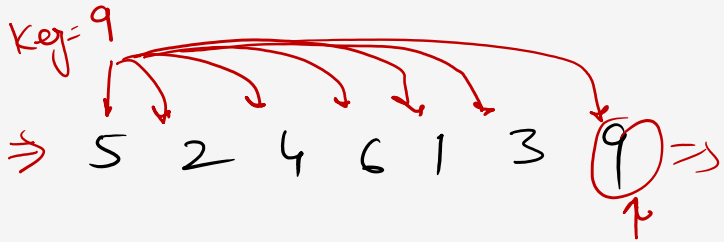
# Algorithm

## Today's Class Topics

**Searching and Sorting**

- Linear Search
- Bubble sort
- Selection Sort
- Insertion Sort
- Heap sort
- counting Sort
- Radix Sort

Linear Search :-




=> O(n)

Sorting :-

A | 5 | 2 | 4 | 6 | 1 | 3 |

Inplace sort :- [Extra space = O(1) or O(log n)] [Bubble sort, selection sort, Insertion sort, Heap sort etc]

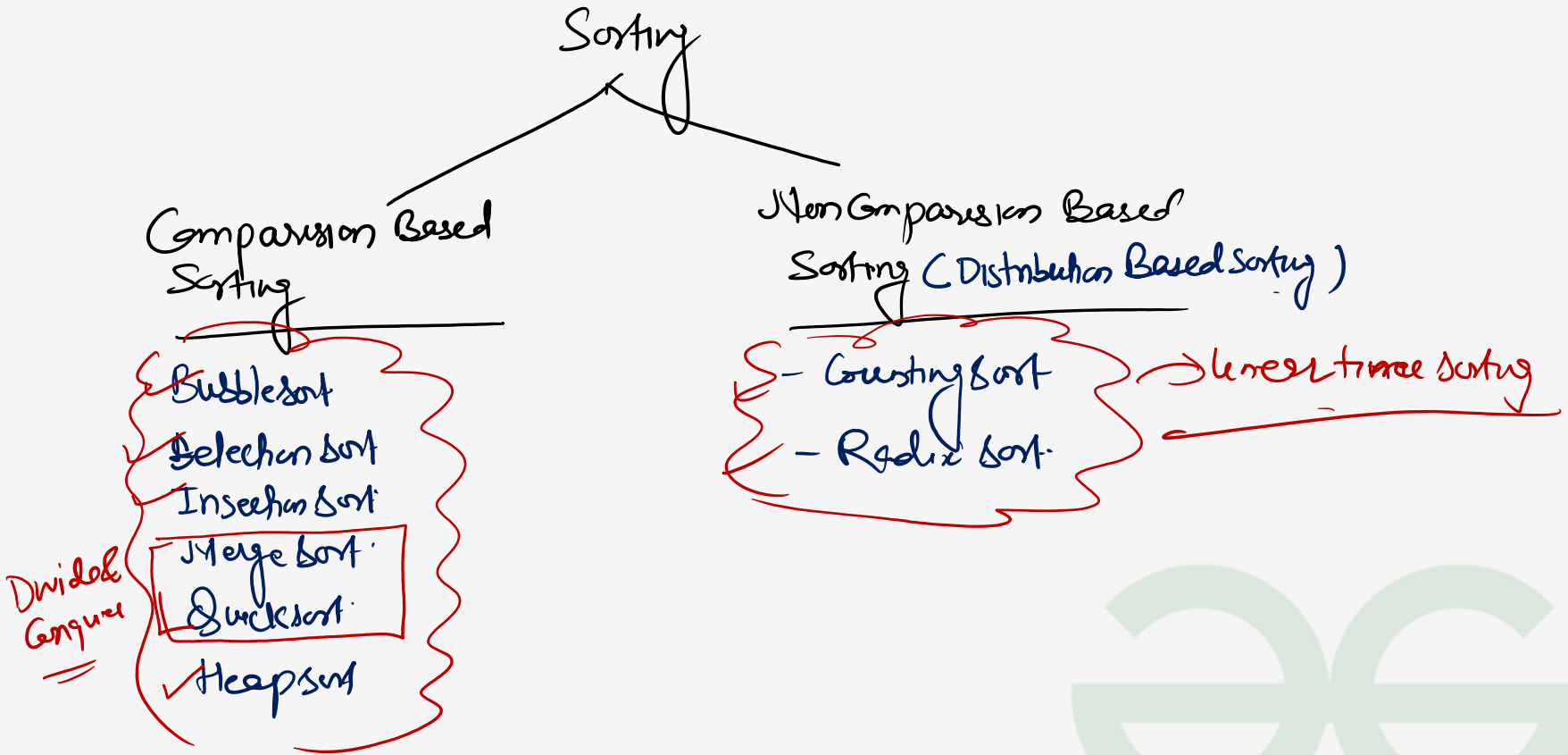
Stable sort :-

5 2 4 6 2 1 3

↓ After sorting

1 2 2 3 4 5 6

Ex: Bubble sort, Insertion sort, merge sort, Counting sort, Radix sort.


