Design and Analysis of Algorithms

Lecture - 15

Success is always inevitable with Hard Work and Perseverance

N. Ravitha Rajalakshmi

Learning Objective

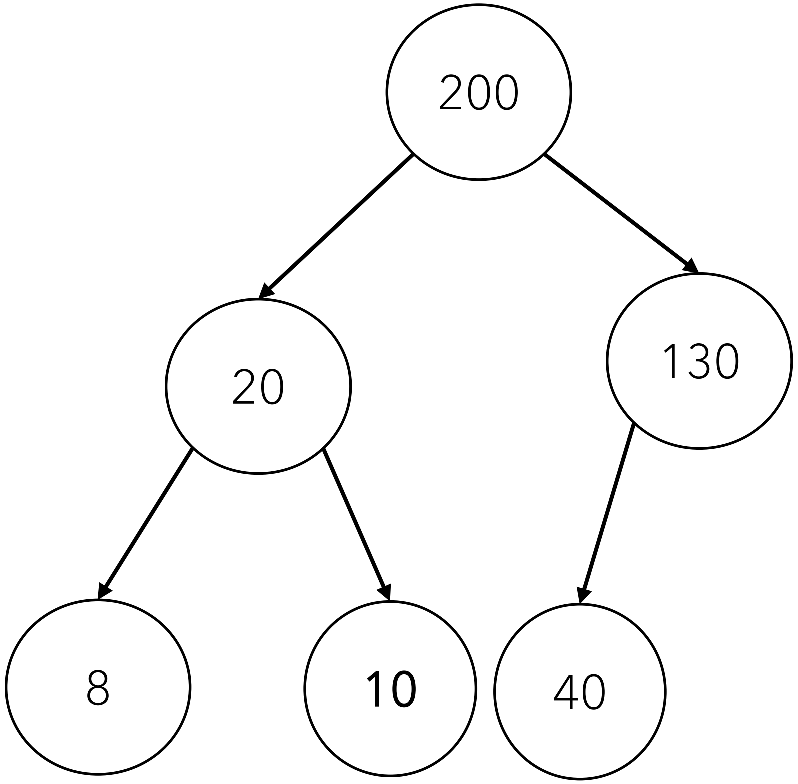
- Understand how Heap data structure can be used for sorting
  - Heap , Property and its Construction

Heap

- Array object visualized as a binary tree
- Nearly complete binary tree (all levels are filled except at the last level)
- Two import attributes of heap
  - Length - Length of the entire array
  - Heap size – Number of elements in the heap

  Heap size ≤ length

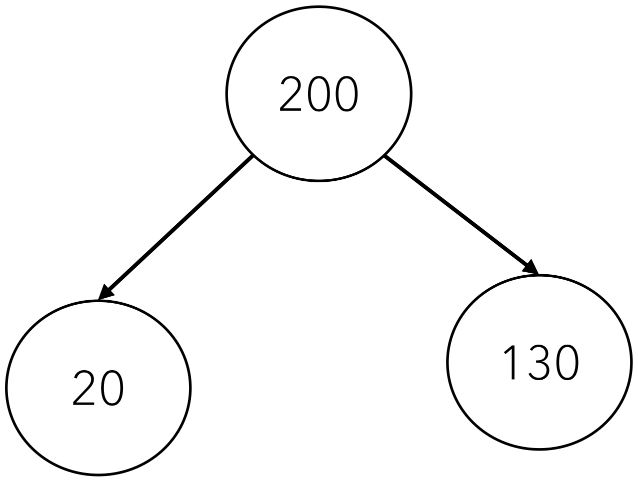
200	20	130	8	10	40



Heap property

- Values in the nodes should satisfy heap property
- Two kinds of heaps : min heap and max heap
- Max heap – Parent should hold larger value compared to its children

