Leftist Heap

Outline

Introduction

Operations

Time Complexity

Example

MERGEABLE HEAP

A mergeable heap is an abstract data type , which is a heap supporting the merge operation

$$H \leftarrow \text{Merge}(H_1, H_2).$$

The two major implementations of mergeable heap are **Binomial heap** and **Leftist heap**

LEFTIST TREE

It is a binary tree that satisfies the **heap property** and an additional property known as the **leftist property**

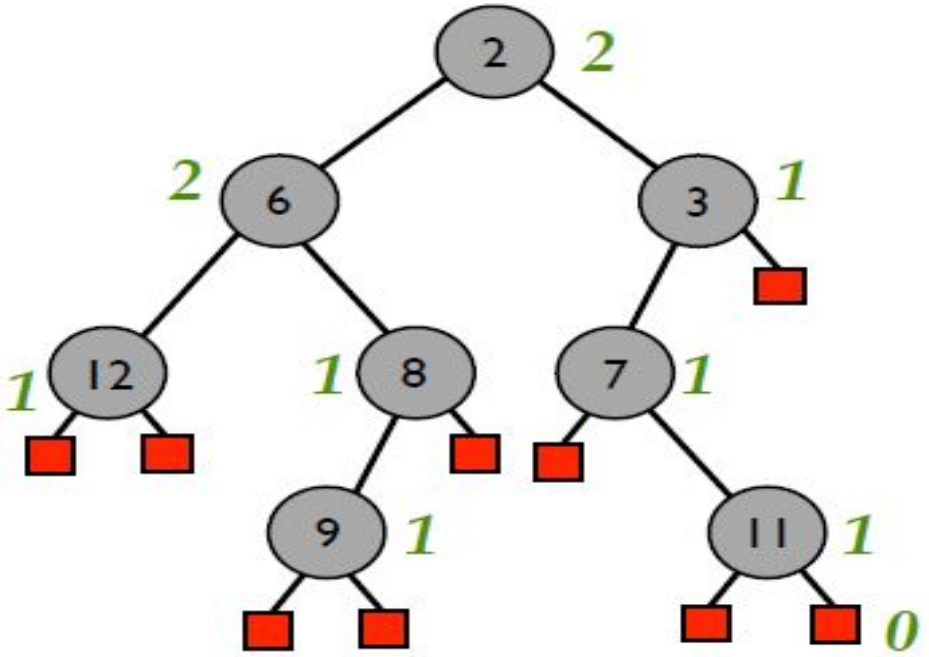
Invented by Clark Allen Crane (1971)

HEAP PROPERTY

$$Key(i) >= Key(parent(i))$$

NULL PATH LENGTH(RANK)

Number of edges on the shortest path from node i to a descendant external node

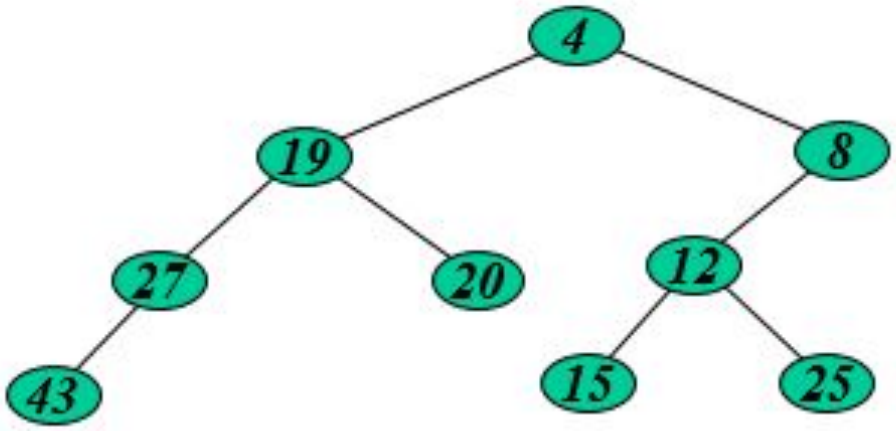


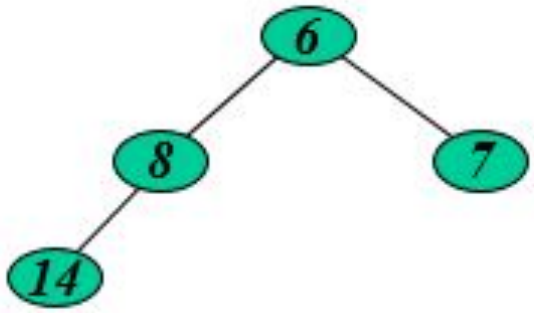
LEFTIST HEAP PROPERTY

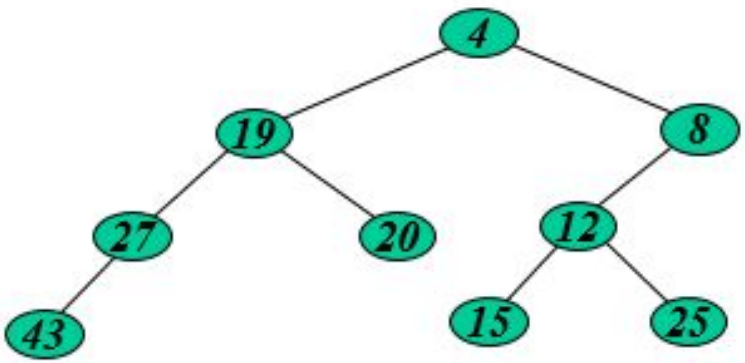
---

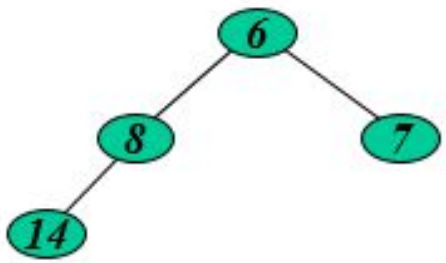
npl(left(u)) ≥ npl(right(u))

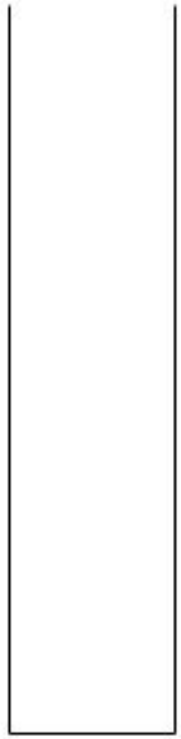
- Every subtree is also a leftist tree
- npl(i) = 1+npl(right(i))