Sorting
Comparison Based Sorting
Non Comparison Based Sorting (Distribution Based sorting)
Bubble sort
Selection sort
Insertion sort
Merge sort
Quick sort
Heap sort
Divide & Conquer
Counting sort
Radix sort
Linear time sorting

# Bubble Sort:

5 2 4 6 1 3

No of elements sorted in Pass-i = i elements.

No of Comparisons in Pass i = (n-i)

**Pass-1:** (n-1) Comparisons

5 2 4 6 1 3 → swap → 2 5 4 6 1 3 → swap → 2 4 5 6 1 3 → 2 4 5 6 1 3 → swap → 2 4 5 1 6 3 → swap → 2 4 5 1 3 | 6

6 properPosition

**Pass-2** (n-2)

2 4 5 1 3 | 6 → 2 4 5 1 3 | 6 → 2 4 5 1 3 | 6 → swap → 2 4 1 5 3 | 6 → 2 4 1 3 | 5 6

**Pass (3)** (n-3)

2 4 1 3 | 5 6 → 2 4 1 3 | 5 6 → swap → 2 1 4 3 | 5 6 → 2 1 3 | 4 5 6

**Pass-4** (n-4)

2 1 3 | 4 5 6 → swap → 1 2 3 | 4 5 6 → 1 2 | 3 4 5 6

**Pass-5**

1 2 | 3 4 5 6 → 1 | 2 3 4 5 6

Total Cost = No of Comp + No of swap

Total Cost of Bubble sort = Pass(1) + Pass(2) + Pass(3) + --- Pass(n-1) ⇒ n-(n-1) ≠ 1

= (n-1) + (n-2) + (n-3) + --- 2 + 1 ⇒ $\frac{n(n+1)}{2}$

TC = $O(n^2)$

$= \boxed{6\ 5\ 4\ 3\ 2\ |} \Rightarrow$

Pass-1

6 5 4 3 2 1
5 6 4 3 2 1
5 4 6 3 2 1
5 4 3 6 2 1
5 4 3 2 6 1
5 4 3 2 1 | 6

$\Rightarrow 5+4+3+2+1$

$\Rightarrow$

**Total Swaps**

$BC = 0$

$WC = \frac{n(n+1)}{2} \Rightarrow \boxed{O(n^2)}$

Total Cost = No of Comp + No of swap

$= \frac{n(n+1)}{2} + \frac{n(n+1)}{2} \Rightarrow \boxed{O(n^2)}$

$\Rightarrow$

Question:

[redacted], 7, [redacted], 9 1 3 19 22 31

No of swap = ?

6+2+1+3+2

5 7 8, 9, 13 19 22, 31, _ ,

⇒ (14) Ans

3 + 5 + 2 + 1 + 2 + 1 = (14)

Algo: BubbleSort(A[], int n)

```
{
    int i, j, temp, flag = 1
    for(i = 0; i < n-1 && flag; i++)
    {
        flag = 0
        for(j = 0; j < n-1-i; j++)
        {
            if(A[j] > A[j+1])
            {
                flag = 1;
                temp = A[j];
                A[j] = A[j+1];
                A[j+1] = temp;
            }
        }
    }
```

SC = I/p + ~~Recursion~~
= Constant
⇒ O(1)

BC
WC
AC

⇒ Avg = $\frac{\text{TC}(1+2+3+4+\cdots+n)}{n}$ ⇒ O(n²)

flag = 0

A: 1 2 3 4 5 6 ⇒ 6 5 4 3 2 1

⇒ O(n²) ⇒ O(n)

