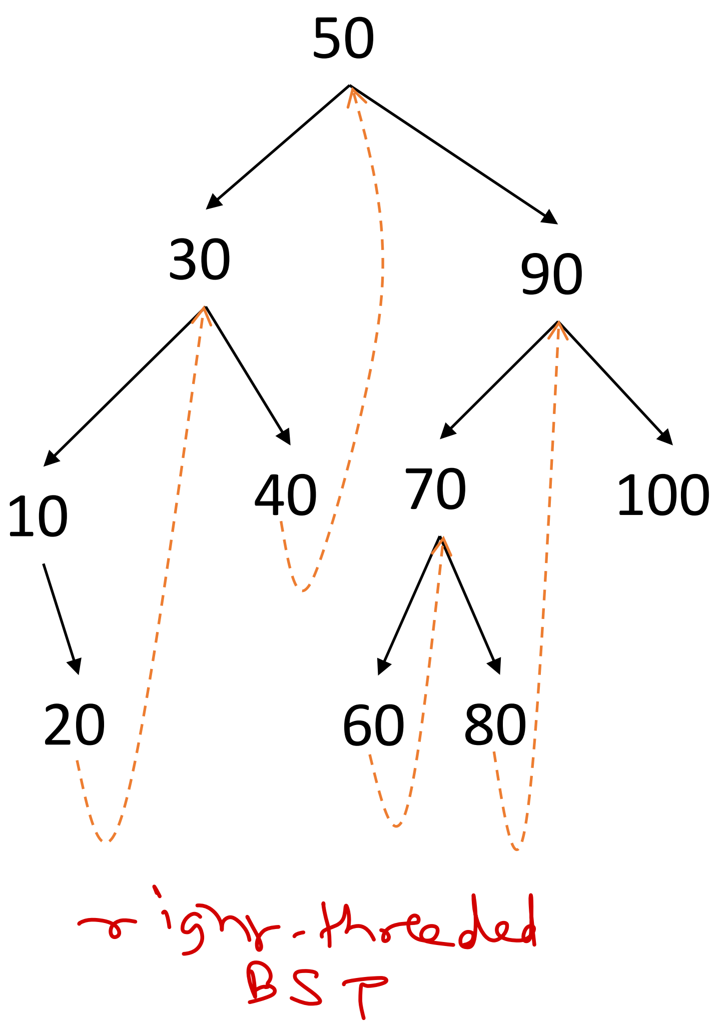
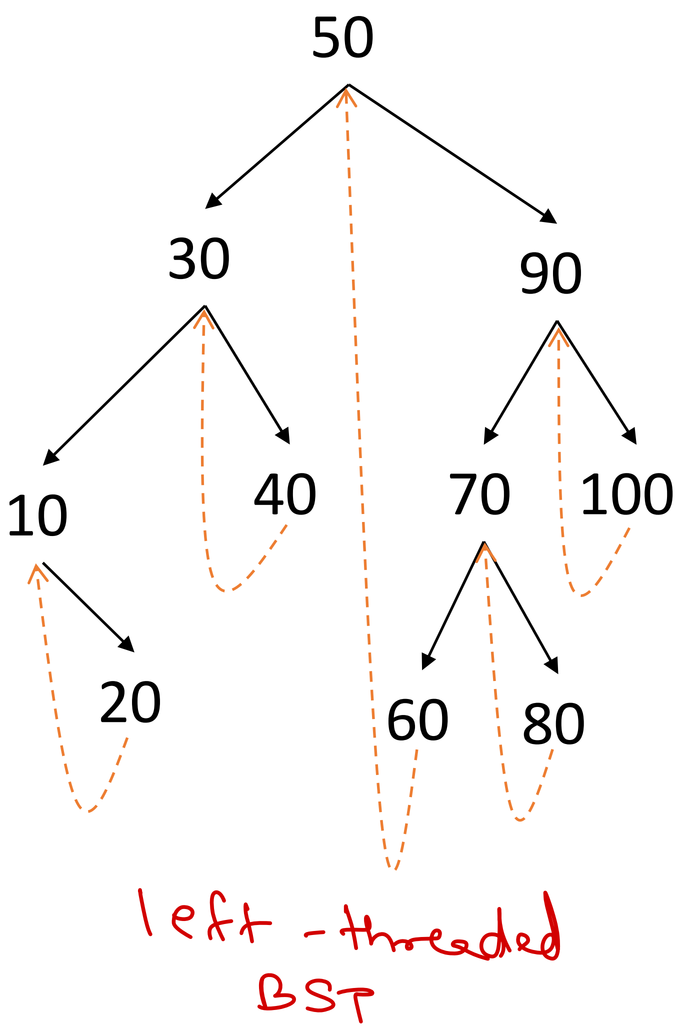
Data Structure & Algorithms

Nilesh Ghule

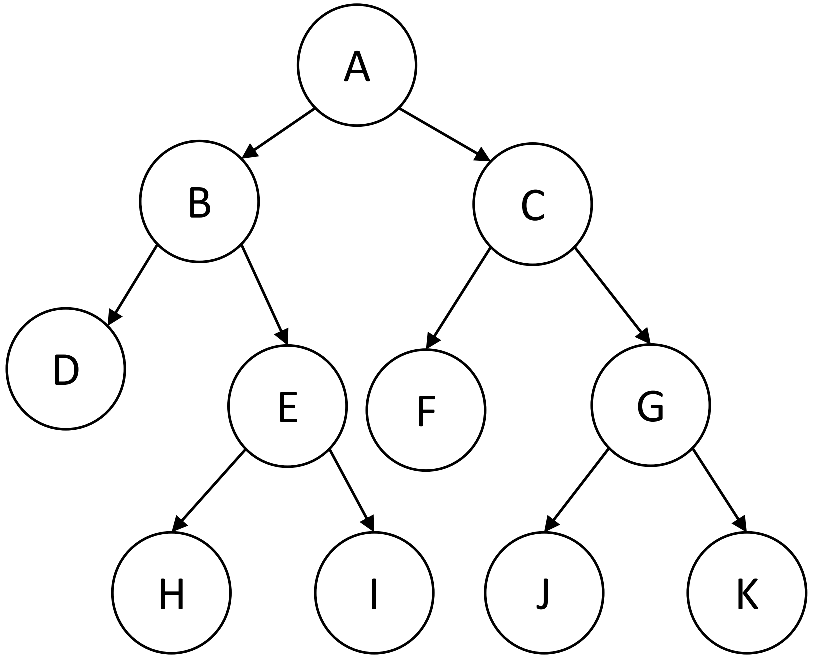
Threaded BST - fast inorder traversal.