$$A[parent(\mathbf{i})] \geq A[\mathbf{i}]$$



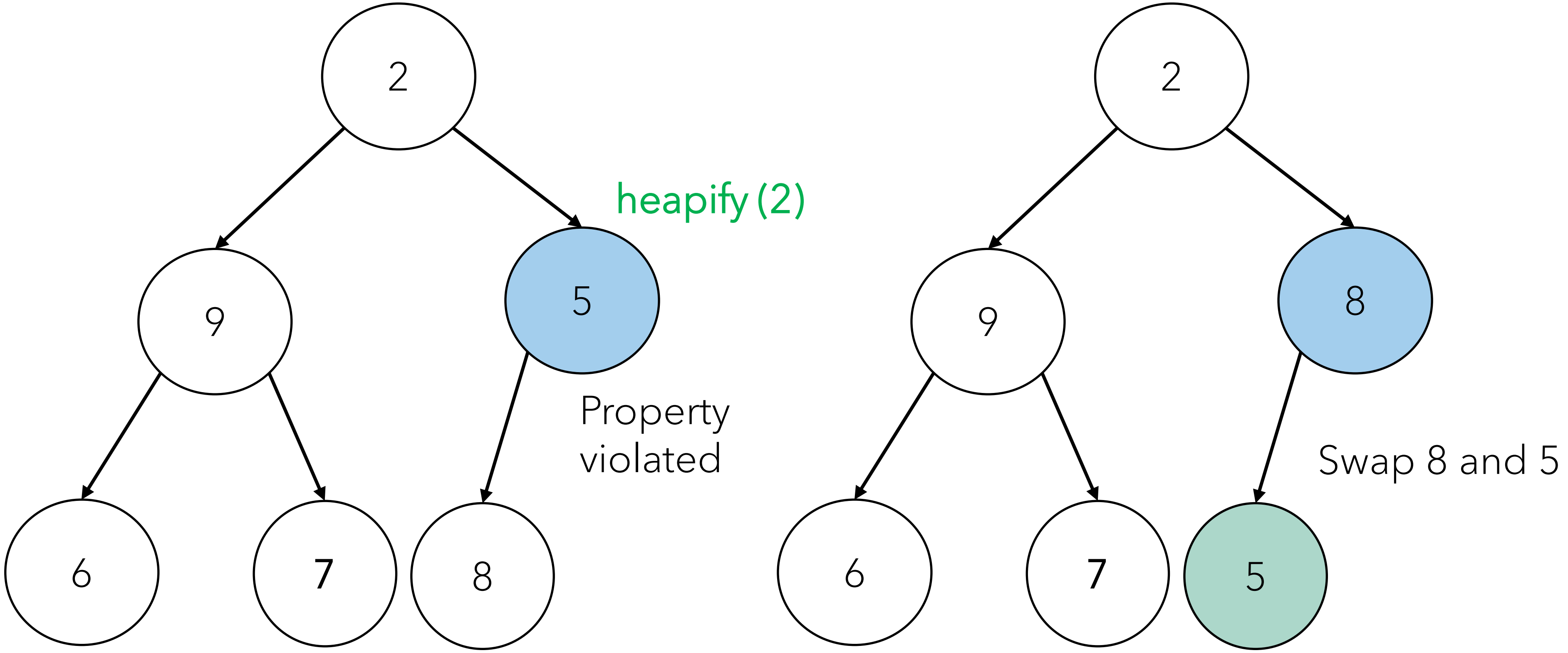
Build Max Heap

- Verify whether the property is satisfied for all nodes in the tree
- Top down vs Bottom up
- Bottom up heap construction
  - Verify the heap property starting from the last non-leaf node in a bottom up fashion
  - Apply heapify whenever the property is violated

Function Build-Max-Heap(A)

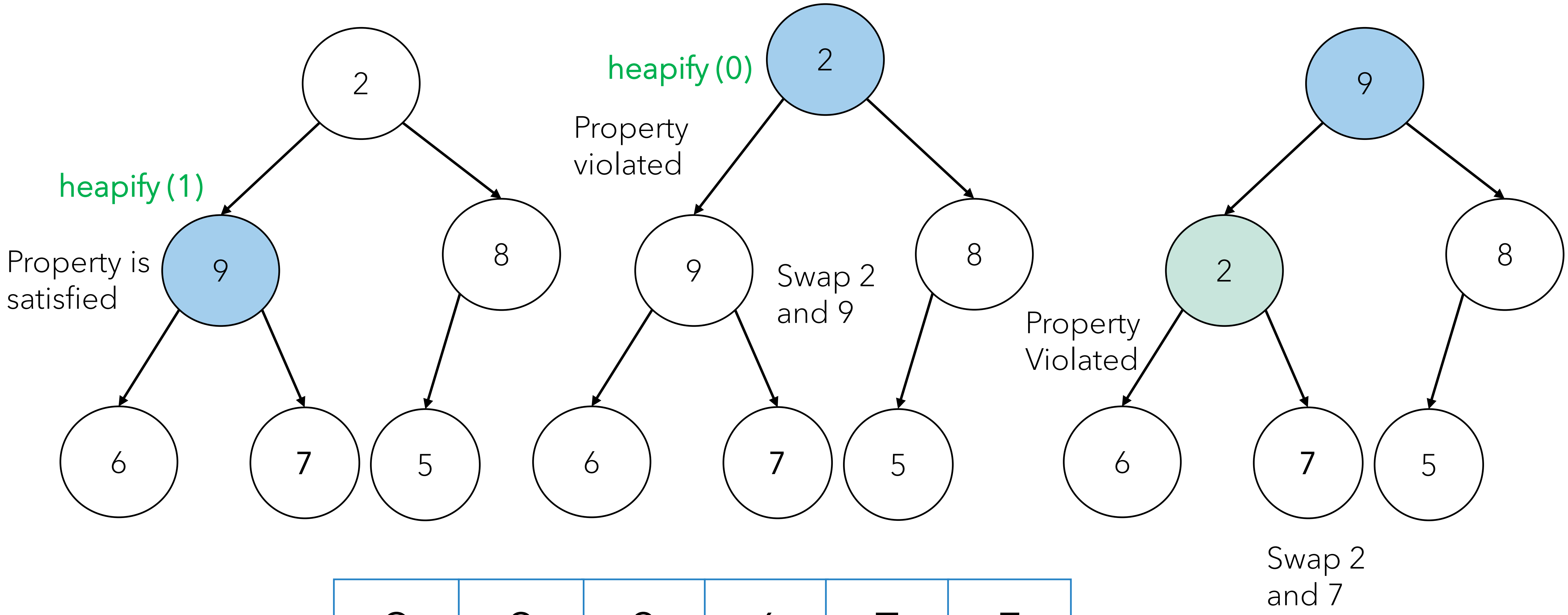
```
Heap_size = n  # all nodes are part of the heap
for i  in range(n/2 to 0)
        Max-Heapify(A, i)
```