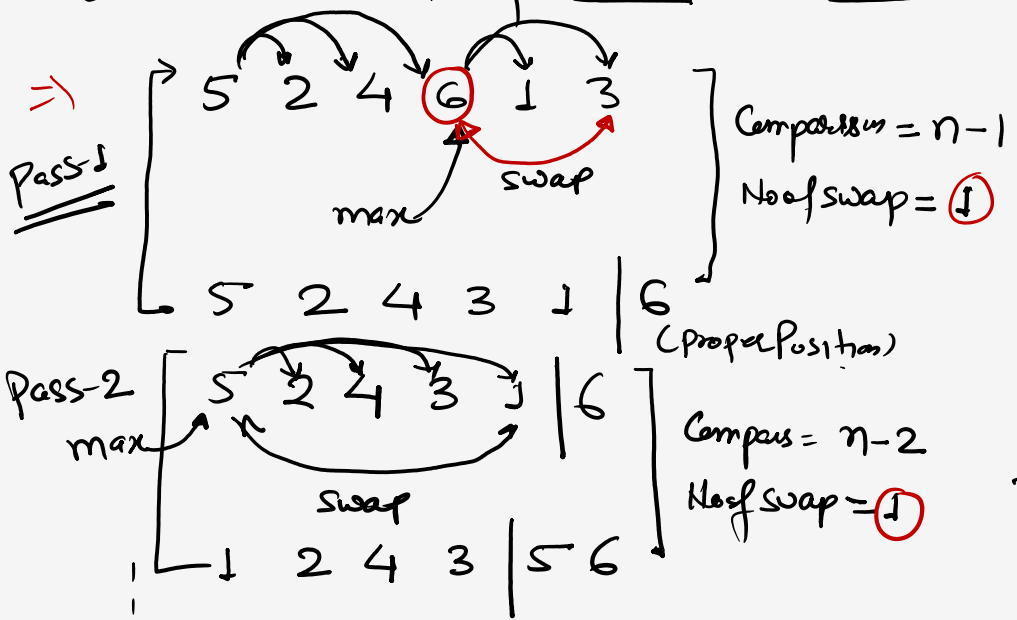
BC = O(n) ✓
WC = O(n²) ✓

AC = O(n²)

Ae:

1 2 3 4 5 → O(n)
2 1 3 4 5 → O(2n)
3 2 1 4 5 → O(3n)
4 3 2 1 5 → O(4n)
5 4 3 2 1 → O(5n)
⋮
O(n·n)
⇒ O(n²)

**Selection sort:-** Concept of maxima or minima element




mini No of swap in worst case = Selection sort = $(n-1)$

TC:

Total swap = $n-1 \Rightarrow O(n)$ — BC, WC, Ac.

Total Comp = Pass1 + Pass2 + Pass3 + ... Pass(n-1)

$= (n-1)+(n-2)+(n-3)+\cdots+2+1$

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

Total Cost = No of comp + No of swap

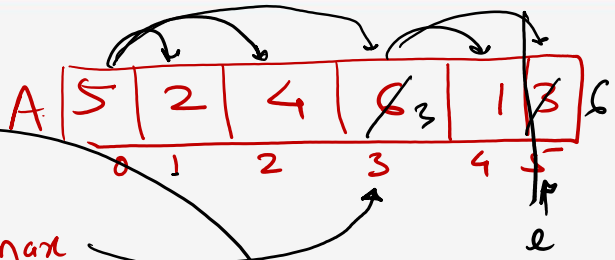
$= O(n^2) + O(n)$

$\Rightarrow O(n^2)$ ✓

Algo: Selectionsort(int A[], int n)

```
{ int i, j, max; flag=0
=> for(i = n-1; i>0; i--)
  { max = 0; flag=0
    for(j=1; j<=i; j++)
      { if (A[j] > A[max])
          max = j;
      }  flag=1
    => int temp = A[max];
       A[max] = A[i];
       A[i] = temp;
  }
}
```

flag:




temp = 6
A[3] = 3
A[5] = 6

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|

d

No of swap ⇒ (n-1) ⇒ O(n) ← BC, WC, AC

No of Comp = $O(n^2)$ ← BC, WC, AC

Total Cost = $O(n^2)$ ← BC, WC, AC.