- Typical BST in-order traversal involves recursion or stack. It slows execution and also need more space.
- Threaded BST keep address of in-order successor or predecessor addresses instead of NULL to speed up in-order traversal (using a loop).
- Left threaded BST +
- Right threaded BST
- In-threaded BST =

Strict/Full Binary Tree



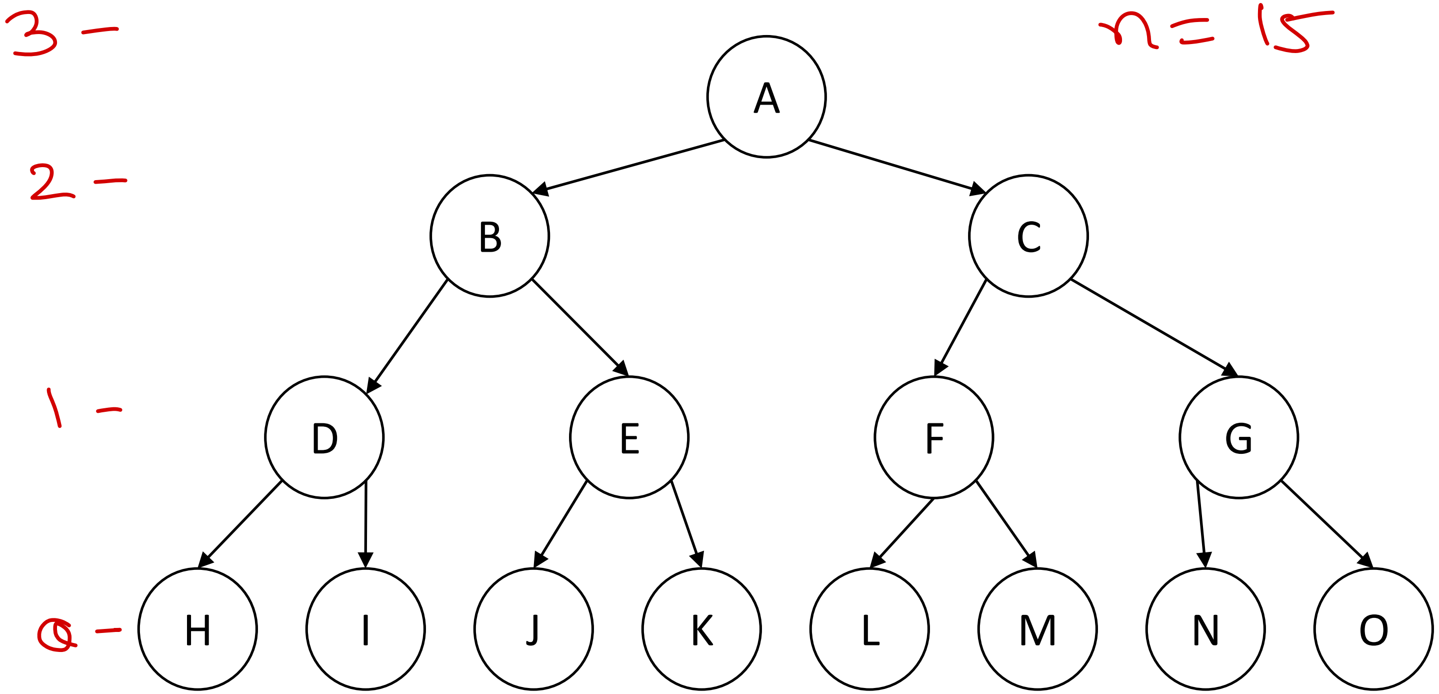
- Binary tree in which each non-leaf node has exactly two child nodes.

single child node is not allowed.
either 0 or 2 child nodes.

Perfect Binary Tree

null tree height = -1

n = 15



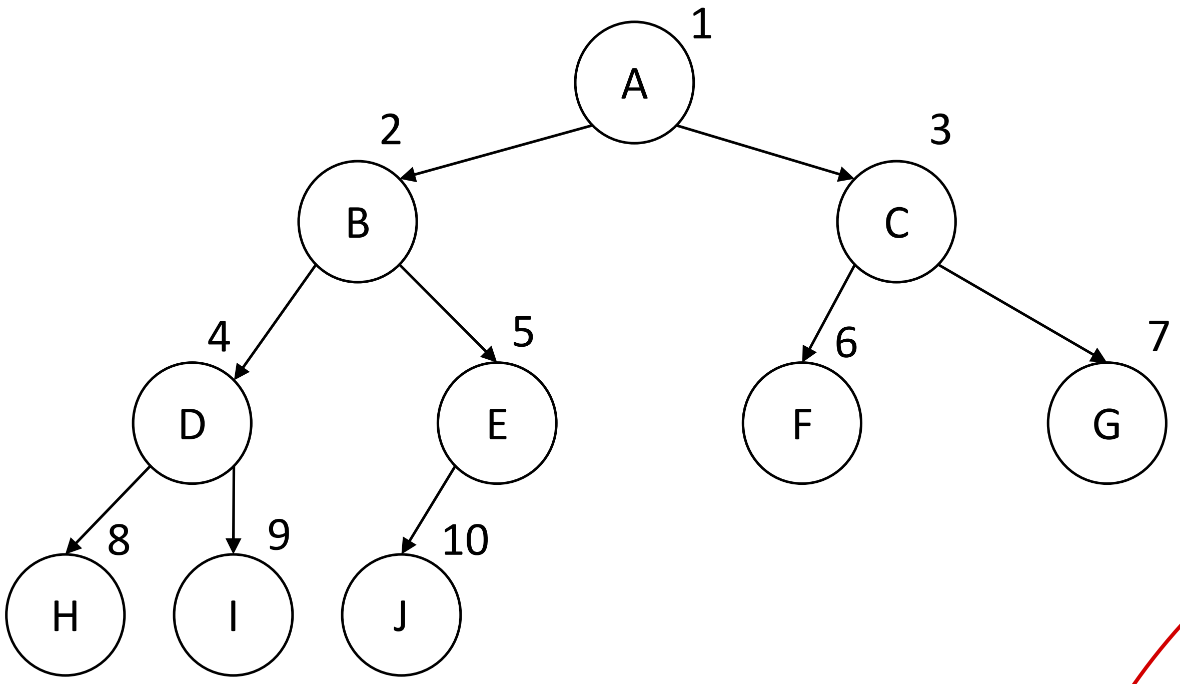
- Binary tree which is full for the given height i.e. contains maximum possible nodes.