Array Elements	2	9	5	6	7	8

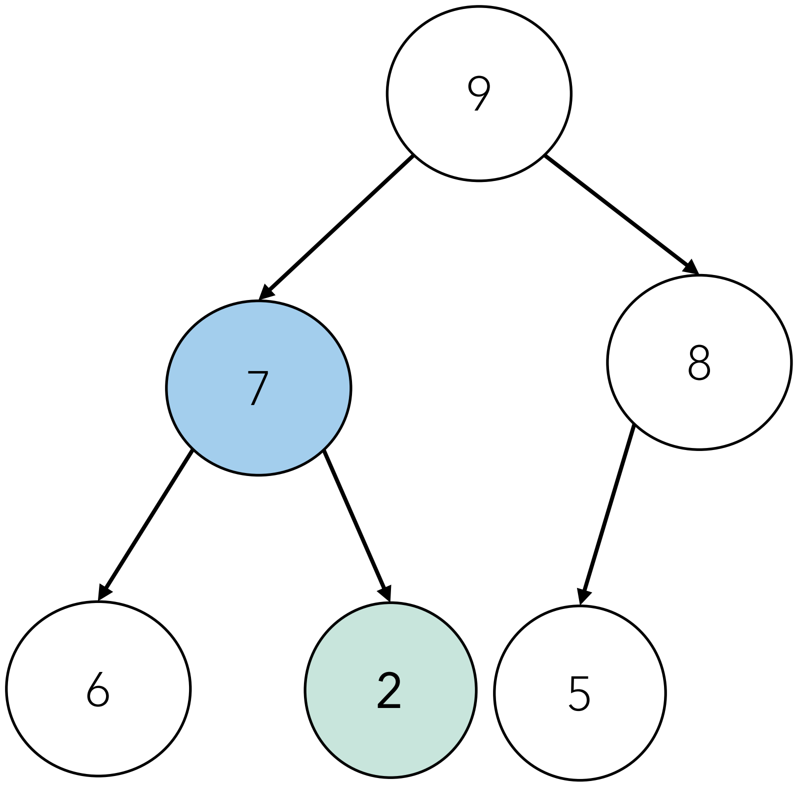


2	9	8	6	7	5

2	9	8	6	7	5



9	2	8	6	7	5



9	7	8	6	2	5

Heap Sort

Array Elements

130	10	40	8	20	200

- Transform and Conquer

    Change the representation of input data

200	20	130	8	10	40

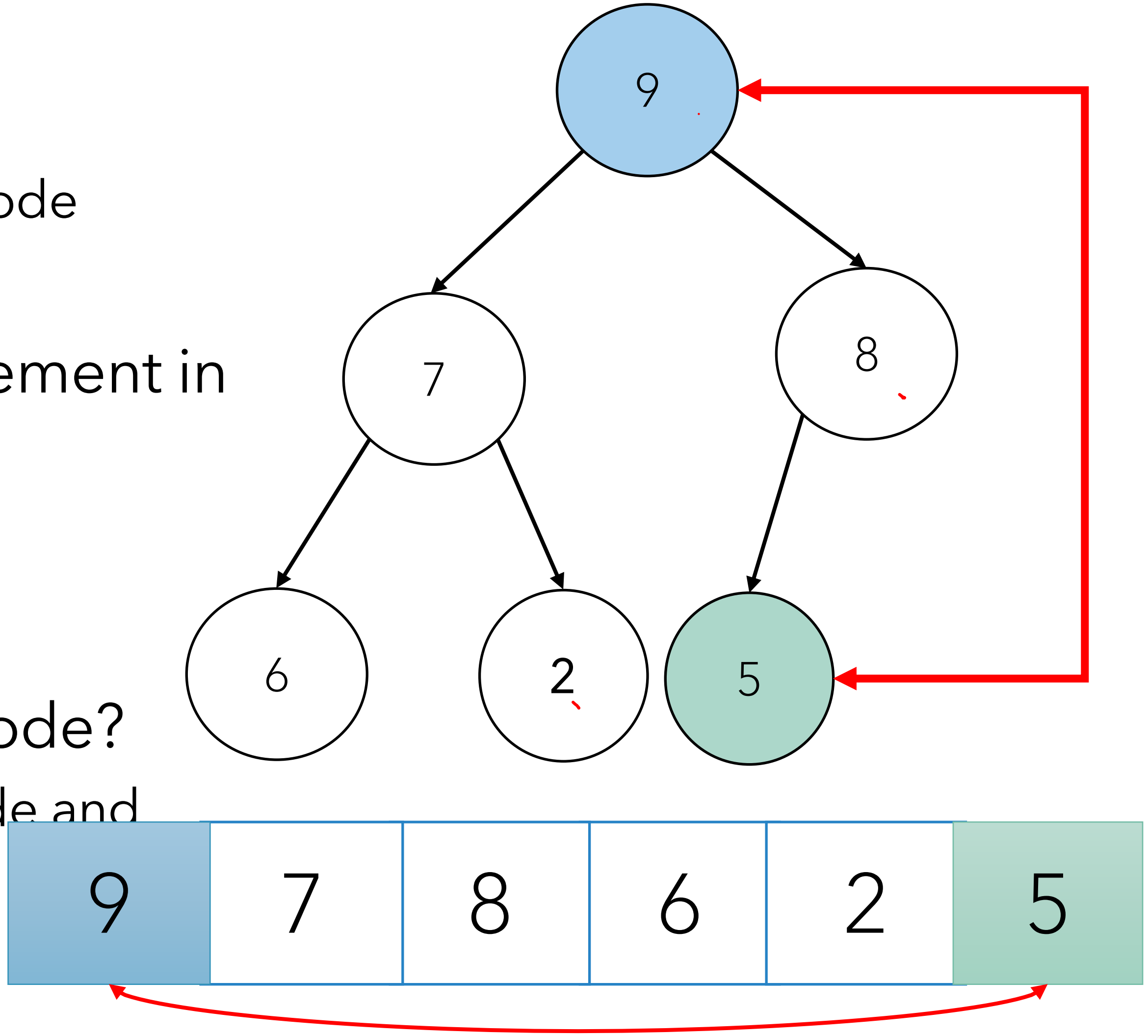
Heap

- Identify and remove Maximum element recursively to find the sorted order

