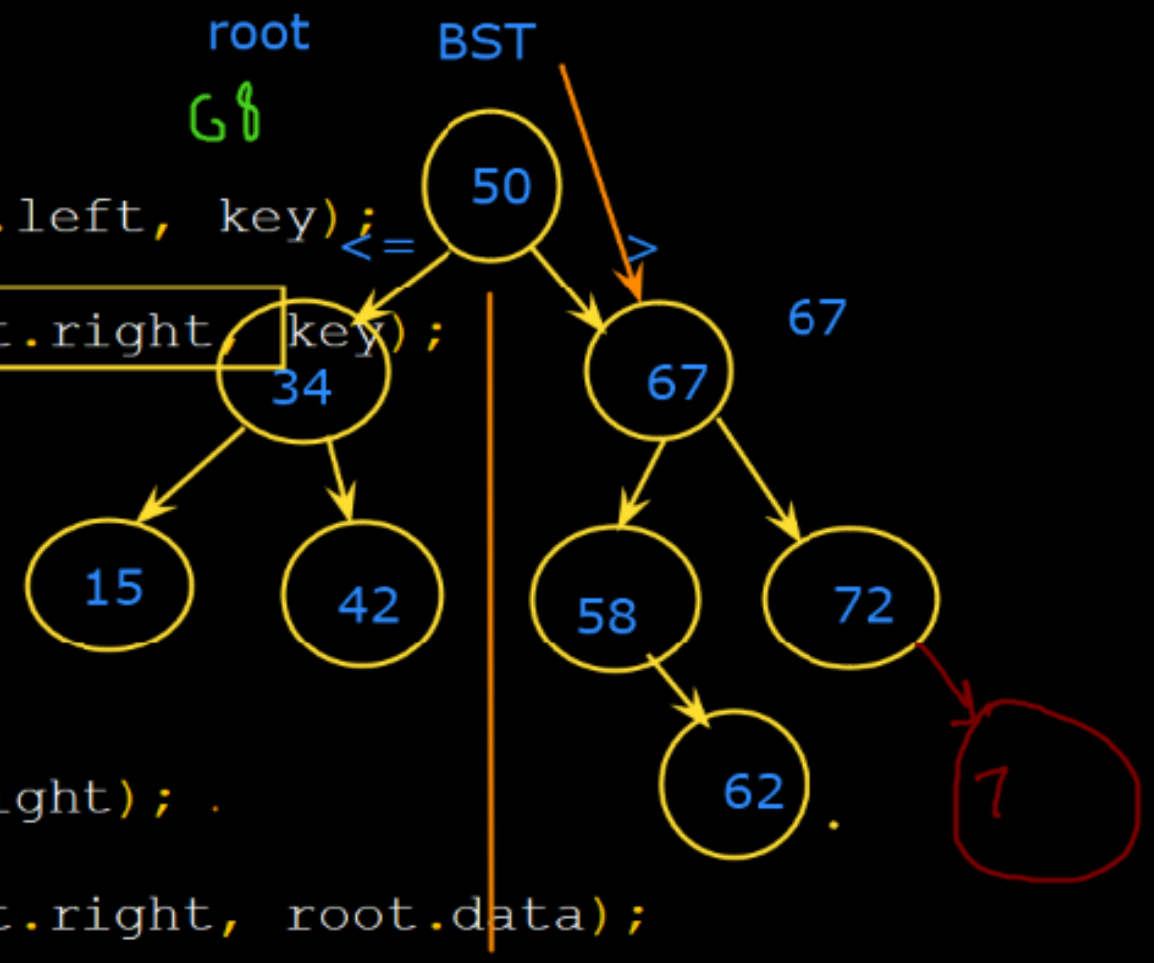
Data Structure

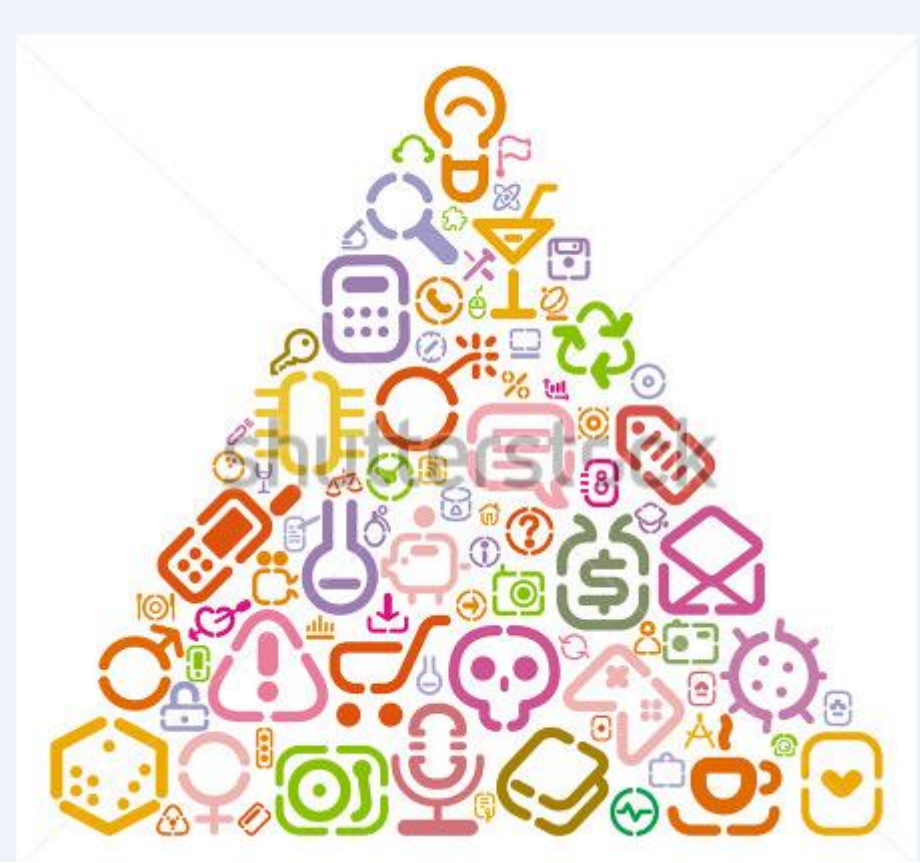
Kiran Waghmare
CDAC Mumbai

```
{
    if(root == null)
        return root;
    if(key < root.data)
        root.left = deletedata(root.left, key);
    else if(key > root.data)
        root.right = deletedata(root.right, key);
    else{
        if(root.left == null)
            return root.right;
        else if(root.right == null)
            return root.left;

        //case 3
        root.data = minvalue(root.right);

        root.right = deletedata(root.right, root.data);
    }
    return root;
}
```