Number of Leaf = 2^h
Number of Non-Leaf = $2^h - 1$

- Number of nodes = ~~$2^h - 1$~~ $2^{h+1} - 1$

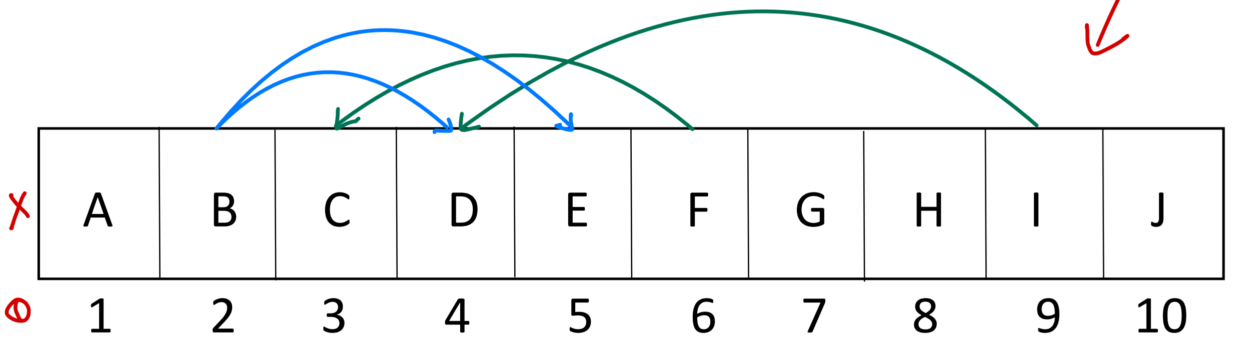
Complete Binary Tree and Heap

tree : hierarchial : parent – child



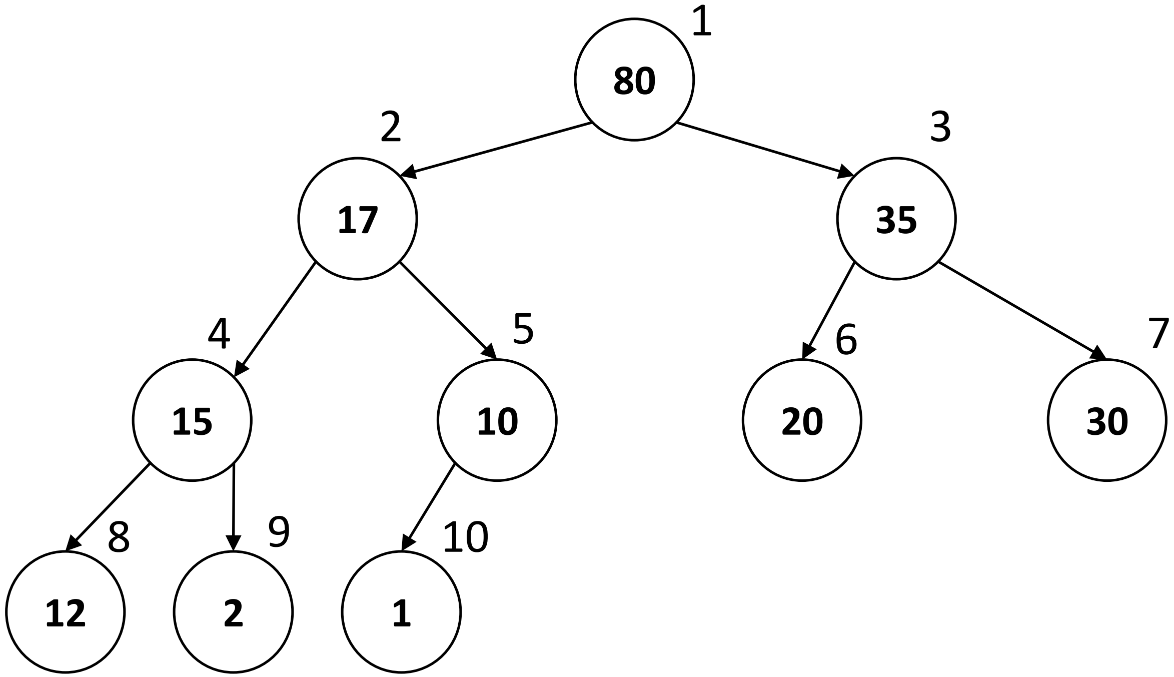
- A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible. → in last level .