8	10	20	40	130	200

Sort

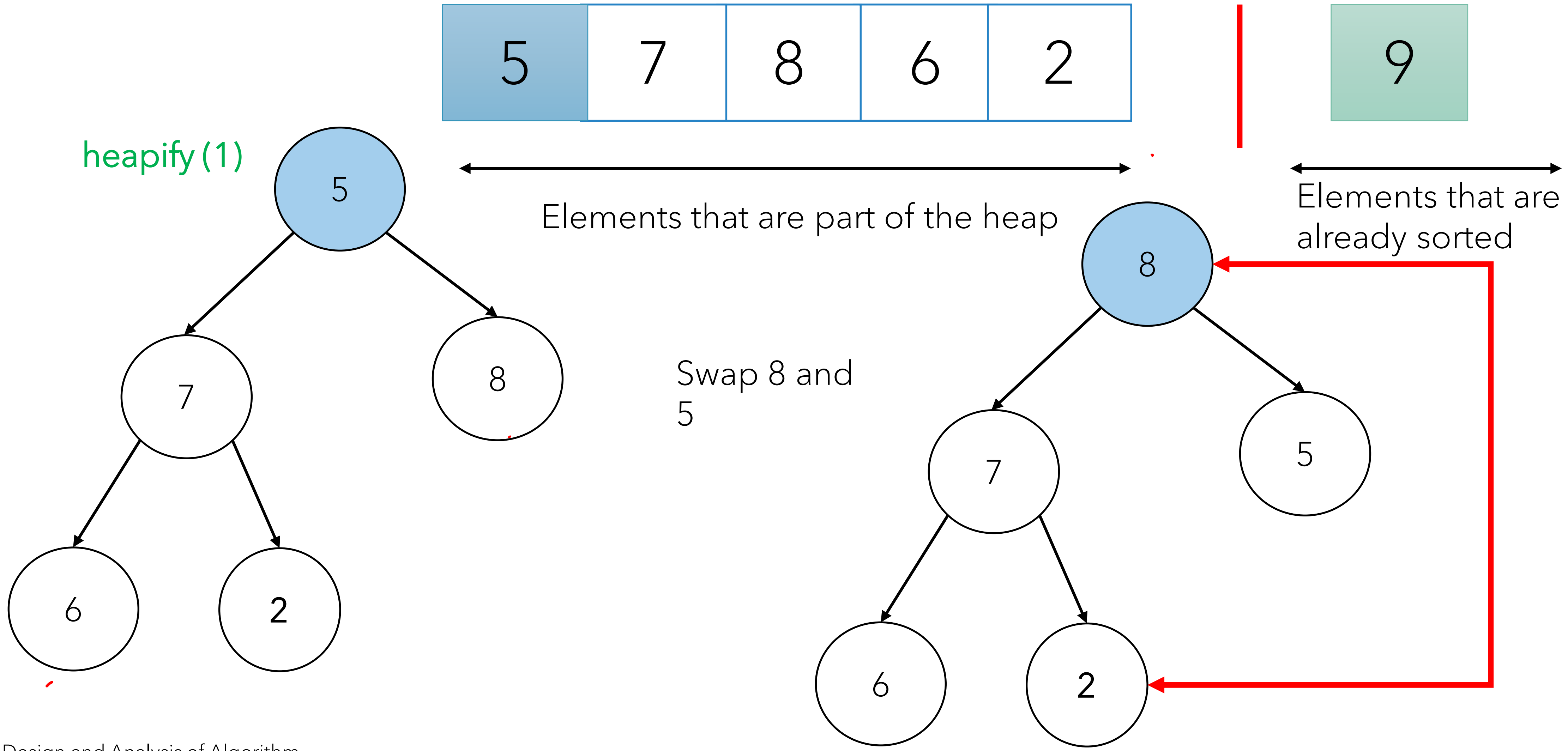
- How to sort elements using heap?
  - Max element will be present at the root node
- Which is the actual position of max element in sorted array?
  - Last element in the tree
- Can we swap last element with root node?
  - Heap property will be violated at root node and last element