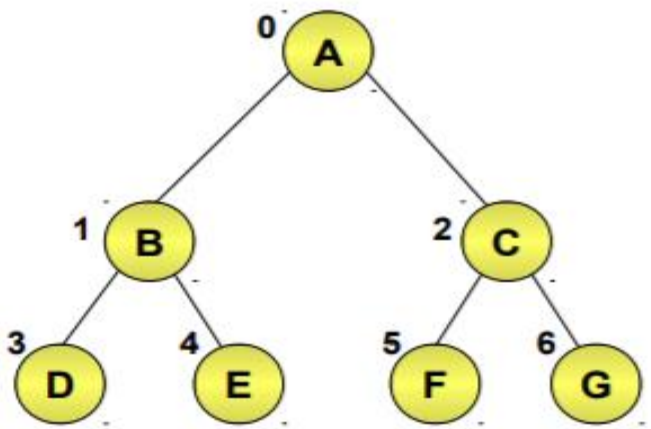
15 34 42 50 58 62 67 72

Heap

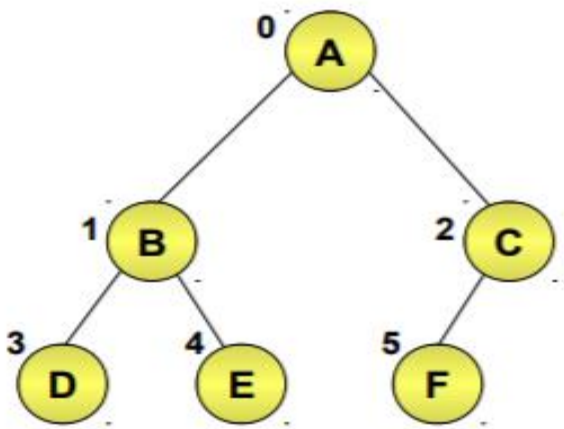


Defining Binary Trees (Contd.)

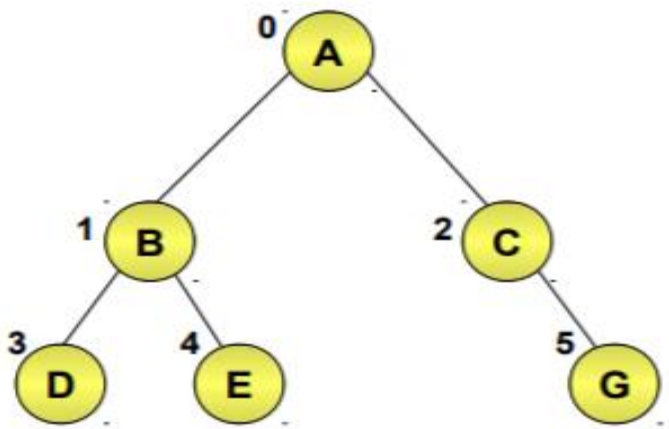
- Complete binary tree:
  - A binary tree with n nodes and depth d whose nodes correspond to the nodes numbered from 0 to n − 1 in the full binary tree of depth k.



Full Binary Tree



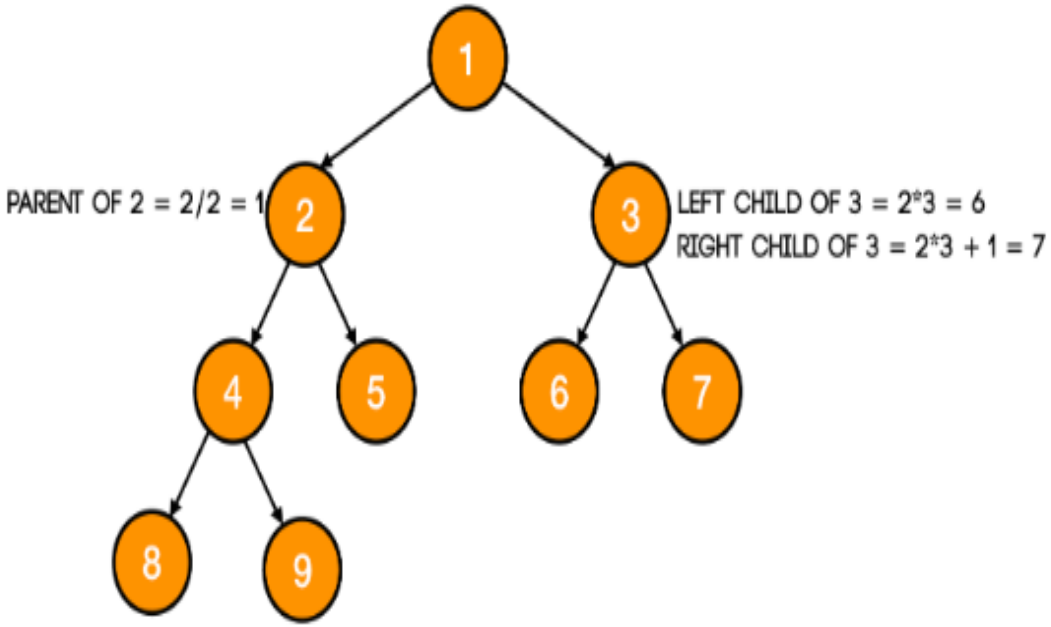
Complete Binary Tree



Incomplete Binary Tree

A complete binary tree also holds some important properties. So, let's look at them.