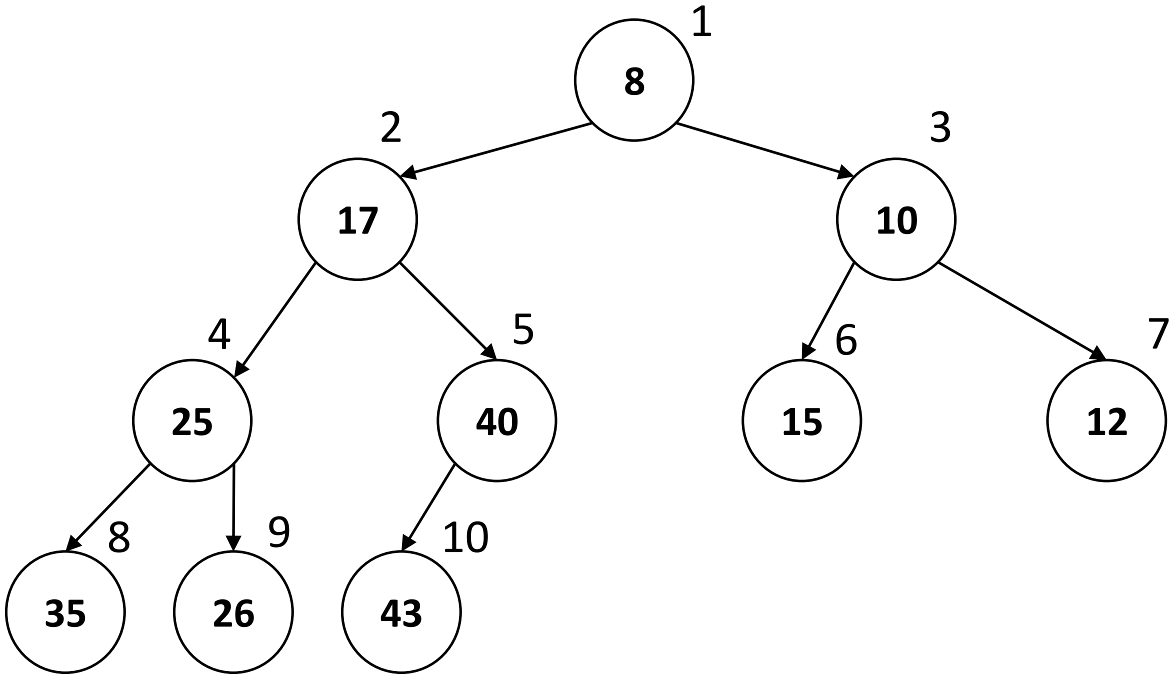
- Heap is array implementation of complete binary tree.
- Parent child relation is maintained through index calculations
  - parent index = child index / 2
  - left child index = parent index * 2
  - right child index = parent index * 2 + 1



A	B	C	D	E	F	G	H	I	J
1	2	3	4	5	6	7	8	9	10

Max Heap & Min Heap

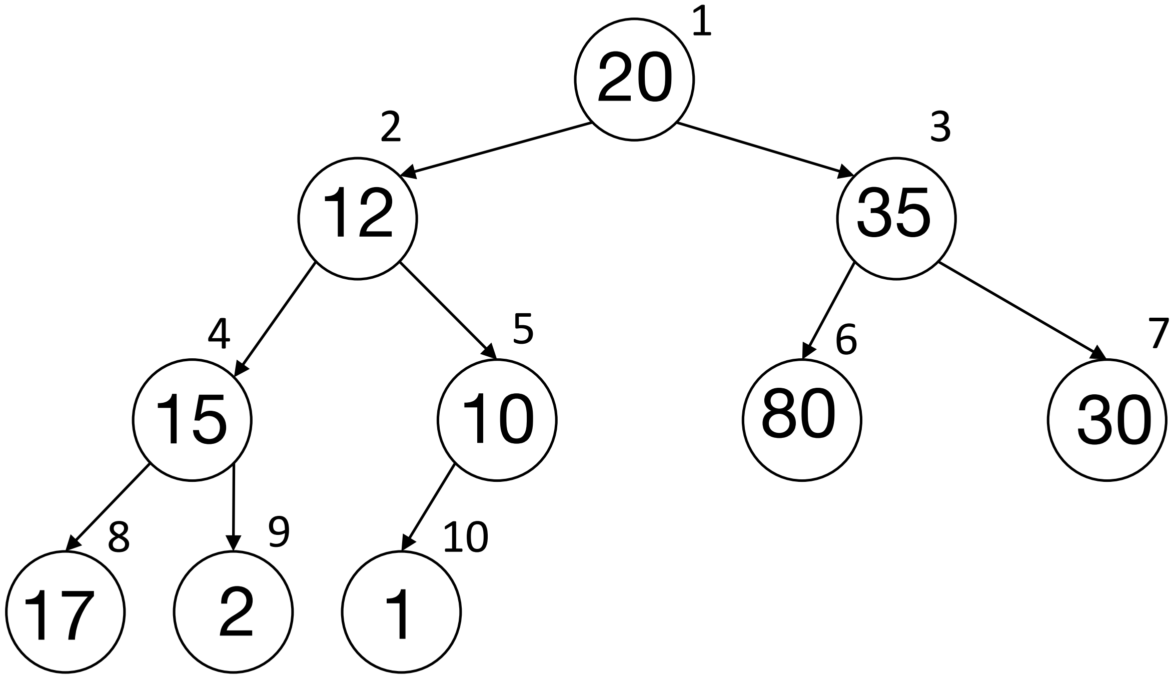




- Max heap is a heap data structure in which each node is greater than both of its child nodes.

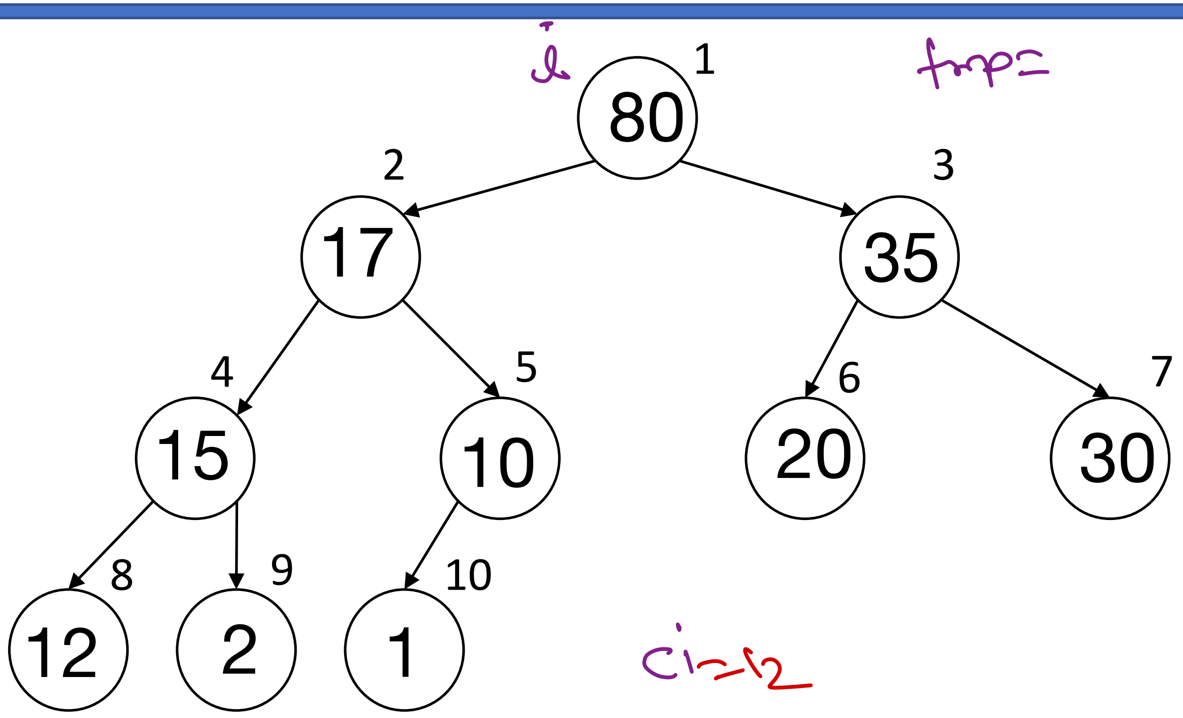
- Min ~~Max~~ heap is a heap data structure in which each node is smaller than both of its child nodes.

