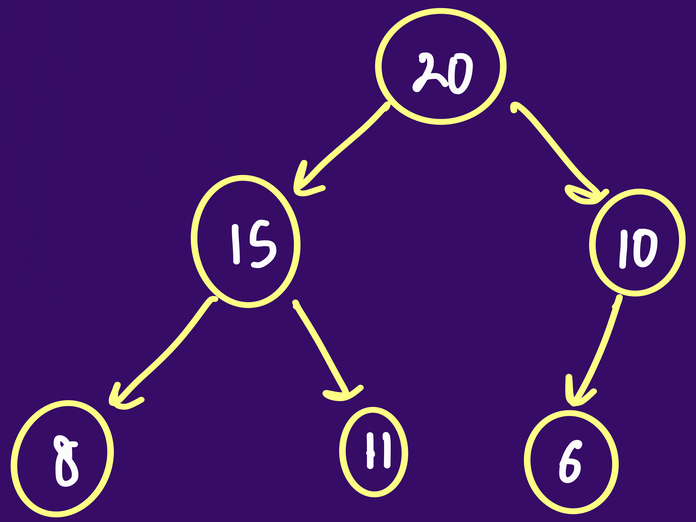
✗

Ques:

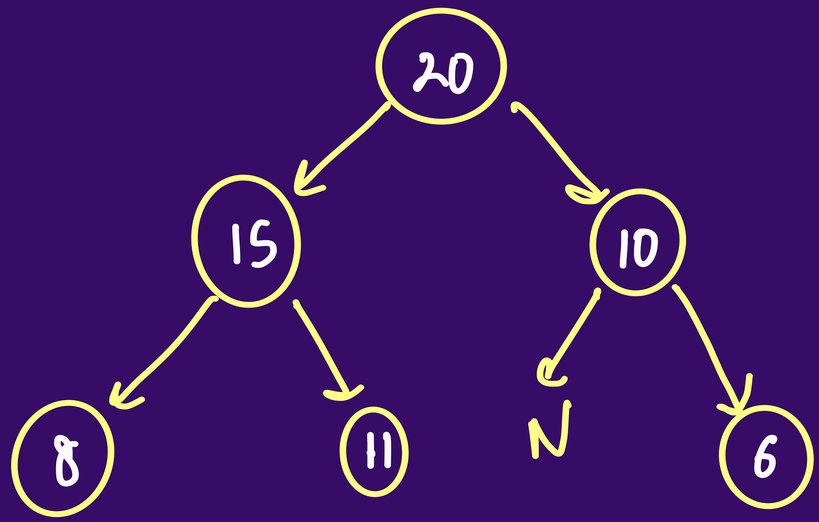
PW SKILLS

Q3 : Check if given Binary Tree is a MaxHeap or not

Condition-2 : It should be a CBT.



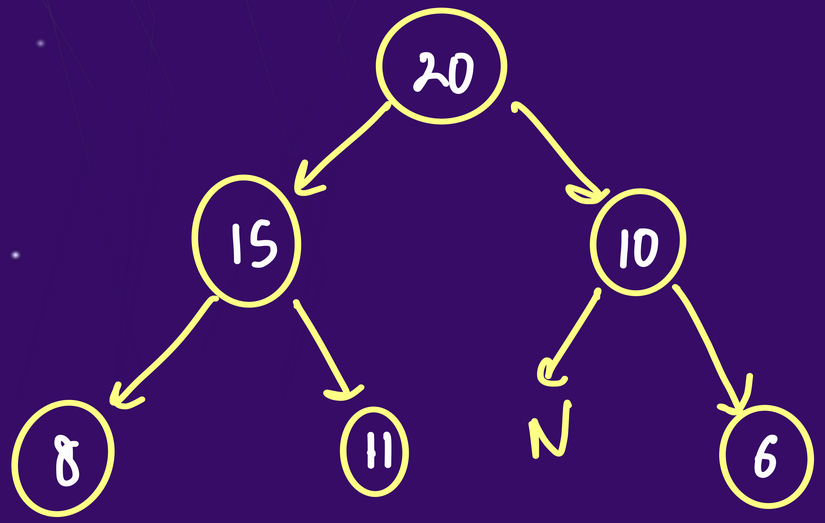
✓



✗

Ques:

Q3 : Check if given Binary Tree is a MaxHeap or not



Size = 6

G N N N N

Q

Count = 0 1 2 3 4 5 6

20 15 10 8 N 11

Ques:

PW SKILLS

Q3 : Check if given Binary Tree is a MaxHeap or not

```
bool isCBT(Node* root){
|
}
```

```
bool isMax(Node* root){
|
}
```

if(isCBT(root) && isMax(root)) → Yes

else → No

Ques:

```cpp
bool isCBT(Node* root){
    int size = sizeOfTree(root);
    int count = 0;
    queue<Node*> q;
    q.push(root);
    while(count<size){
        Node* temp = q.front();
        q.pop();
        count++;
        if(temp!=NULL){
            q.push(temp->left);
            q.push(temp->right);
        }
    }
    if(q.size()>0){
        Node* temp = q.front();
        if(temp!=NULL) return false;
        q.pop();
    }
    return true;
}
```

```cpp
bool isMax(Node* root){
    if(root==NULL) return true;
    if(root->left!=NULL && root->val<root->left->val) return false;
    if(root->right!=NULL && root->val<root->right->val) return false;
    return isMax(root->left) && isMax(root->right);
}
```

THANK YOU