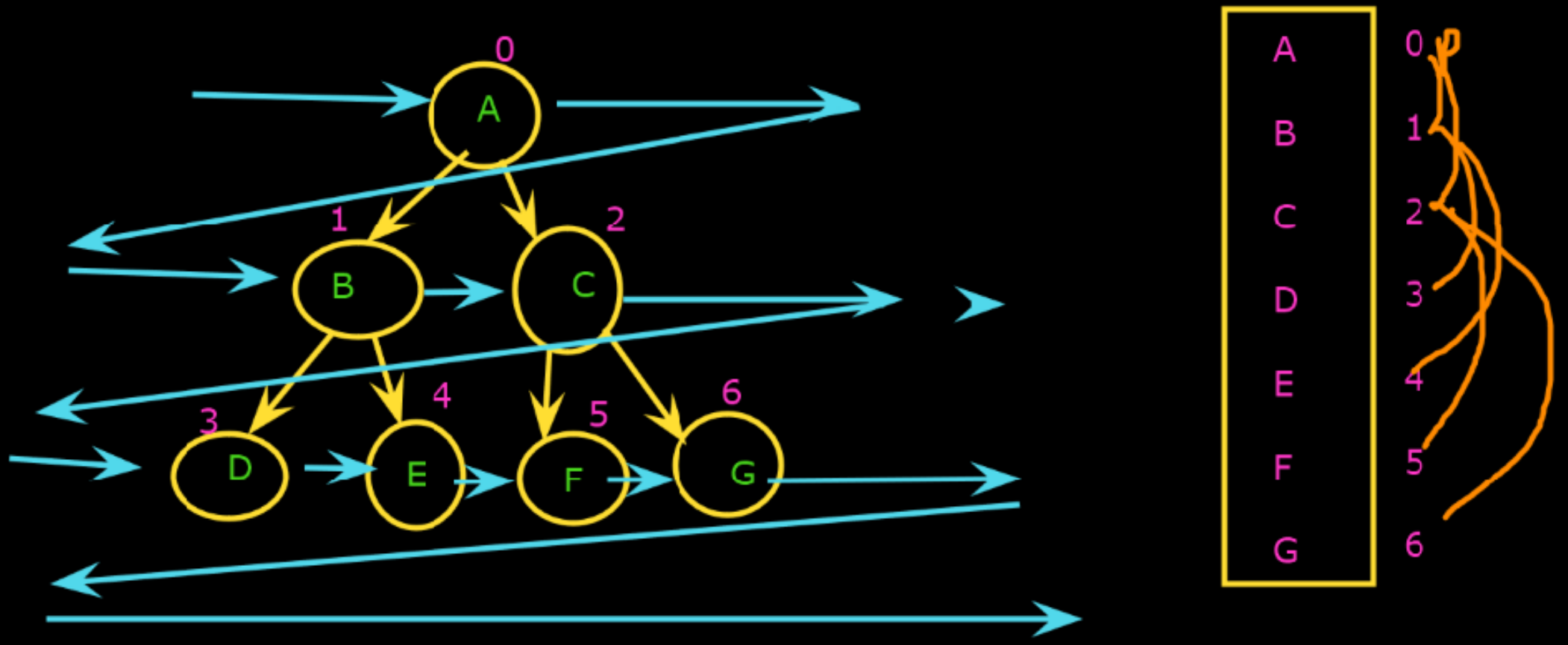
- The **parent of node i** is $\left\lfloor \frac{i}{2} \right\rfloor$. For example, the parent of node 4 is 2 and the parent of node 5 is also 2.
- The **left child of node i** is $2i$.
- The **right child of node i** is $2i + 1$



Binary Tree:

A binary tree is a tree in which every node has at most two children.

0,1,2,2



Array Representation of Binary Tree

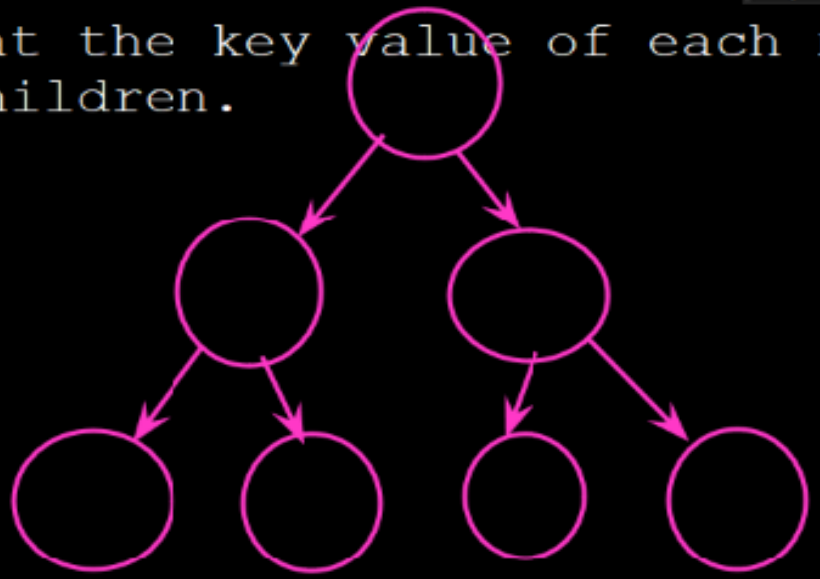
DEfinition:

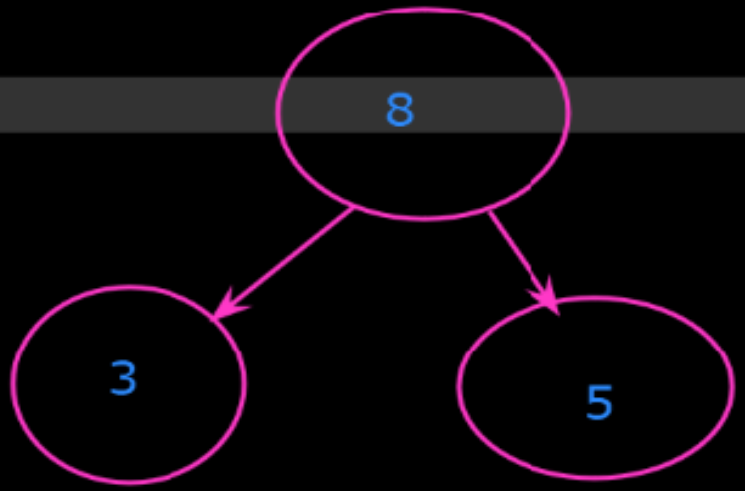
-a special form of complete binary tree that the key value of each no
smaller(larger) than the key value of it children.