Max Heap – Initialize



20	12	35	15	10	80	30	17	2	1
1	2	3	4	5	6	7	8	9	10

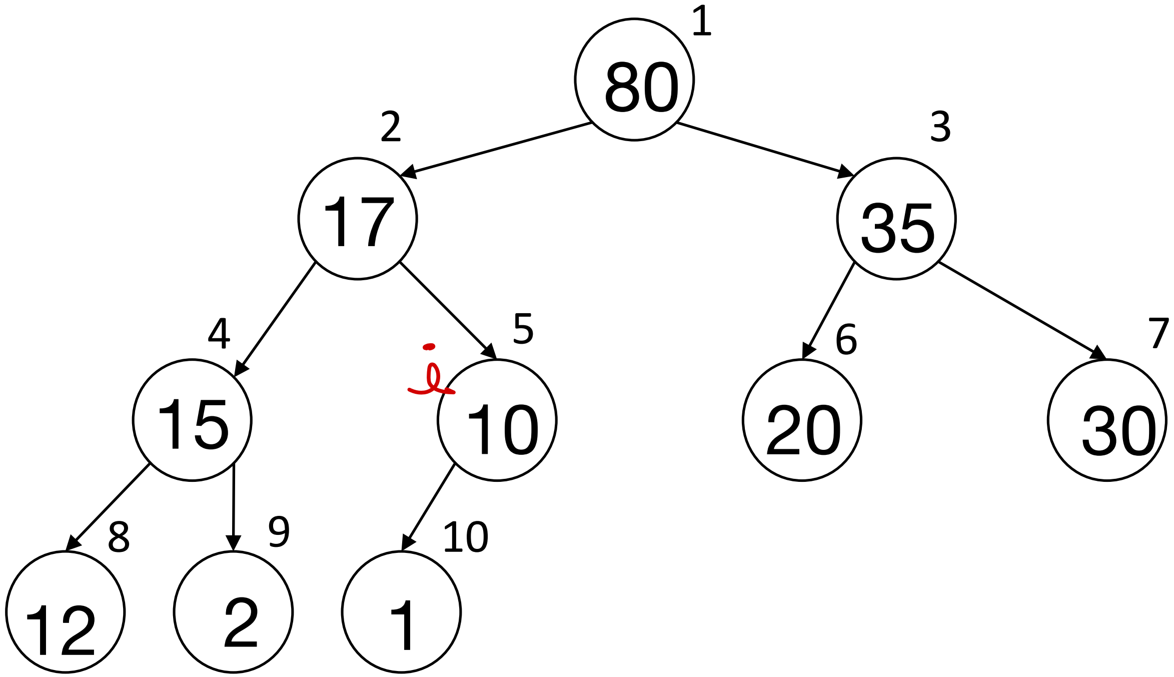
Max Heap – Initialize



1	2	3	4	5	6	7	8	9	10

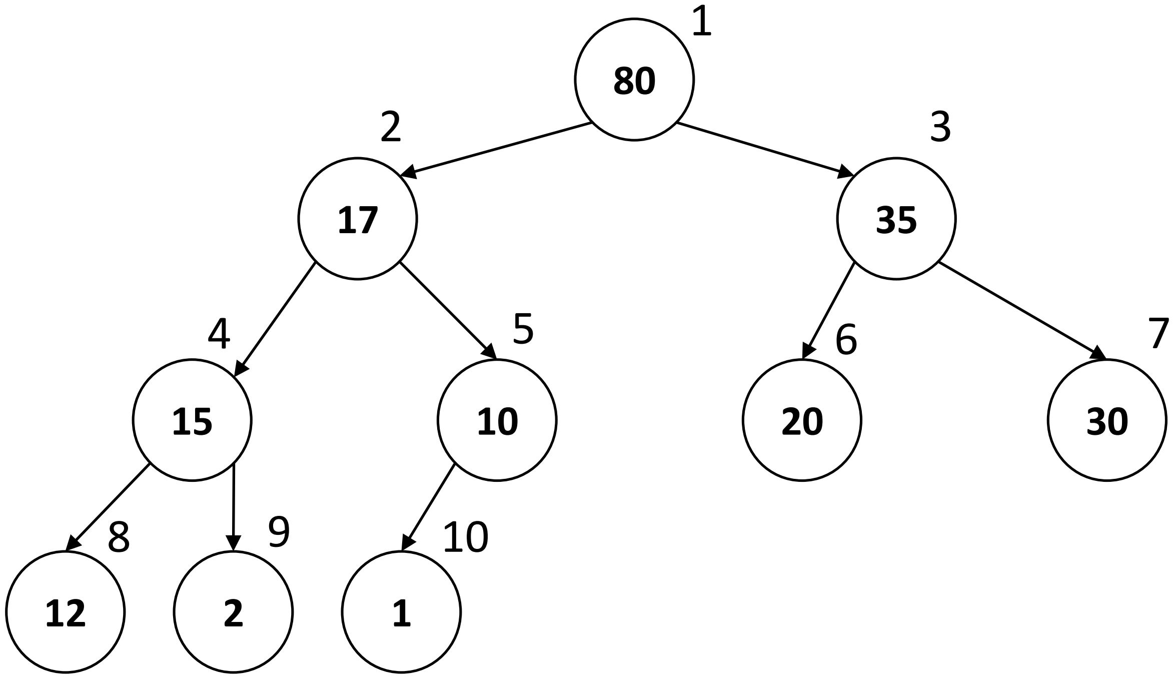
```
size = arr.length -1; → 10
for(i=size/2; i>=1; i--) {
    temp = arr[i];
    ci = i*2;
    while(ci <= size) {
        if(arr[ci+1] > arr[ci])
            ci++;
        if (temp > arr[ci])
            break;
        arr[ci/2] = arr[ci];
        ci = ci*2;
    }
    arr[ci/2] = temp;
}
```