4
19
8
27
20
12
43
15
25

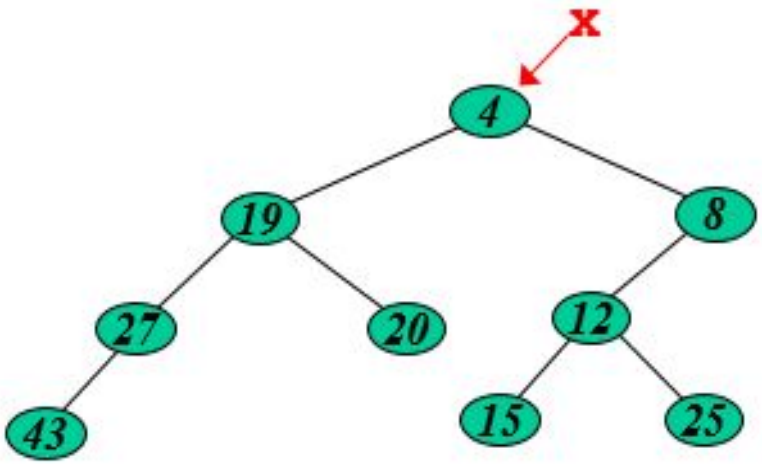

6
8
7
14

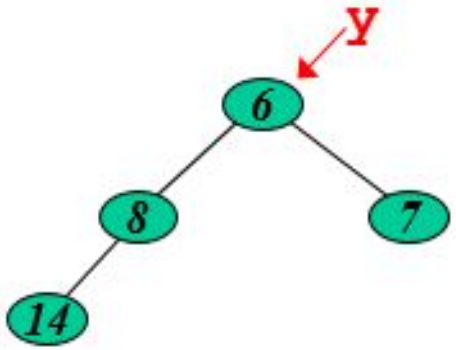


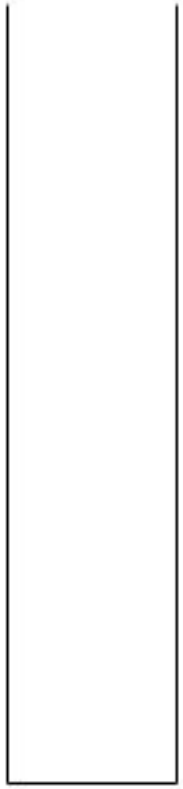




First, instantiate a Stack

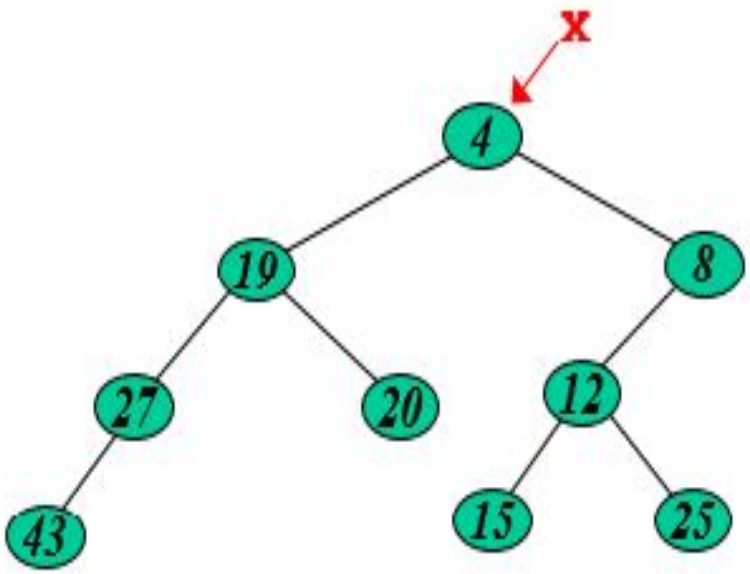


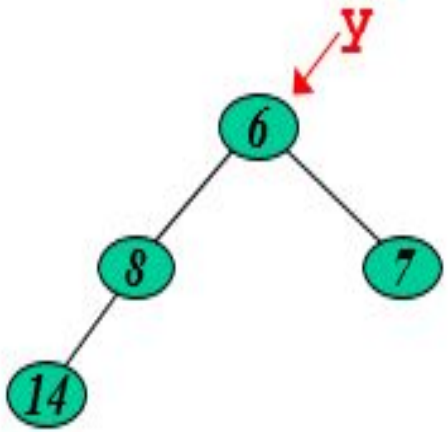


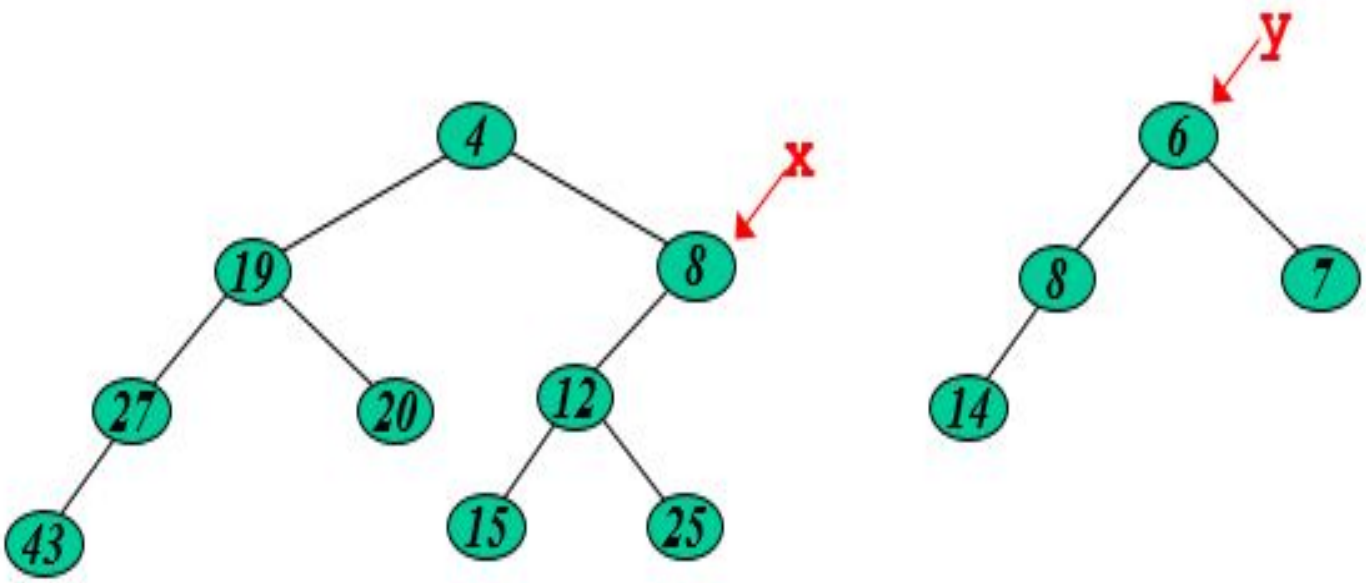


Compare root nodes

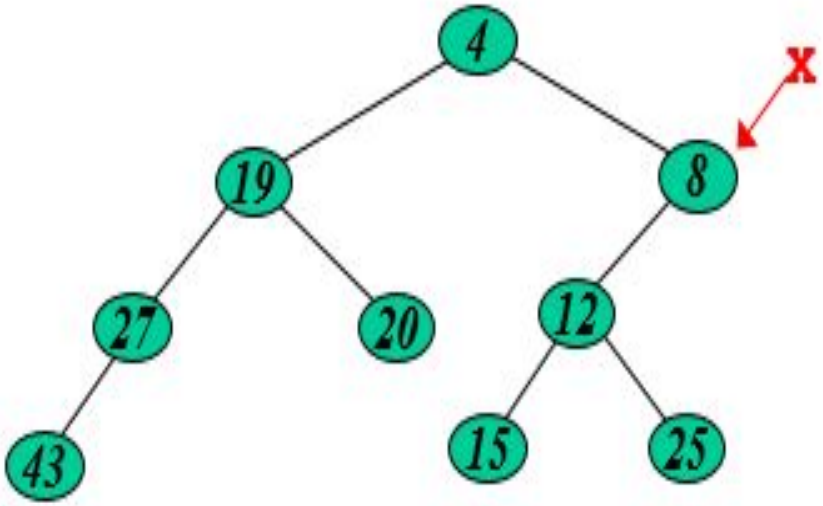
`merge(x,y)`

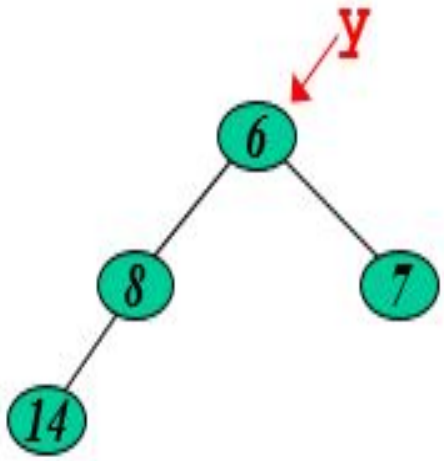

x
4