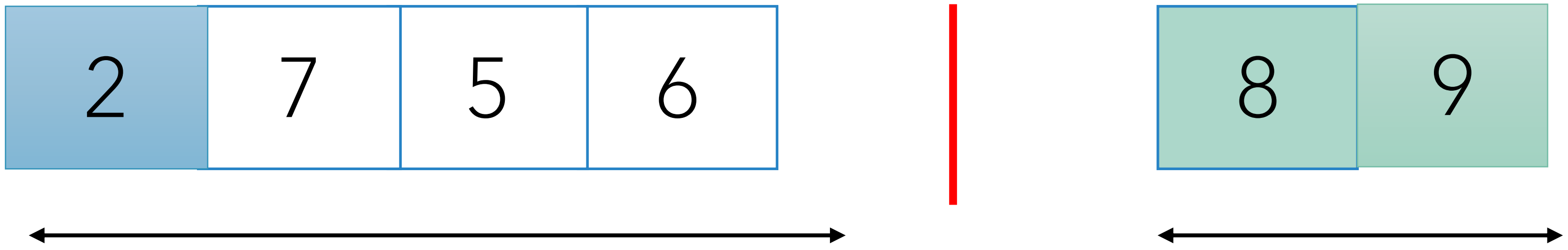
```
HeapSort (A) {
  1. Build-max-heap (A)

  2. for (i=0; i<n; i++) {
         Swap (A[0], A[heapsize-1])
         heapsize--;
         heapify (A, 0)
     }
}
```

Sort


5 7 8 6 2
9
heapify (1)
Elements that are part of the heap
Elements that are already sorted
Swap 8 and 5
5
7
8
6
2
8
7
5
6
2