Max Heap – Initialize



80	17	35	15	10	20	30	12	2	1
1	2	3	4	5	6	7	8	9	10

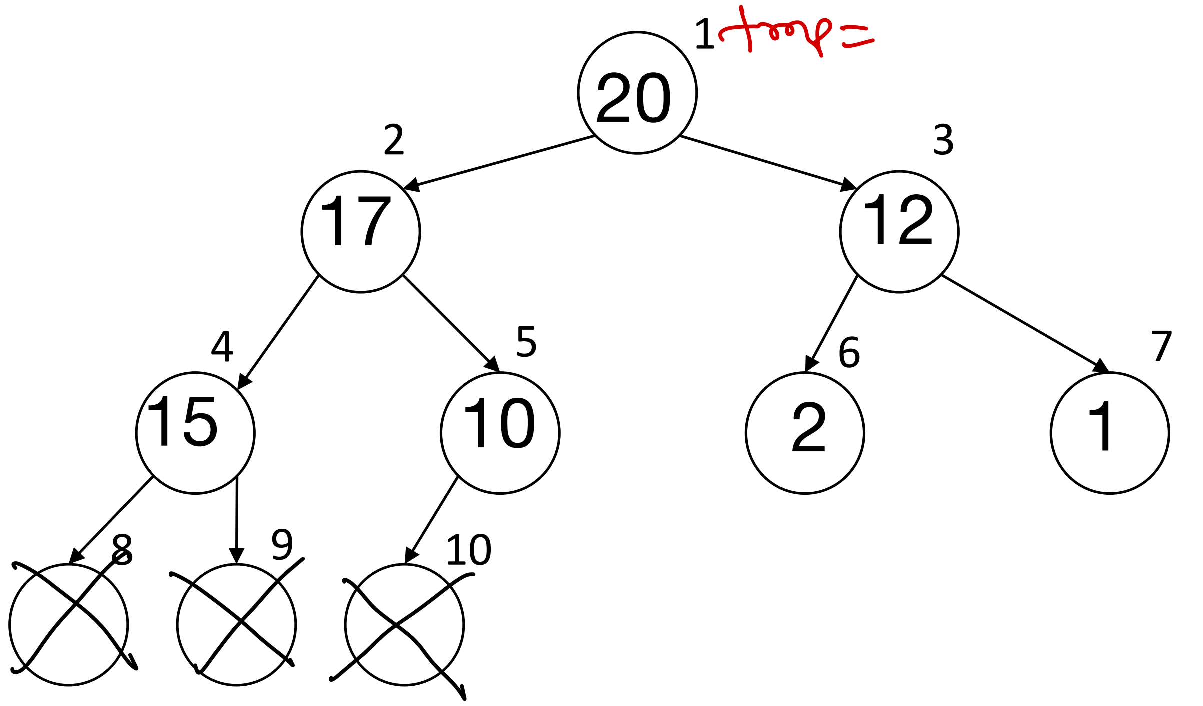
Max Heap



80	17	35	15	10	20	30	12	2	1
1	2	3	4	5	6	7	8	9	10

Max Heap – Delete Element

80 35 30



1	2	3	4	5	6	7	8	9	10

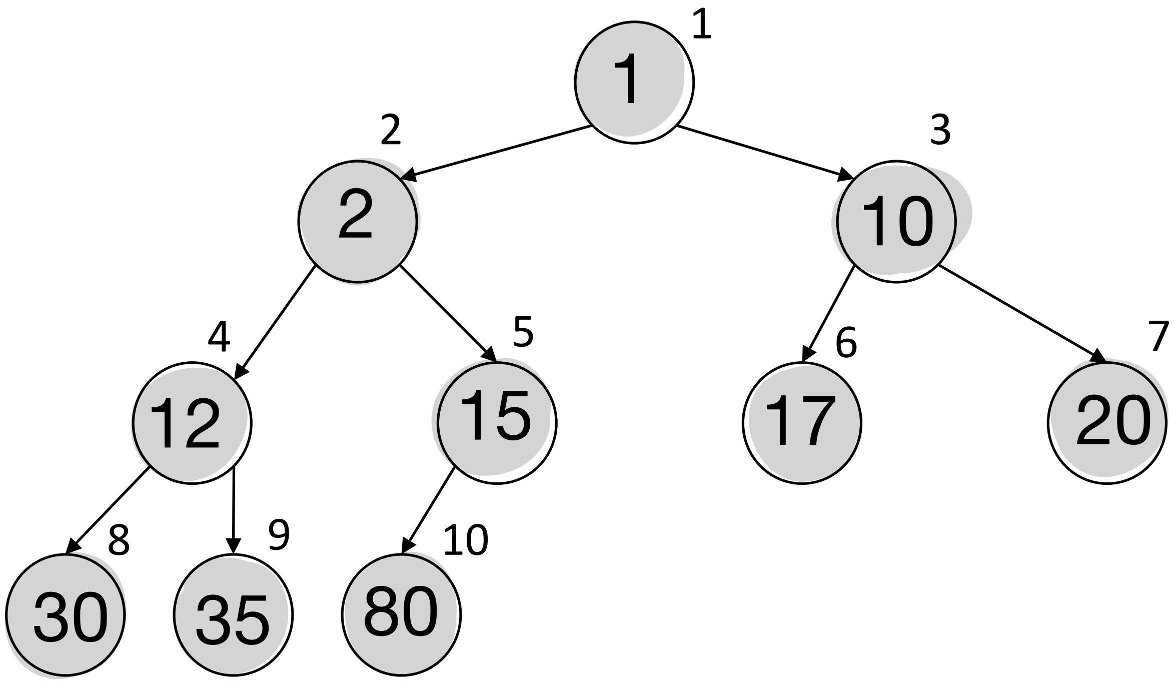
find 3rd highest elem of the array.

step 1 : convert into maxheap

step 2 : delete 2 elems one by one.