19
8
27
20
12
43
15
25

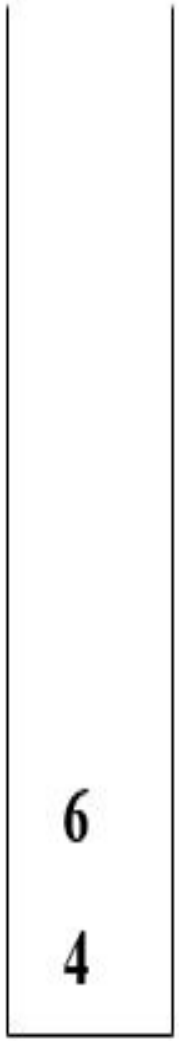

y
6
8
7
14

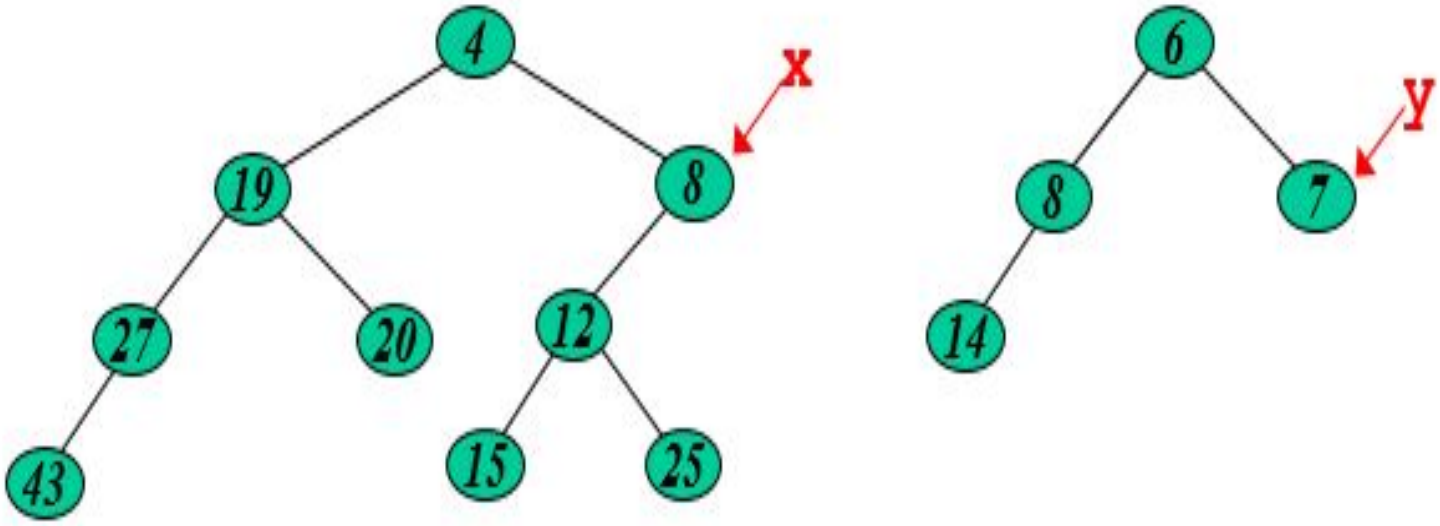


4


4
19
8
27
20
12
43
15
25
x

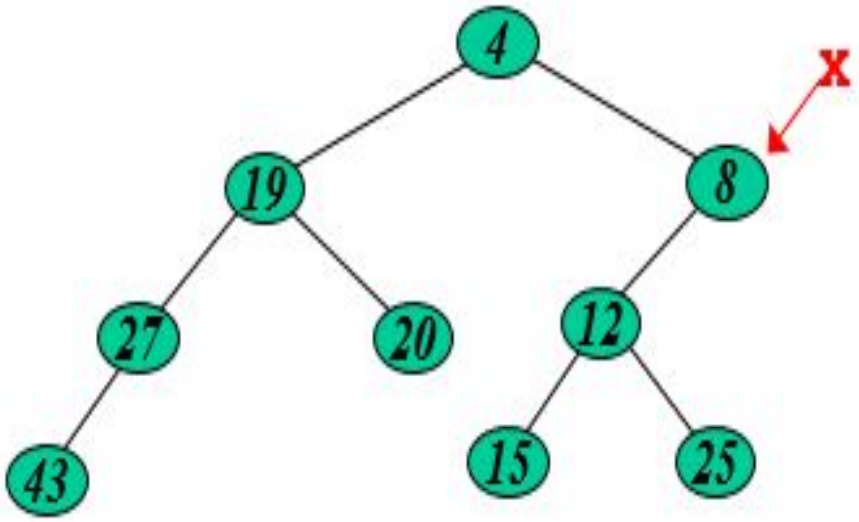

y
6
8
7
14

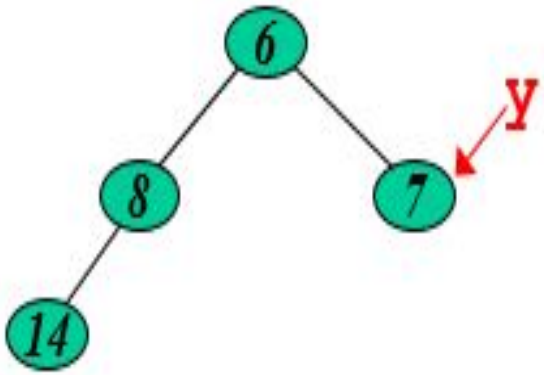

6
4

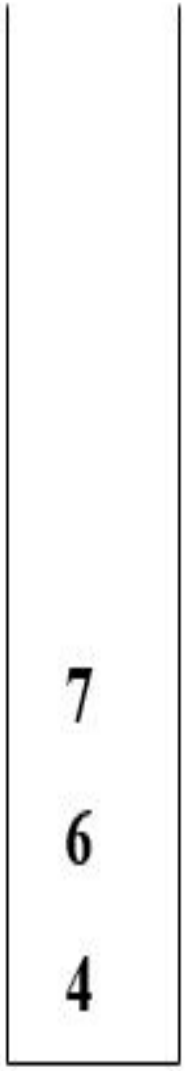


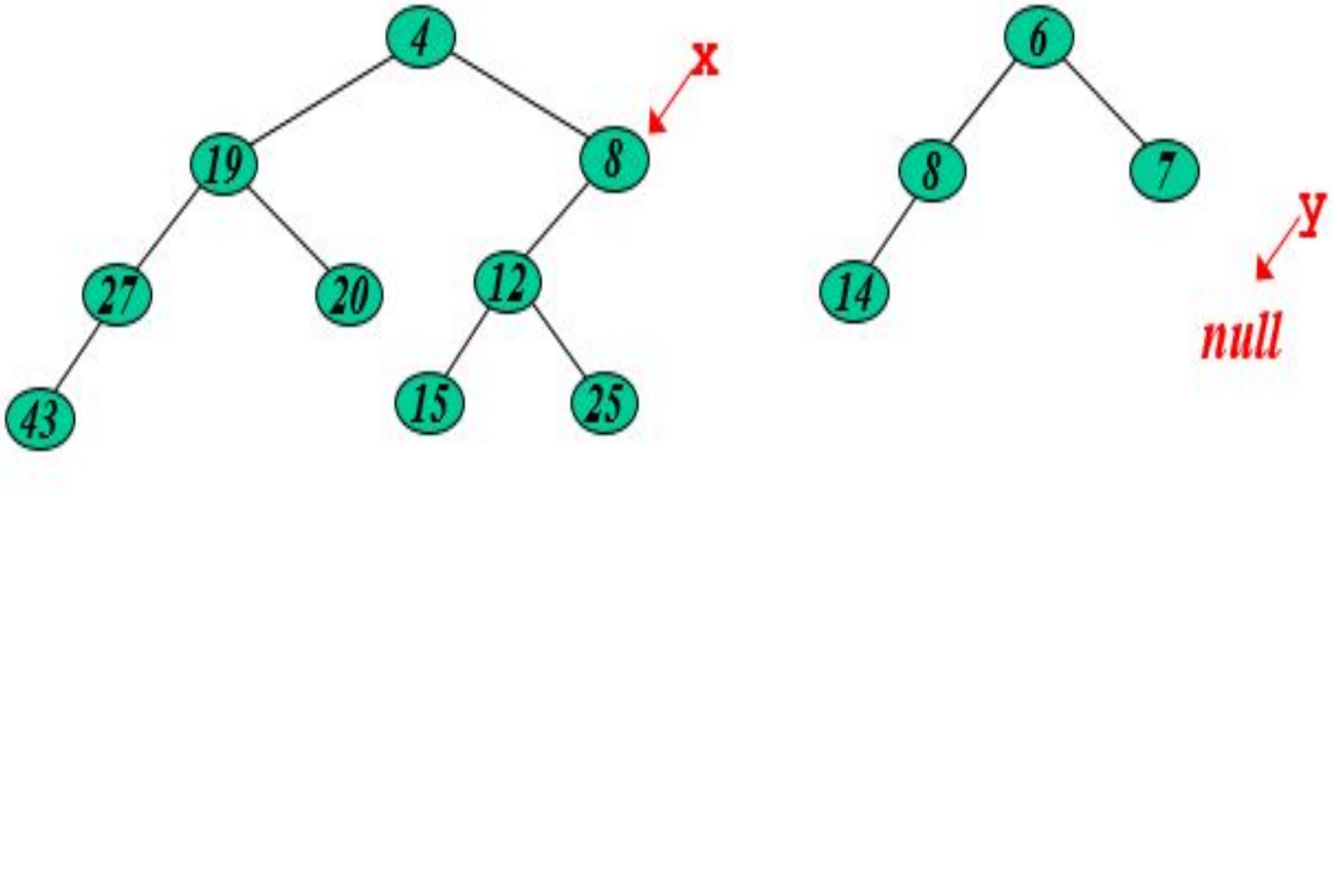
6

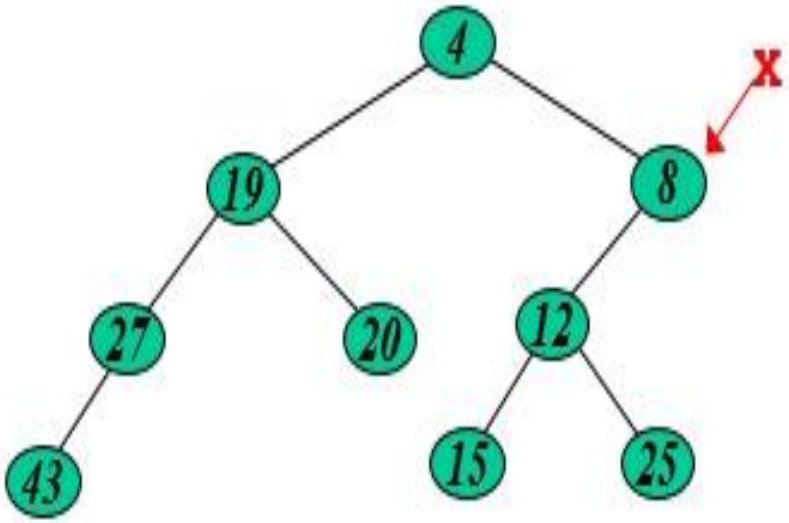
4

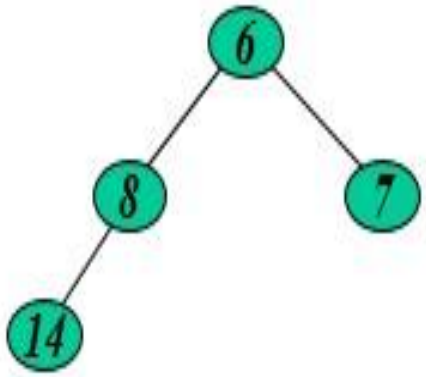

4
19
8
27
20
12
43