Types of heap:

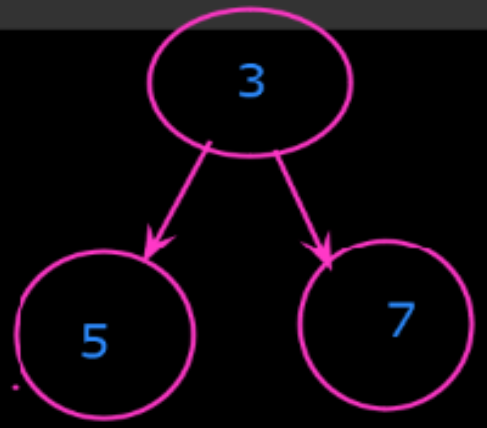
1. Max-Heap: root node has largest value

2. Min-Heap: root node has smallest value





Max Heap



Min Heap

Heap

Heap

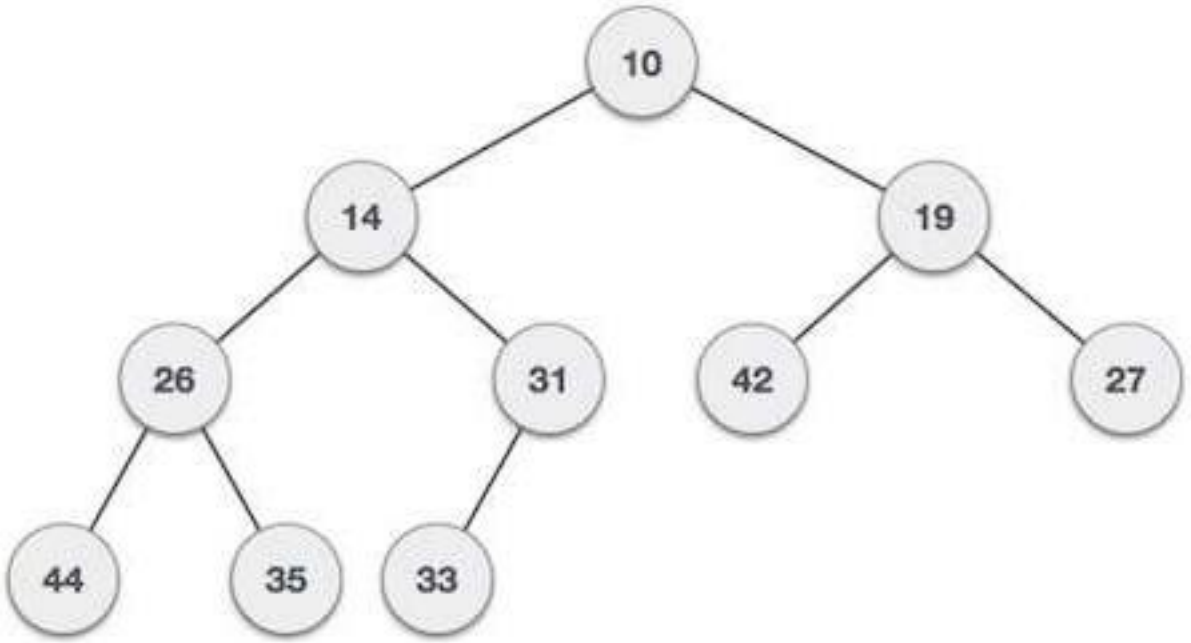
- **Definition in Data Structure**
  - **Heap:** A special form of complete binary tree that key value of each node is no smaller (larger) than the key value of its children (if any).

- **Max-Heap: root node has the largest key.**
  - A *max tree* is a tree in which the key value in each node is no smaller than the key values in its children.
  - A *max heap* is a complete binary tree that is also a max tree.

- **Min-Heap: root node has the smallest key.**
  - A *min tree* is a tree in which the key value in each node is no larger than the key values in its children.
  - A *min heap* is a complete binary tree that is also a min tree.

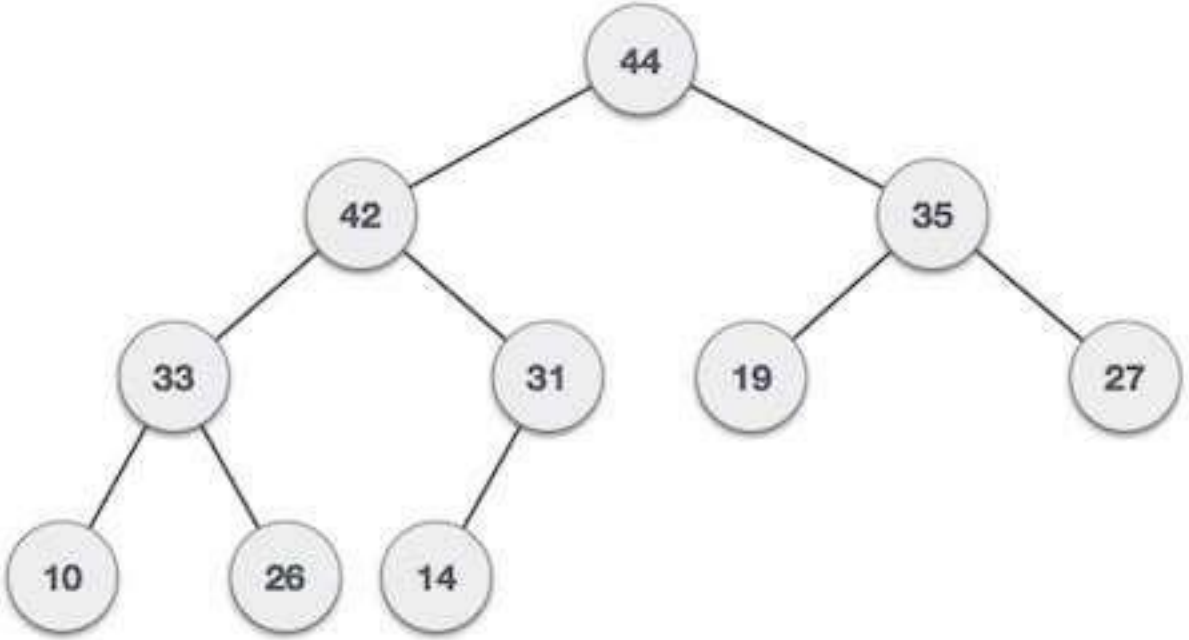
Heap

- Min-Heap
  - Where the value of the root node is less than or equal to either of its children
  - For input 35 33 42 10 14 19 27 44 26 31