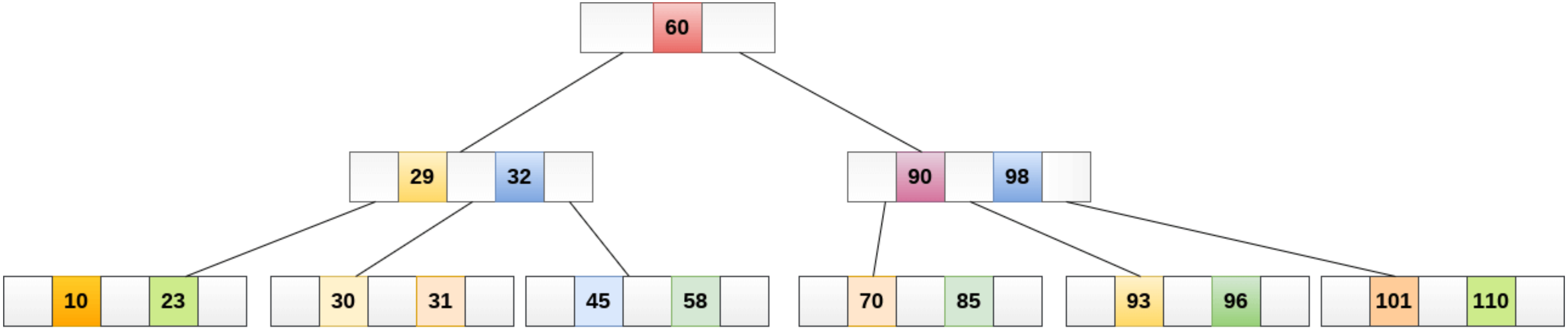
step 3 : return root→arr[1];

Heap Sort



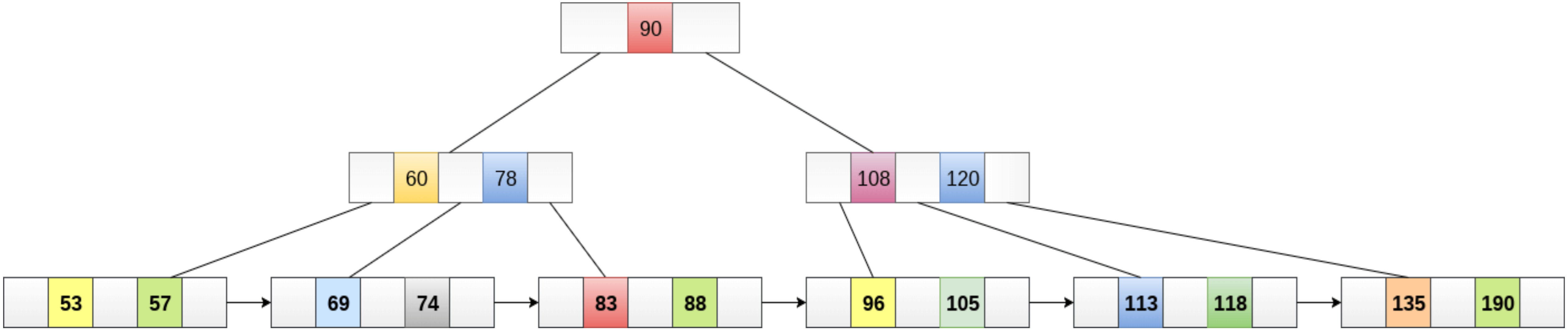
1	2	3	4	5	6	7	8	9	10

B Tree



- A B-Tree of order m can have at most m-1 keys and m children.
- B tree store large number of keys in a single node. This allows storing number of values keeping height minimal.
- Note that in B-Tree all leaf nodes are at same level.
- B-Tree is commonly used for indexing into file systems and databases. It ensures quick data searching and speed up disk access.