15
25
x

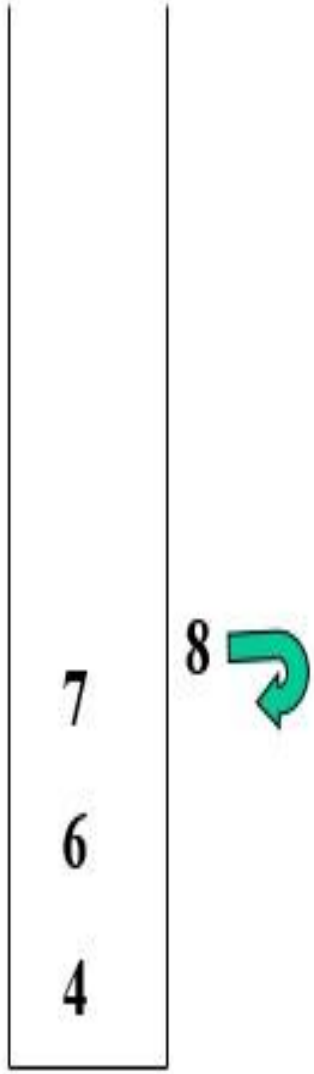

6
8
7
14
y

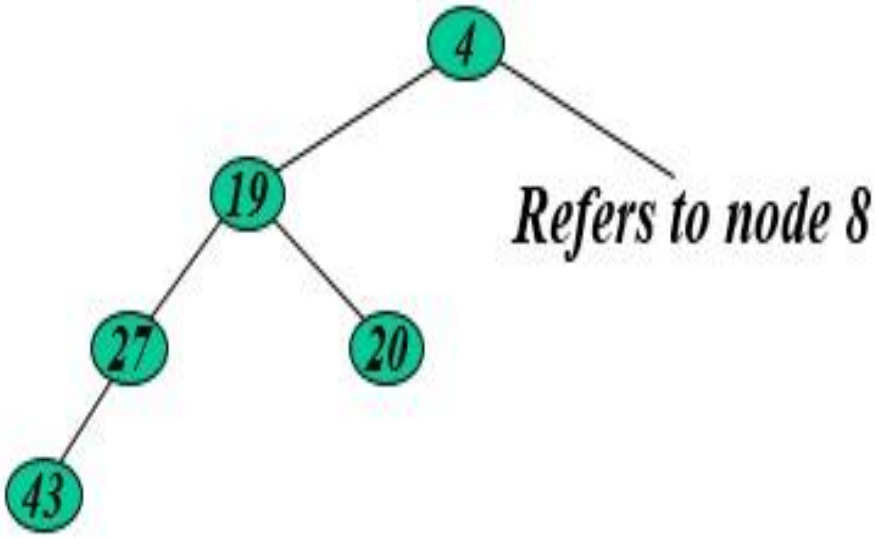

7
6
4


4
19
8
27
20
12
43
15
25
x
6
8
7
14
y
null
7
6
4

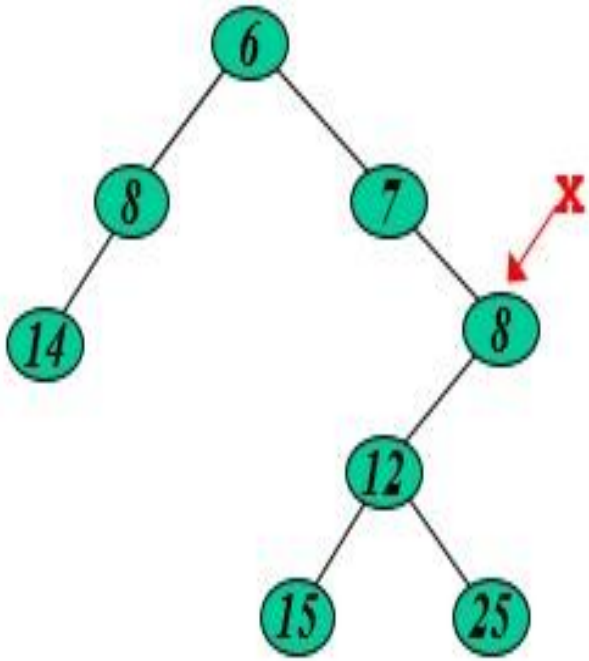

4
19
8
27
20
12
43
15
25
x

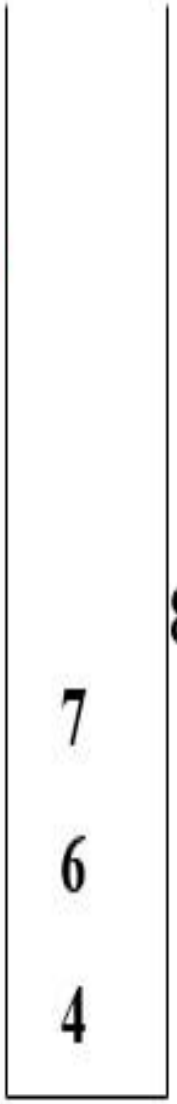

6
8
7
14

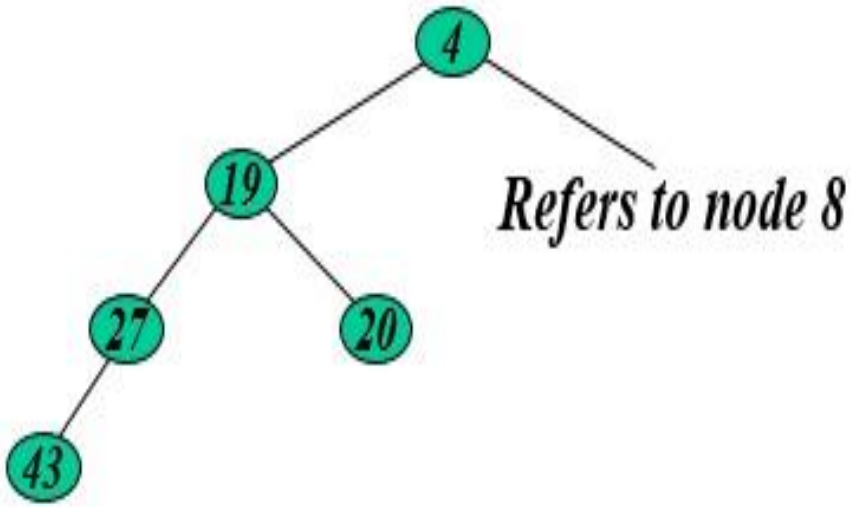

8
7
6
4



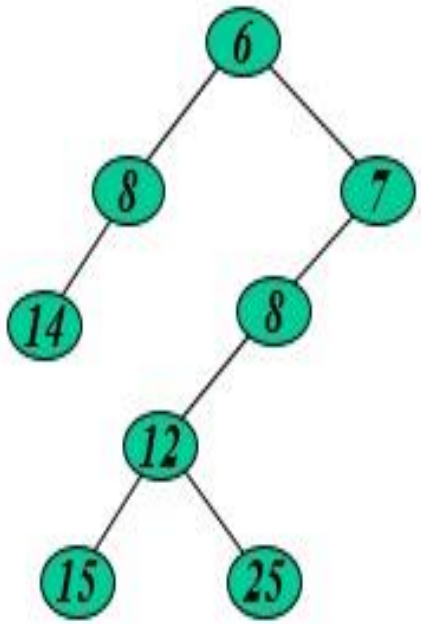
Refers to node 8

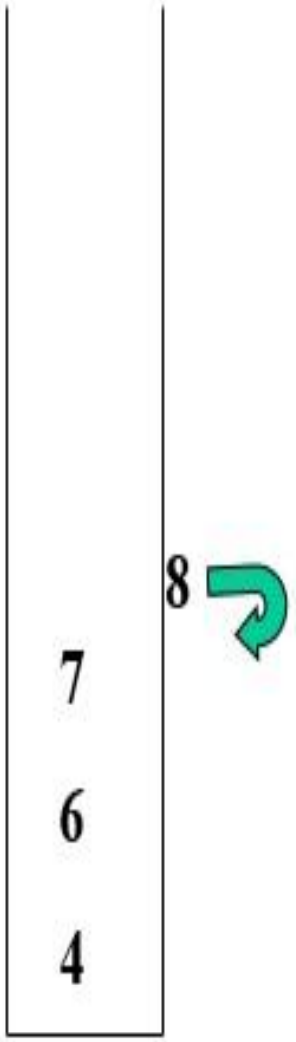