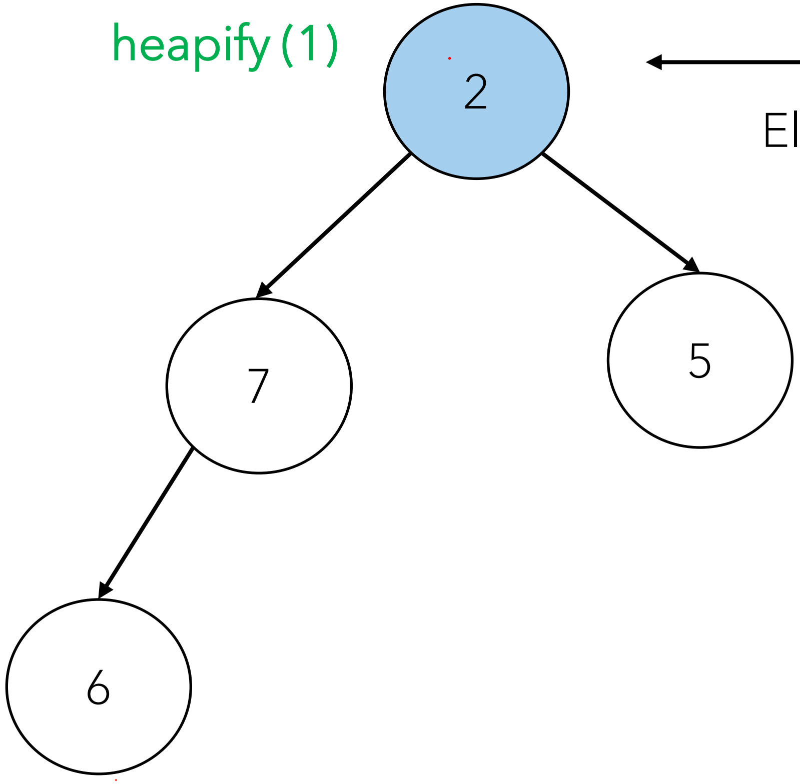
Sort



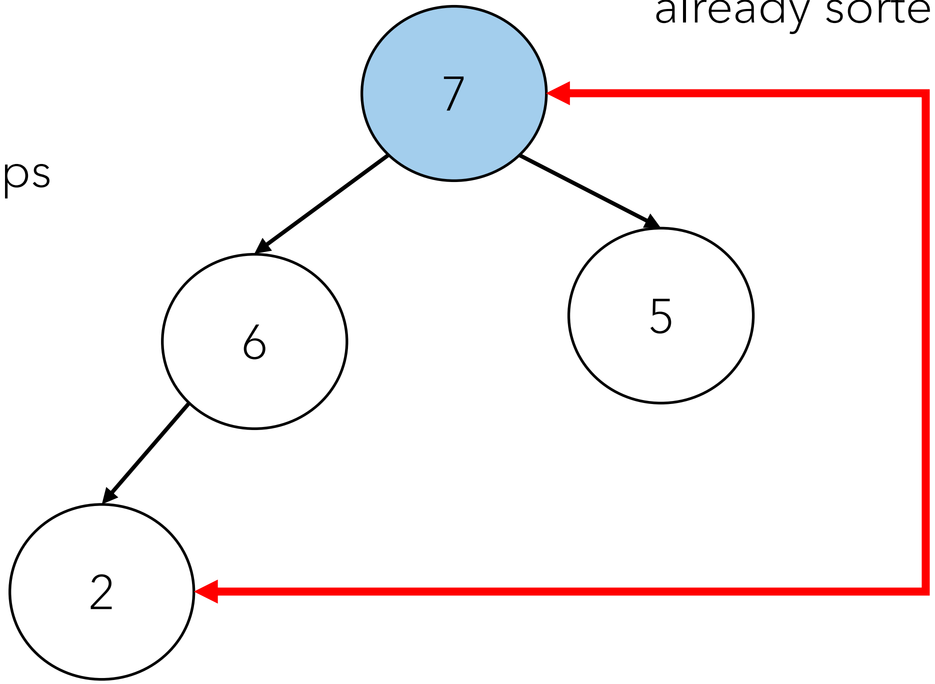
Elements that are part of the heap

Elements that are already sorted

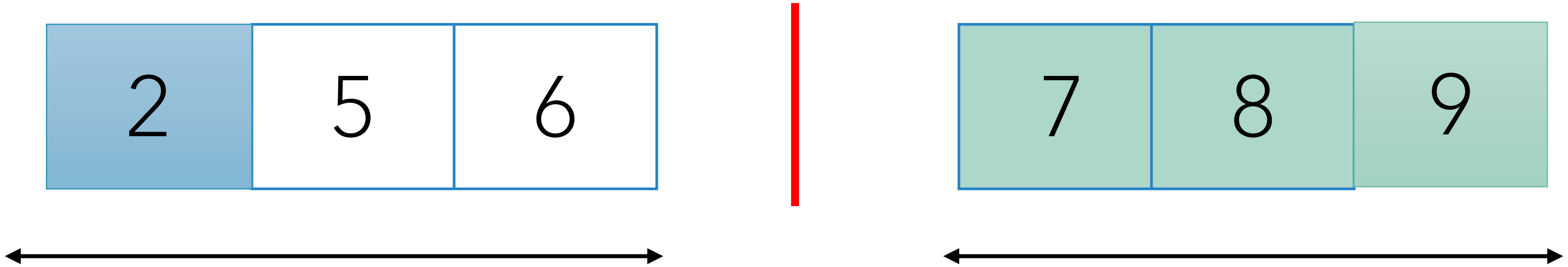
heapify (1)