B+ Tree

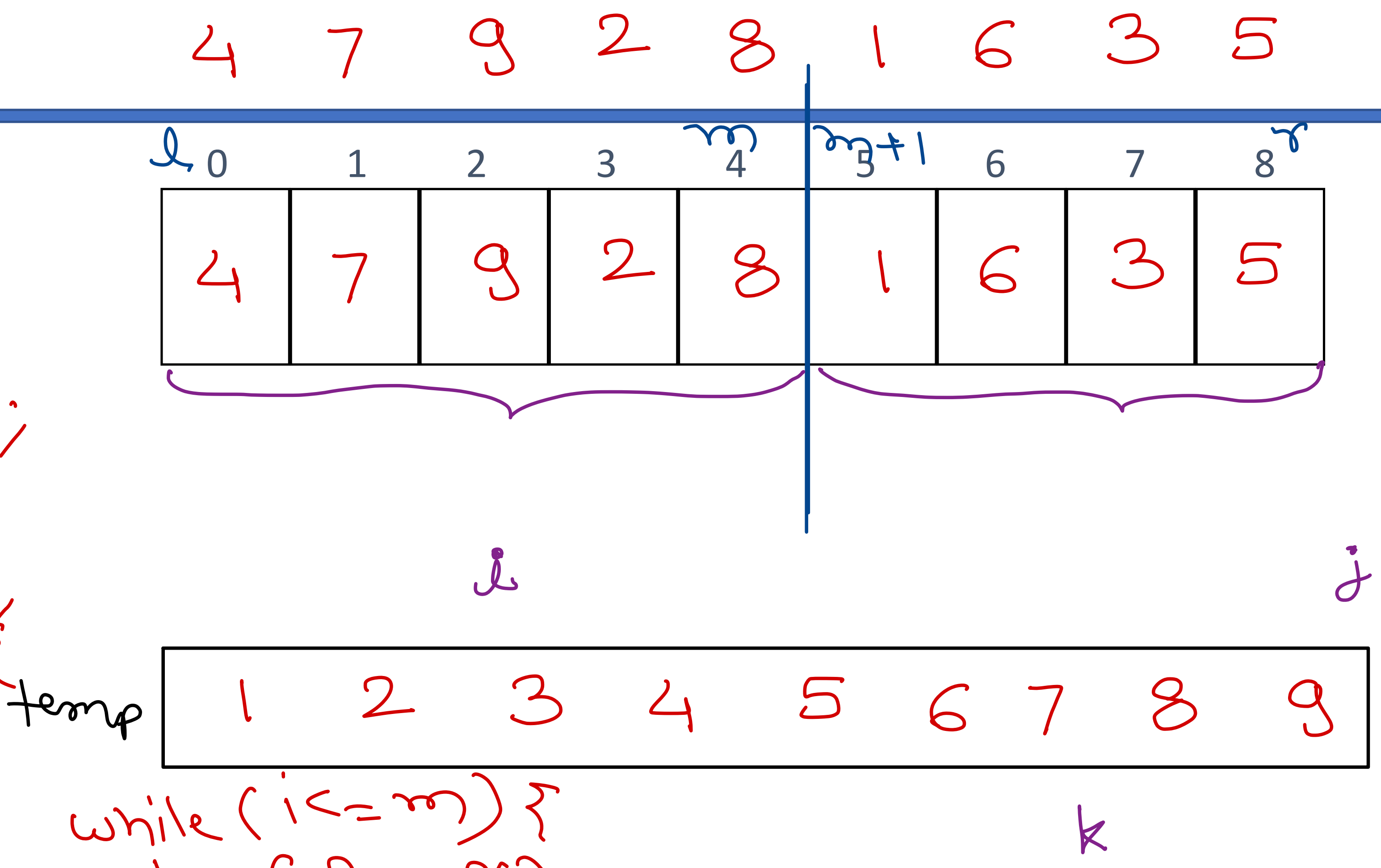


- Extension of B-Tree for efficient insert, delete and search operation.
- Data is stored in leaf nodes only and all leaf nodes are linked together for sequential access.
- Search keys may be redundant.
- Faster searching, simplified deletion (as only from leaf nodes).
- B+Tree is commonly used for indexing into file systems and databases. It ensures quick data searching and speed up disk access.

Merge Sort

→ merge two sorted Partitions into a single array

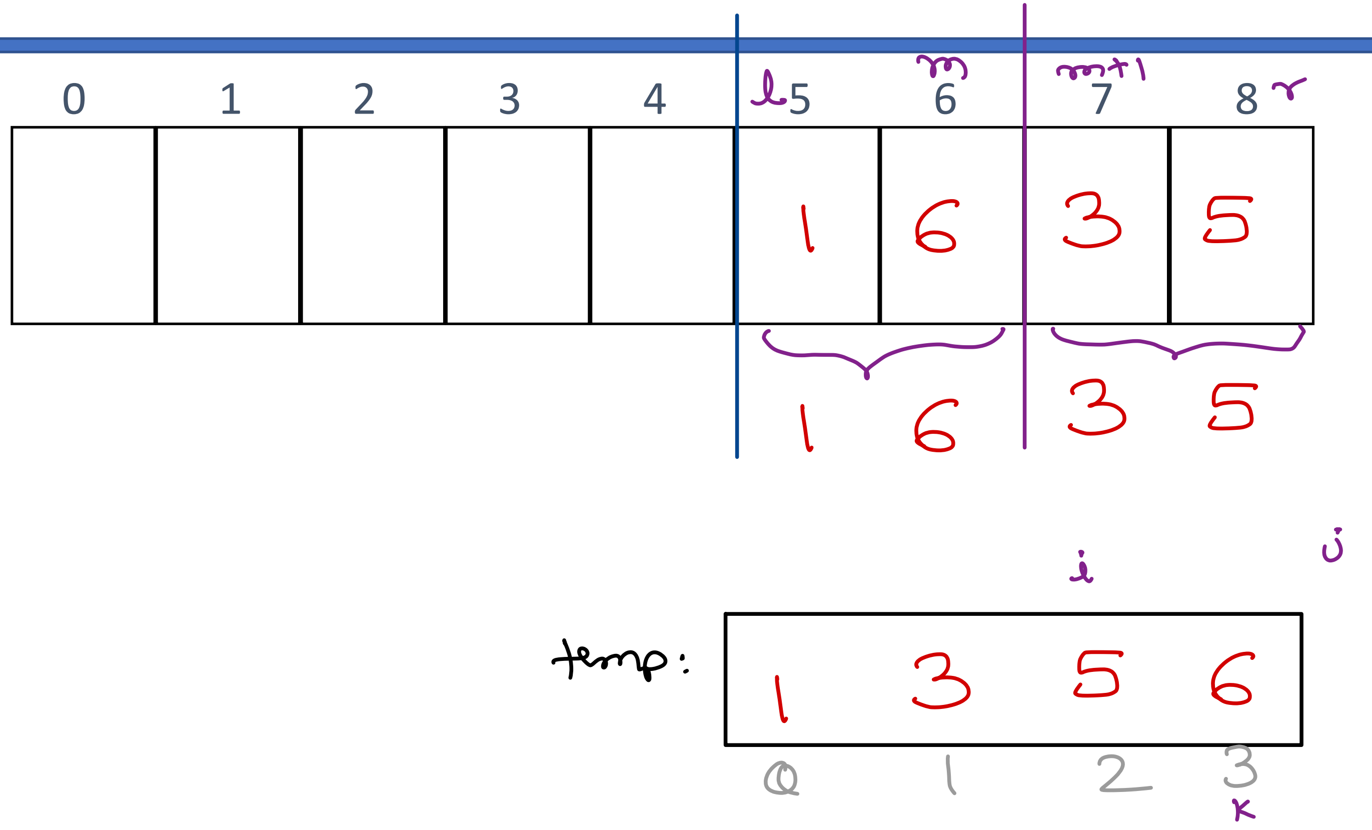
```
if(l>=r)
    return;
m=(l+r)/2;
mergesort(arr,l,m);
mergesort(arr,m+1,r);
i=left; j=m+1; k=0;
temp=new int[r-l+1];
while(i<=m && j<=r){
    if(a[i]<a[j]){
        temp[k]=a[i];
        i++; k++;
    }else{
        temp[k]=a[j];
        j++; k++;
    }
}
```



```
while(i<=m){
    temp[k]=a[i];
    i++;k++;
}
while(j<=r){
    temp[k]=a[j];
    j++;k++;
}
```

```
for(k=0;k<temp.length;k++)
    a[l+k]=temp[k];
```

Merge Sort



0	1	2	3	4	5	6	7	8
					1	6	3	5

0	1	2	3
1	3	5	6

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>