Heap

- Max-Heap –
  - where the value of root node is greater than or equal to either of its children.
  - For input 35 33 42 10 14 19 27 44 26 31



Types of heap:

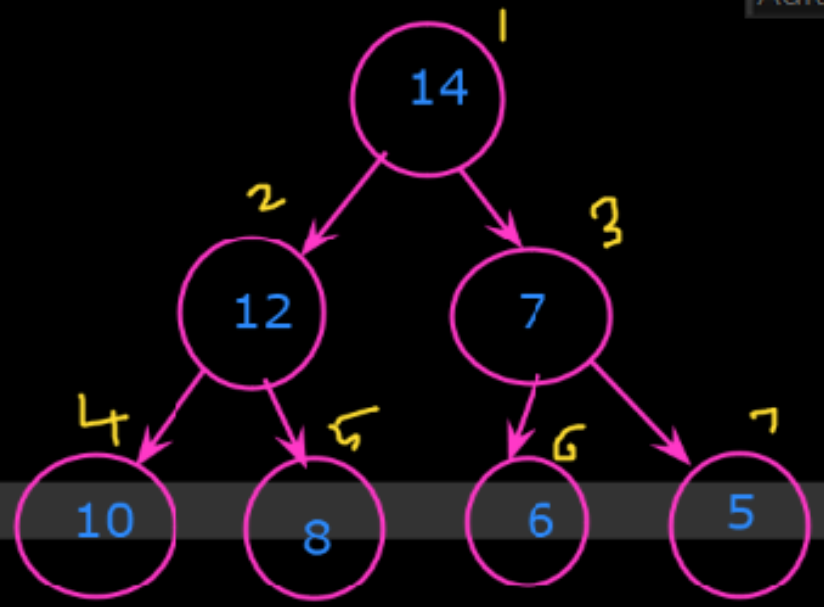
1. Max-Heap: root node has largest value

2. Min-Heap: root node has smallest value

Parent = i/2

Lc = 2i

RC = 2i+1



Max heap

14	12	7	10	8	6	5
1	2	3	4	5	6	7

Types of heap:

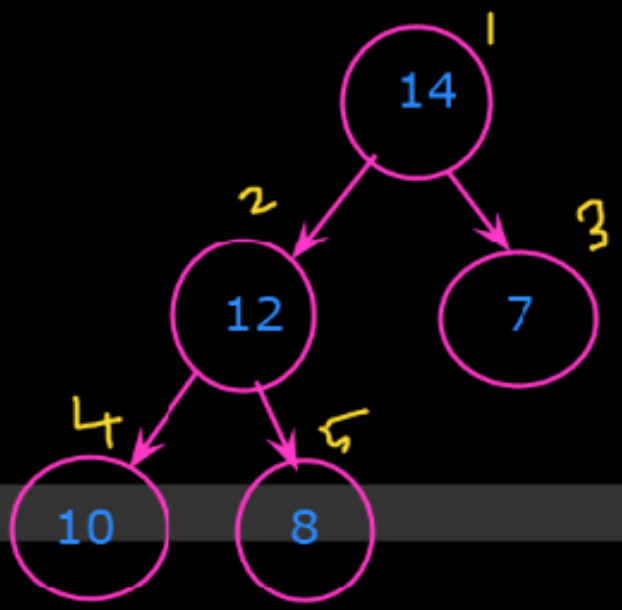
1. Max-Heap: root node has largest value

2. Min-Heap: root node has smallest value

Parent = i/2

Lc = 2i

RC = 2i+1



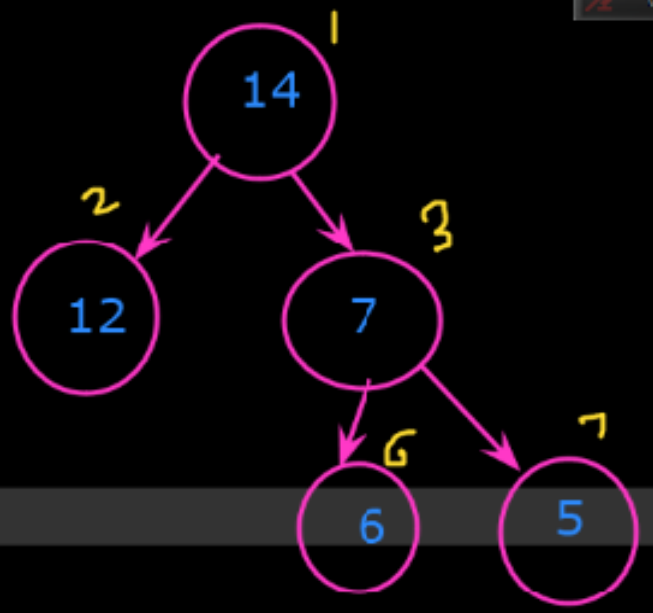
Max heap

14	12	7	10	8
1	2	3	4	5

Types of heap:

1. Max-Heap: root node has largest value

2. Min-Heap: root node has smallest value



Max heap

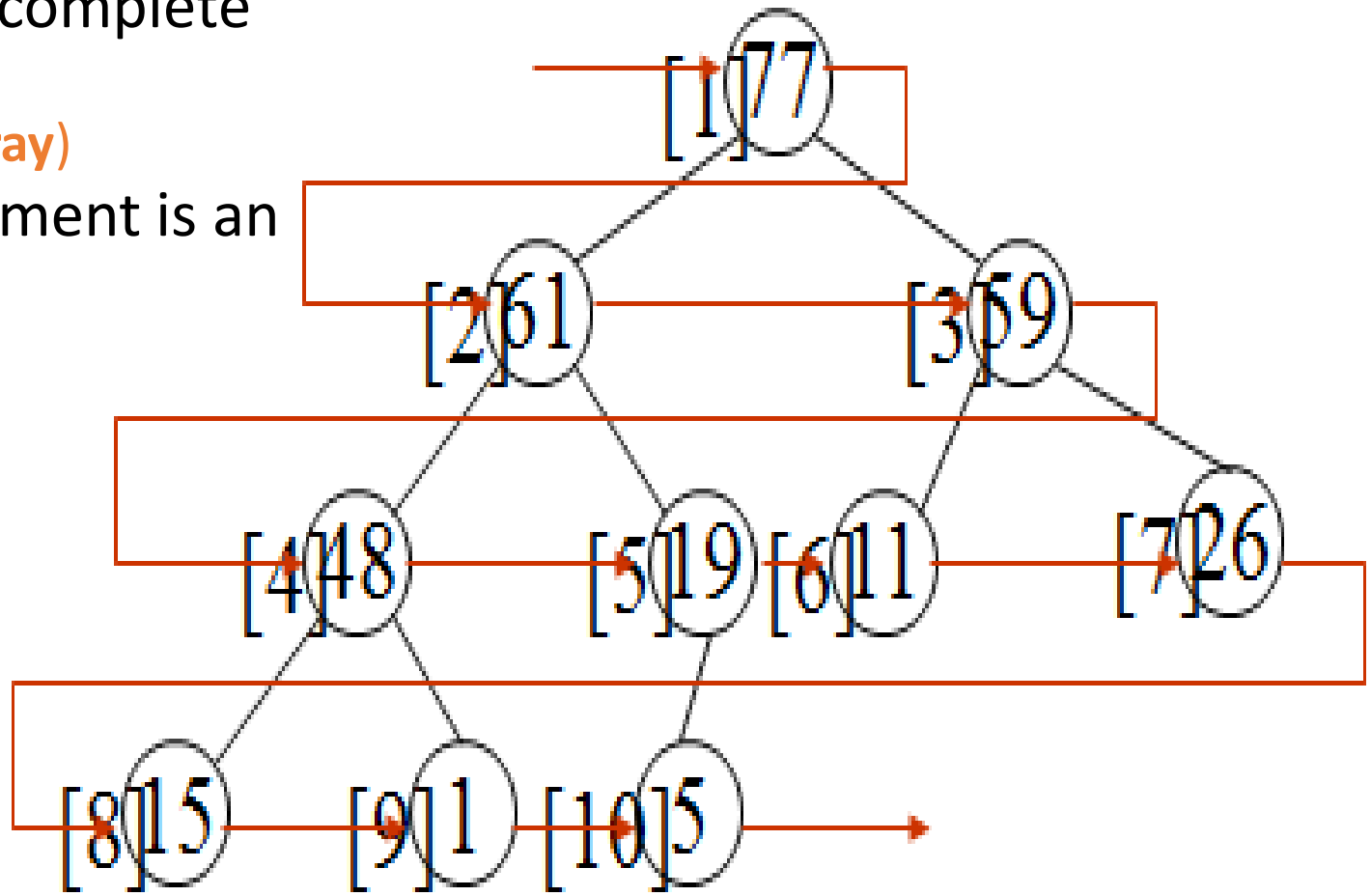
Parent = i/2

Lc = 2i

RC = 2i+1

14	12	7	-	-	6	5
1	2	3	4	5	6	7

- **Note:**
  - Heap in data structure is a complete binary tree!
    - (Nice representation in Array)
  - Heap in C program environment is an array of memory.
-