Two swaps
2 and 7
2 and 6



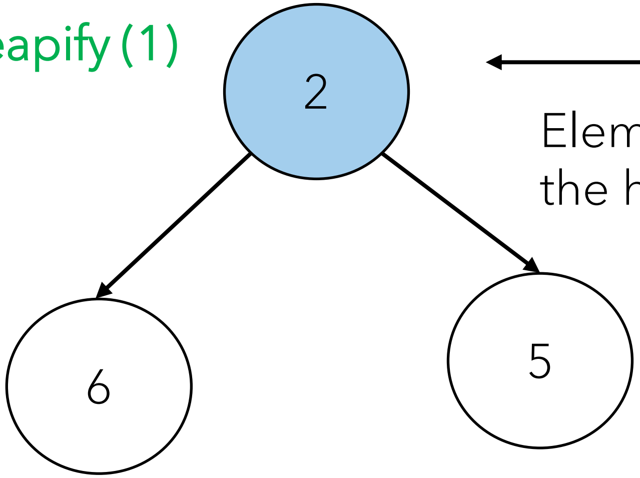
Sort



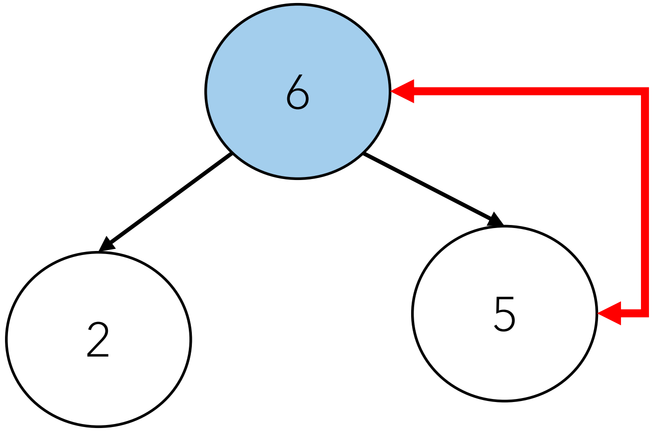
Elements that are part of the heap

Elements that are already sorted

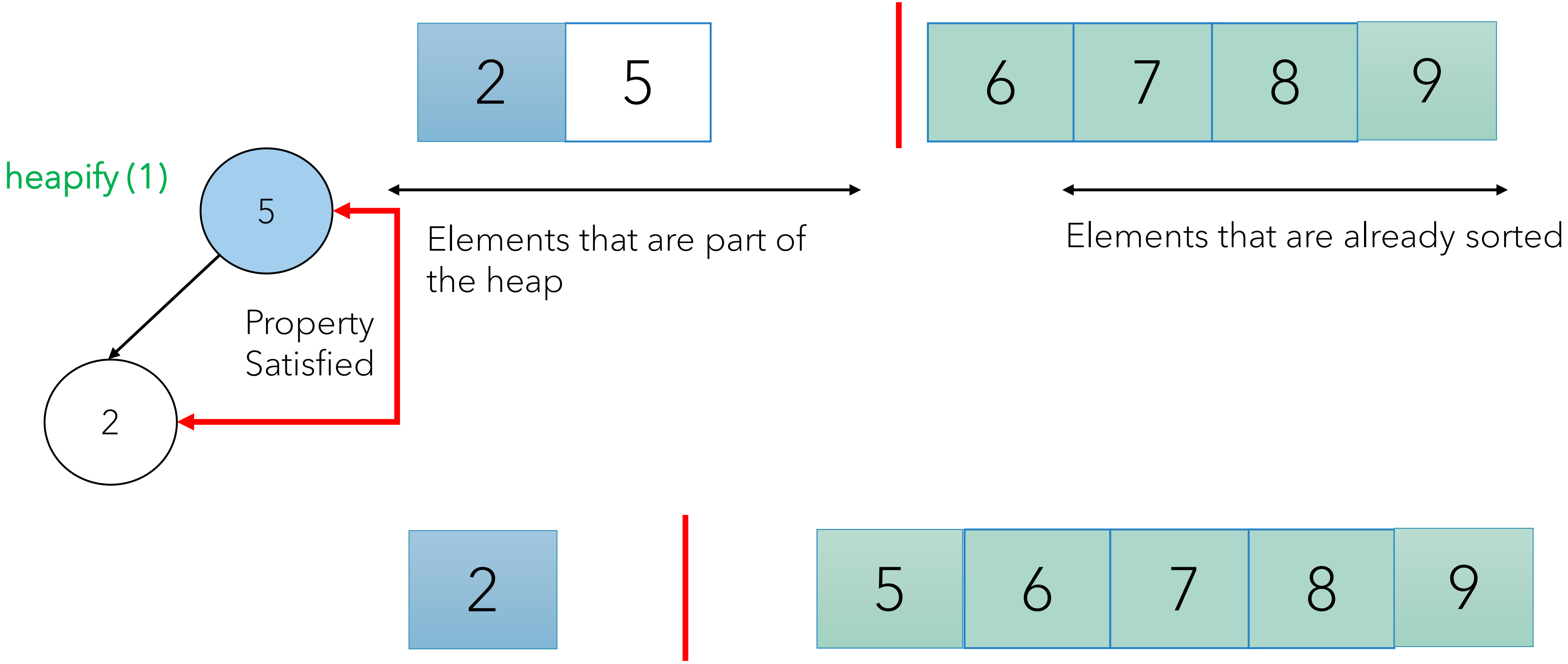
heapify (1)