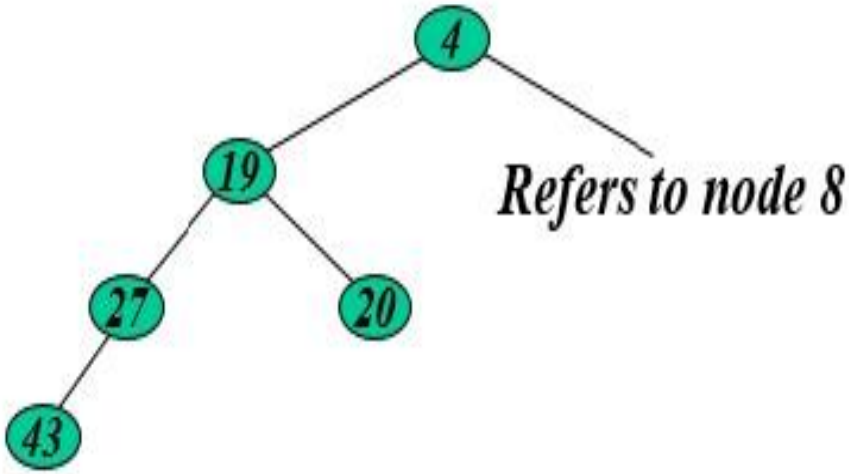


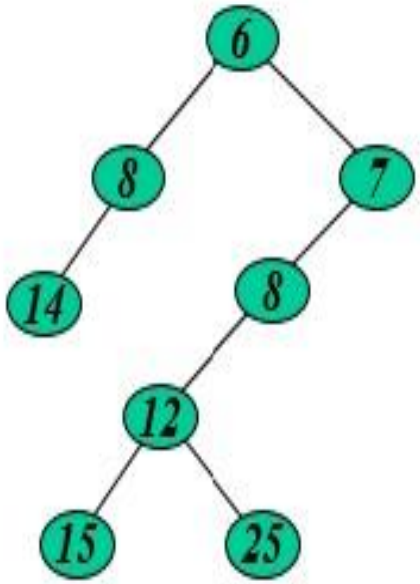


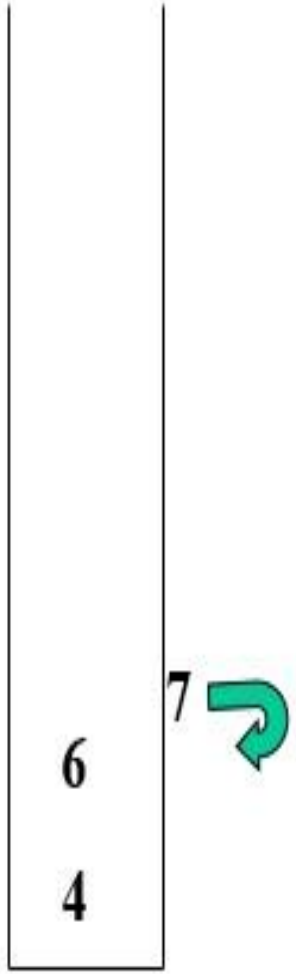
Refers to node 8

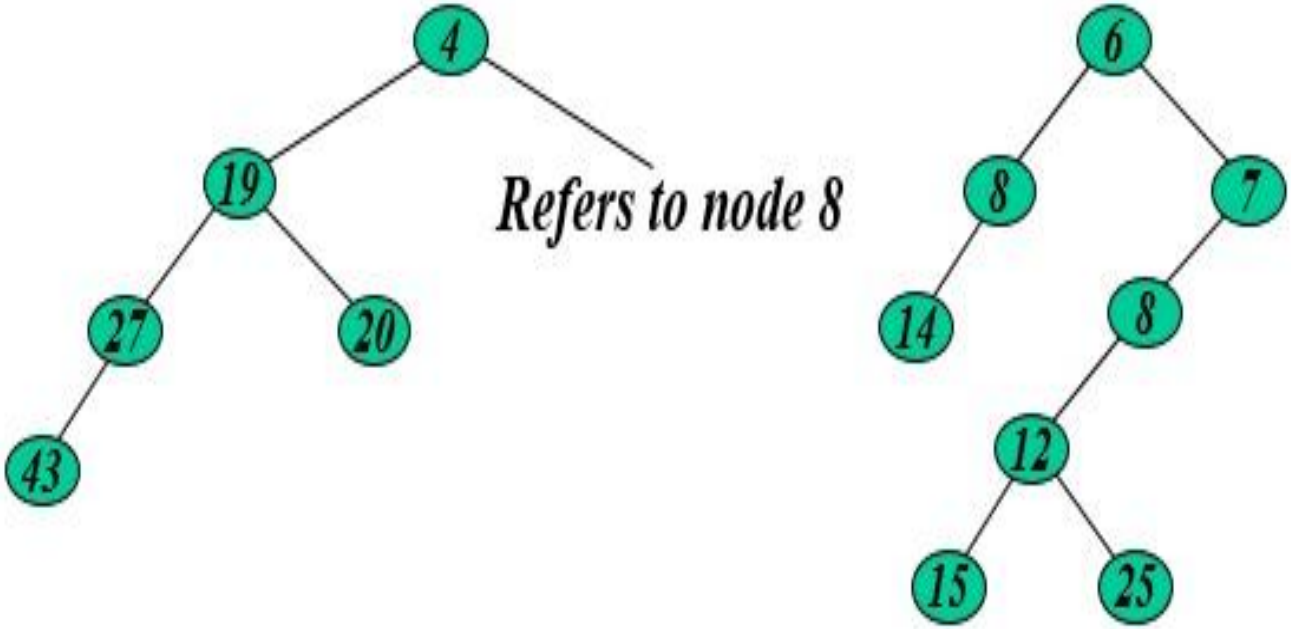


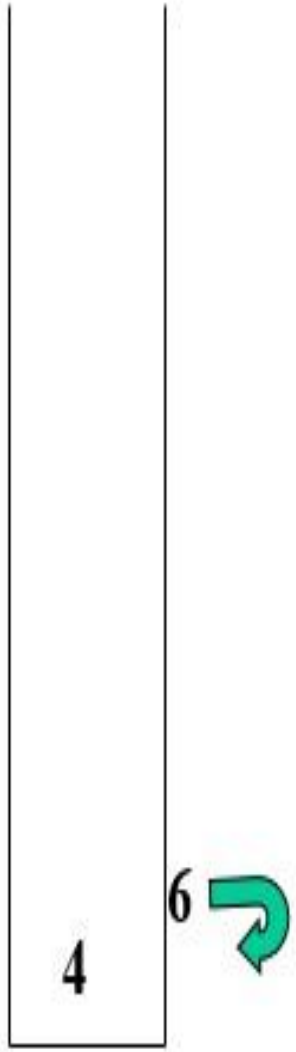


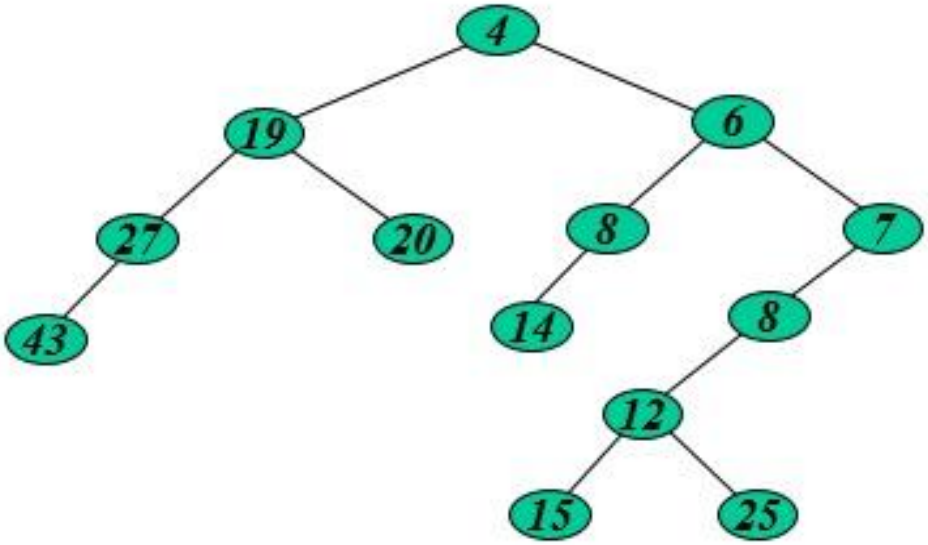

4
19
27
20
43
Refers to node 8

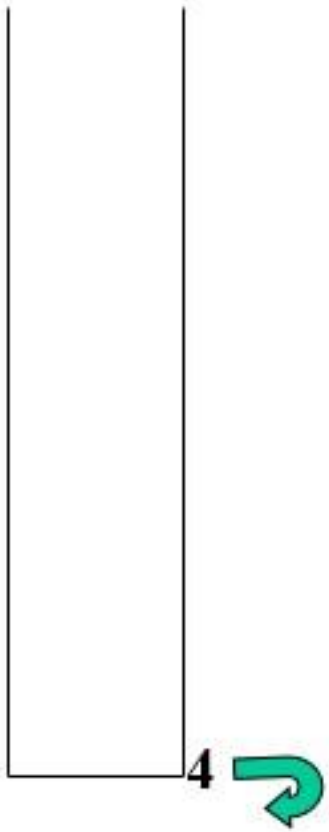

6
8
7
14
8
12
15
25


7
6
4

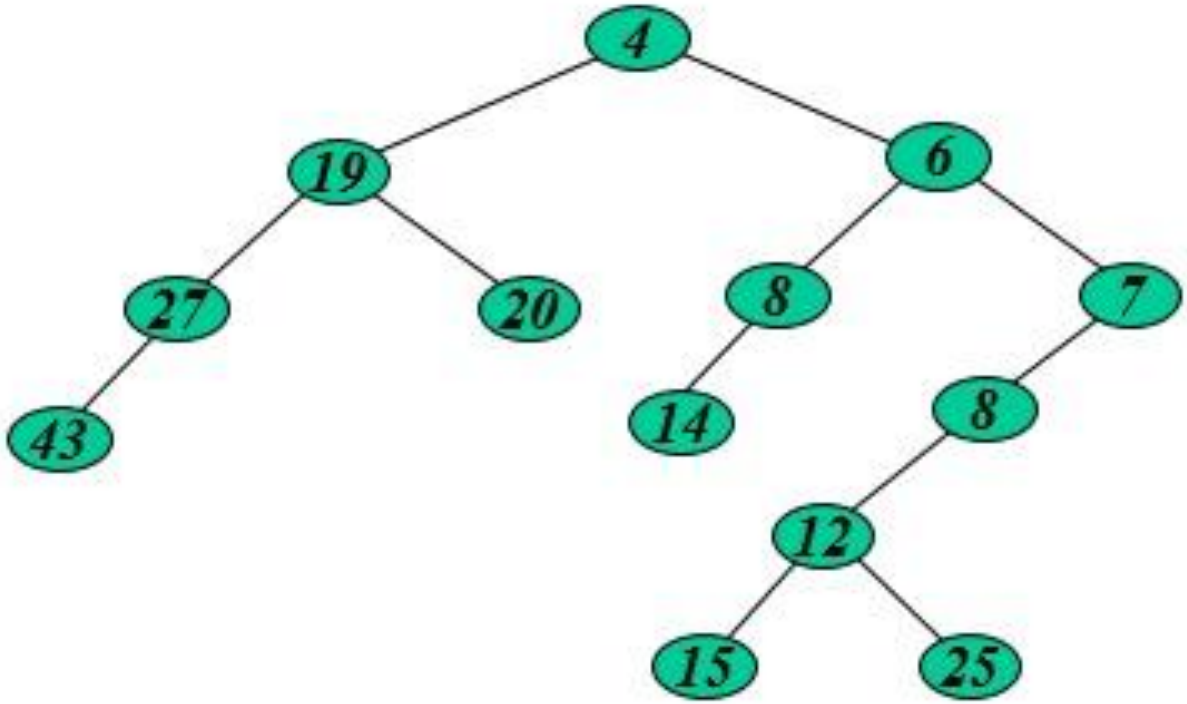

4
19
27
20
43
Refers to node 8
6