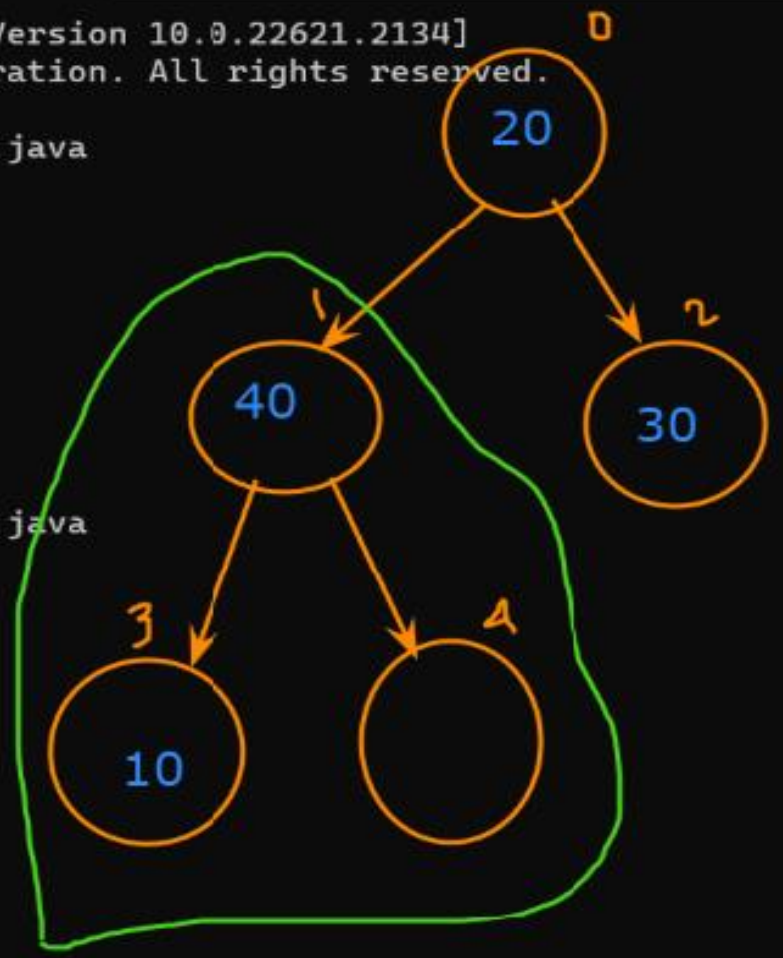


— Stored using array in C

index	1	2	3	4	5	6	7	8	9	10
value	77	61	59	48	19	11	26	15	1	5

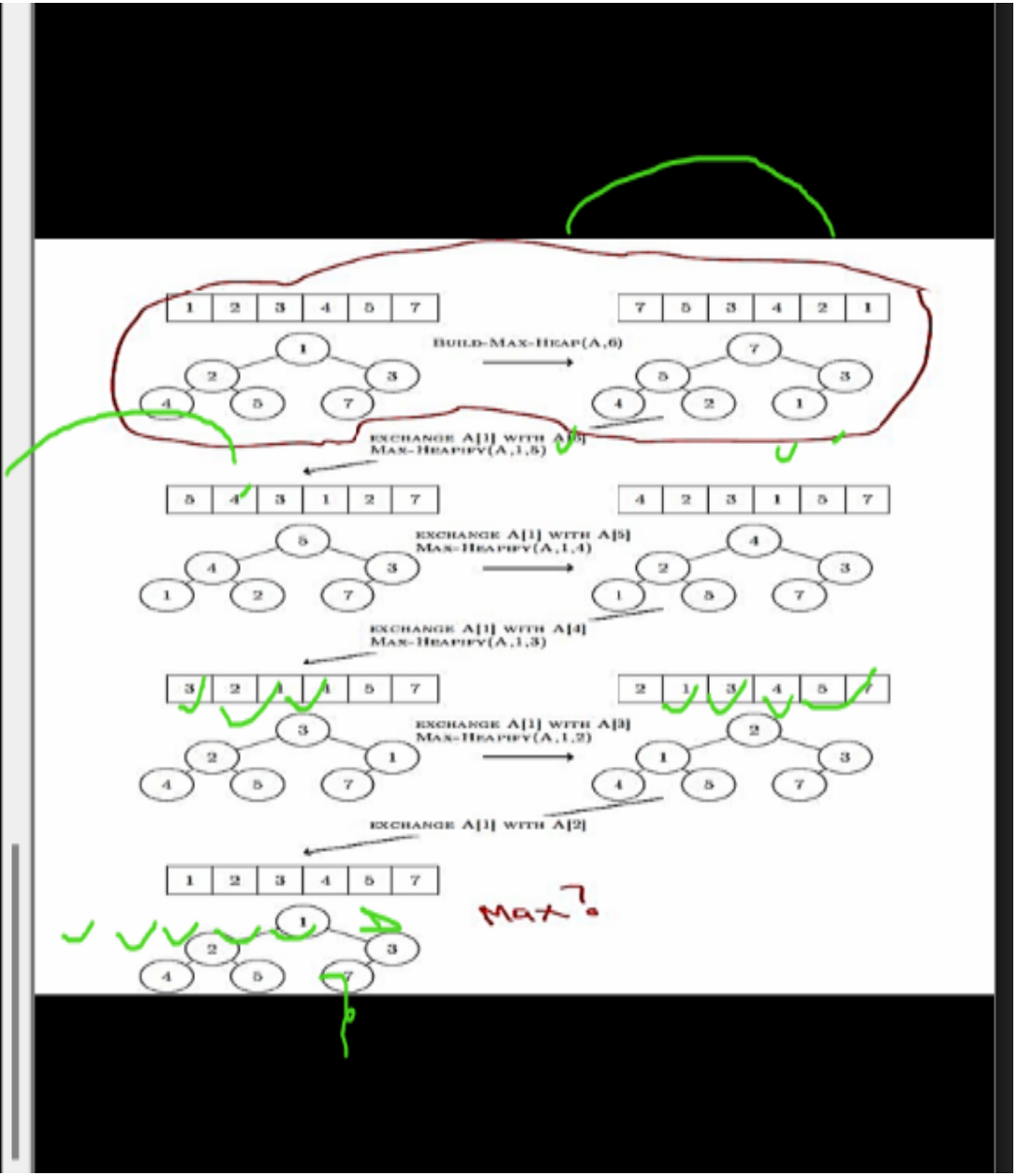
```
        heapify(arr,n,i);

    //n-1 elements consider ka
    for(int i=n-1;i>0;i--)
    {
        //replacement of last
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        //balancing of maxheap
        heapify(arr,i,0);

    }

      10 20 30 40 50

}

void display(int arr[])
{
    int n = arr.length;
    for(int i=0;i<n;i++)
```

```
C:\Windows\system32\cmd.e:
Microsoft Windows [Version 10.0.22621.2134]
(c) Microsoft Corporation. All rights reserved.

D:\Test>javac HSort.java

D:\Test>java HSort
Unsorted array:
10
40
30
50
20

D:\Test>javac HSort.java

D:\Test>java HSort
Unsorted array:
10
40
30
50
20
Sorted array:
10
20
30
40
50

D:\Test>
```