Swap 6 and 2



Sort


heapify (1)
5
2
Property Satisfied
2
5
6
7
8
9
Elements that are part of the heap
Elements that are already sorted
2
5
6
7
8
9

Function Build-Max-Heap(A)

```
Heap_size = n
# all nodes are part of the heap
for i  in range(n/2 to 0)
        Max-Heapify(A, i)
```

Function Heap Sort (A)

```
Build-Max-Heap(A)
for(i=n-1 ;i>0;i++)
        swap(A[0],A[i])
        Heap_size = Heap_size - 1
        Max-Heapify(A, 0)
```

Function Max-Heapify(i)

```
l = Left(i)
r = right(i)
gt = i  # index of largest element
if(A[gt]<A[l] && l<heapsize)
        gt = l
if(A[gt]<A[r] && r<heapsize)
        gt = r
if(gt!=i){
        swap (A[i], A[gt])
        Max-Heapify(gt)
}
```

Time Complexity

Analysis of Max – Heapify:

It is dependent on the index i

Worst case scenario :

Input size : n

Basic Operation : number of comparisons

Property violated at every level in the tree

$$\boldsymbol{T(n) = T(2\,n/3) + \theta(1)}$$

By master's theorem , $T(n) = O(\log n)$

Analysis of Build-Max-Heap:

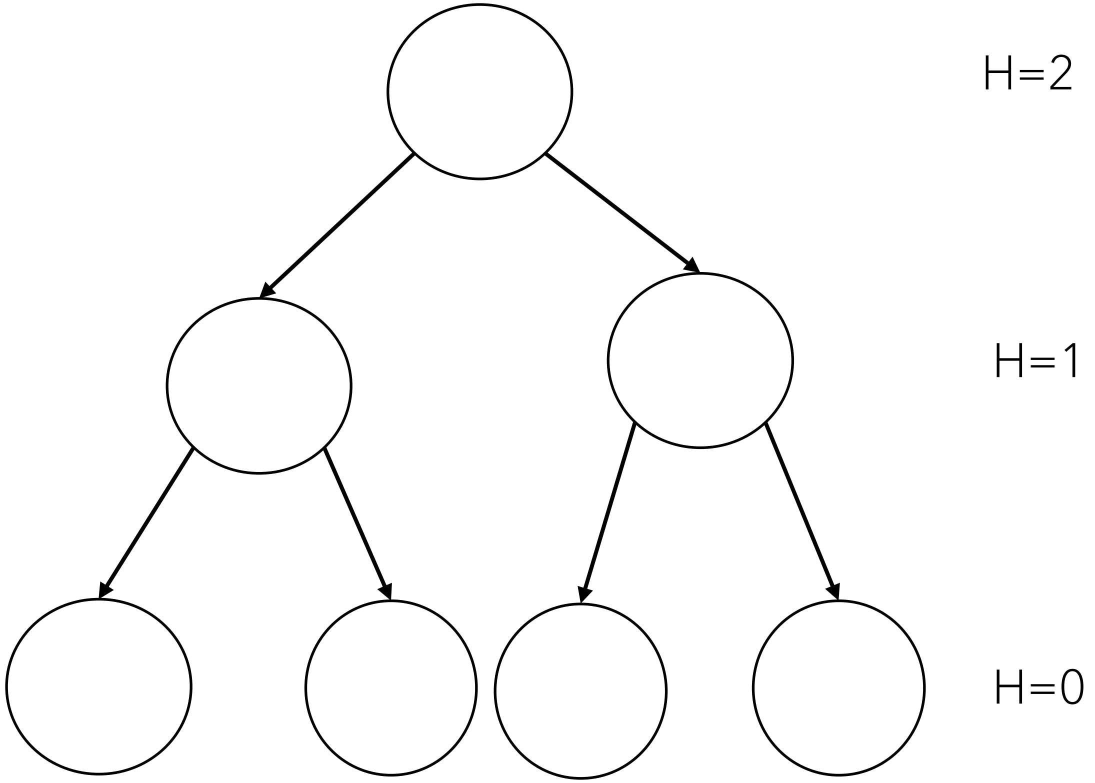
At any node number of comparisons depends on the height of the node

$$\sum_{h=0}^{\log n} \frac{n}{2^{h+1}}(h)$$

$$\sum_{h=0}^{\log n} \frac{n}{2^{h}}(h) = \mathrm{n} \sum_{h=0}^{\infty} \frac{h}{2^{h}}$$

$$= \frac{1/2}{(1-1/2)^2}(n)$$

$$= O(n)$$



Analysis of Heap Sort:

= Build Heap + n (Heapify)

$= O(n) + n \log n$

$= O(n \log n)$

Summary

- Discussed Heap Sort along with its time complexity

Thank You
Happy Learning

Success is always inevitable with Hard Work and Perseverance