8
7
14
8
12
15
25


6
4


4
19
6
27
20
8
7
43
14
8
12
15
25


4

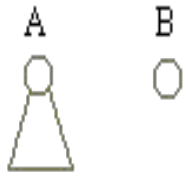
Et Voila!



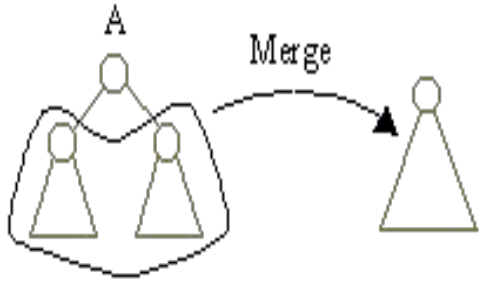
MERGE

- Compare the roots of two heaps
- push the smaller key into an empty stack, and move to the right child of smaller key
- recursively compare two keys and go on pushing the smaller key into the stack and move to its right child
- repeat until a null node is reached
- take the last node processed and make it the right child of the node at the top of the stack and convert it to leftist heap if the properties of leftist heap are violated
- recursively go on popping the elements from the stack and making them the right child of new stack top

```
Insert (x,A)
     B ← MakeIntoTree(x)
     A ← Merge(A,B)
```




```
DeleteMin(A)
     t ← data(root(A))
     A ← Merge(left(A), right(A))
     return t
```



```
Merge (A,B)
      if A = NULL return B
      if B = NULL return A
      if key(B) < key(A)
            swap A, B
      right(A) ← Merge(right(A),B)
      if dist(right(A)) > dist(left(A))
            swap right(A), left(A)
      if right(A) = NULL
            dist(A) ← 0
      else
            dist(A) ← 1 + dist(right(A))
      return A
```

Time Complexity

Queries

References

www.wikipedia.org

www.geeksforgeeks.org

www.dgp.toronto.edu

Thank You!