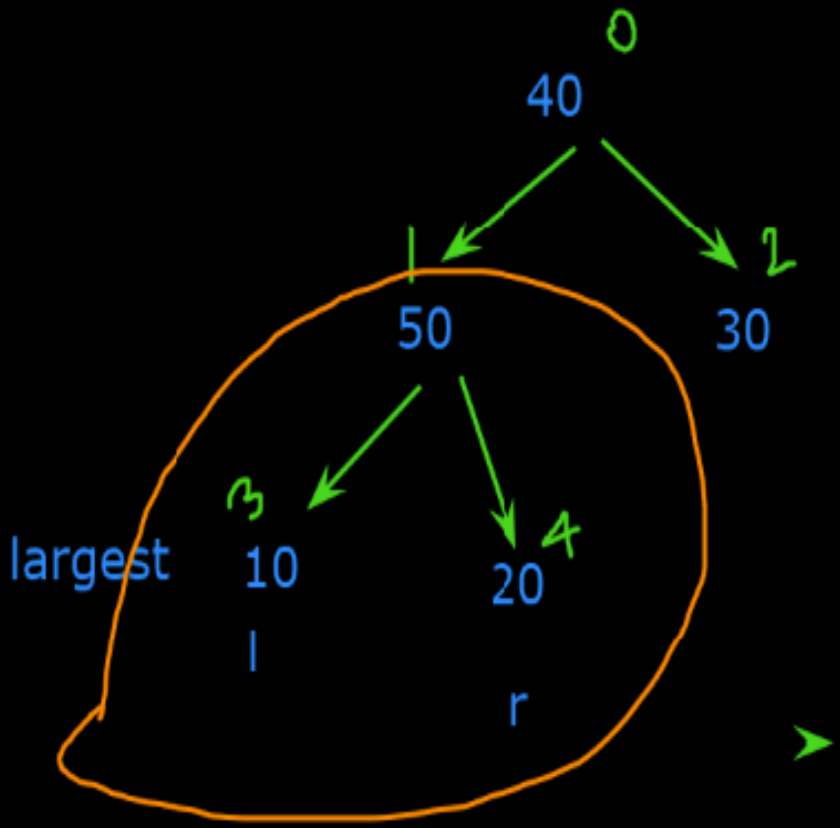


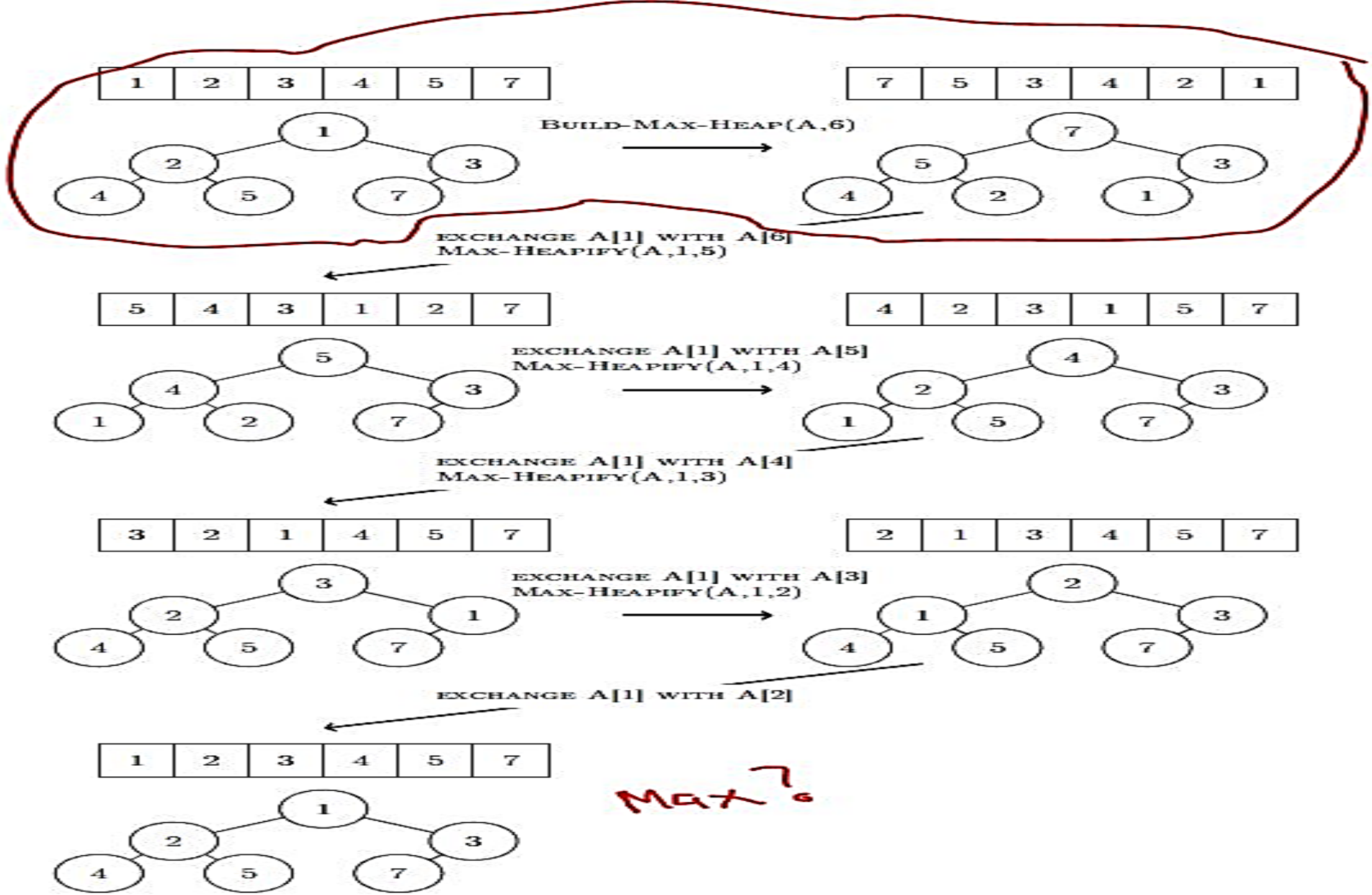
50

```java
public static void main(String args[])
{
    HSort h1 = new HSort();
    int arr[] = {10, 40, 30, 50, 20};
    System.out.println("Unsorted array:");
    h1.display(arr);
    h1.heapsort(arr);
    System.out.println("Sorted array:");
    h1.display(arr);


}

}
```



```
//to construct max heap at Root, LC and RC
void heapify(int arr[],int n, int i)
{
    //Max heap
    int largest = i;//i/2 : Parent
    int l = 2*i+1;//2*i : LC
    int r = 2*i+2;//2*i+1 : RC

    if(l < n && arr[l] > arr[largest])
        largest = l;
    if(r < n && arr[r] > arr[largest])
        largest = r;

    if(largest != i)
    {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        heapify(arr,n,largest);
    }
}
```




1 2 3 4 5 7
7 5 3 4 2 1
BUILD-MAX-HEAP(A,6)
EXCHANGE A[1] WITH A[6]
MAX-HEAPIFY(A,1,5)
5 4 3 1 2 7
4 2 3 1 5 7
EXCHANGE A[1] WITH A[5]
MAX-HEAPIFY(A,1,4)
EXCHANGE A[1] WITH A[4]
MAX-HEAPIFY(A,1,3)
3 2 1 4 5 7
2 1 3 4 5 7
EXCHANGE A[1] WITH A[3]
MAX-HEAPIFY(A,1,2)
EXCHANGE A[1] WITH A[2]
1 2 3 4 5 7
Max 7

Thanks