## Insertion Sort:-

Algo: Insertion sort[A]

1- for J ← 2 to length[A] → n

2- do key ← A[J]

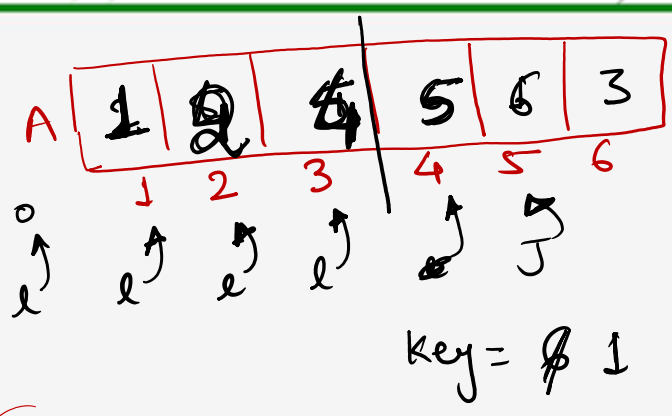
3- i = J-1

TRUE 4- → while(i>0) and (A[i] > key) + =>I

Inversion 5- do A[i+1] ← A[i]

6- i ← i-1

7- A[i+1] ← key




SC = O(1)

SC = I/p + Recursive stack dep?

= Constant

Total Cost = Total Comparision + Total Shifting

Worst Case (J)

Descending to Ascending

| | | | | |
|---|---|---|---|---|
| 2 | = | 1 | + | 1 |
| 3 | = | 2 | + | 2 |
| 4 | = | 3 | + | 2 |
| ⋮ | | ⋮ | | ⋮ |
| n | = | (n-1) | + | (n-1) |

Total = $\frac{n(n-1)}{2} + \frac{n(n-1)}{2}$

$= n(n-1)$

WC

TC = $O(n^2)$

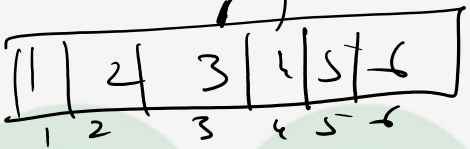
BC: $O(n)$

WC / AC: $O(n^2)$

Total Cost = Total Comp + Total Shift

BC

| | | | | |
|---|---|---|---|---|
| Array already Sorted | 2 | = | 1 | + 0 |
| | 3 | = | 1 | + 0 |
| | 4 | = | 1 | + 0 |
| | ⋮ | | ⋮ | |
| | n | = | 1 | + 0 |
| | | | (n-1) | + 0 |




BC

TC = $O(n)$

⇒ Ans

O(n)


TC:
BC = O(n)
WC = AC = O(n²)
SC = O(1)
BC
WC
AC

Insertion Sort with Binary Search & Linked List :-

| | Comparisons | Shifting | Total Cost |
|---|---|---|---|
| Binary Search | O(log n) | O(n) | O(n log n) + O(n²) = O(n²) |
| Linked list | O(n) | O(1) | O(n²) + O(n) = O(n²) |

**Inversion :-** Let $A[1, 2, \ldots n]$ be an Array of $n$-distinct numbers if $i < j$ and $A[i] > A[j]$ then the pair $(i, j)$ is called an inversion of A

**Ex:** A | 34 | 8 | 64 | 51 | 32 | 21 | And out the no. of inversion pairs

1 2 3 4 5 6

(34,8) (34,32) (34,21) (64,51) (64,32) (64,21) (51,32) (51,21) (32,21)

**Ex:** A | 2 | 4 | 1 | 3 | 5 | No of Inversion Pair

1 2 3 4 5

(2,1) (4,1) (4,3) ⇒ (3)

**Max. No. of Inversion :-** Array in Descending order

6 5 4 3 2 1

$n, n-1, n-2, n-3, \ldots 2, 1$

Total No of pairs $(n-1)+(n-2)+(n-3)+(n-4) \ldots +1+0$

WC $\Rightarrow \frac{n(n-1)}{2} \Rightarrow O(n^2)$

**Min No of Inversion :-** Array in Ascending order

1 2 3 4 5 6 ... $n-1$ $n$

Total No of Inversion $0+0+0+0+0+0 \ldots 0+0 = 0$

BC = 0 $\Rightarrow O(1)$

Avg No of Inversions : $\frac{BC+WC}{2} \Rightarrow \frac{0+\frac{n(n-1)}{2}}{2} \Rightarrow \frac{n(n-1)}{4} \Rightarrow O(n^2)$

TC of Insertion sort = $O(n + I)$

$TC \propto I$

$n$ → no of elements

$I$ → No of Inversion Pairs

BC: $I = 0 \Rightarrow TC = O(n+0) = O(n)$

WC: $I = \frac{n(n-1)}{2} \Rightarrow TC = O(n + \frac{n(n-1)}{2}) \Rightarrow O(n^2)$

Ac: $I = \frac{n(n-1)}{4} \Rightarrow TC = O(n + \frac{n(n-1)}{4}) \Rightarrow O(n^2)$

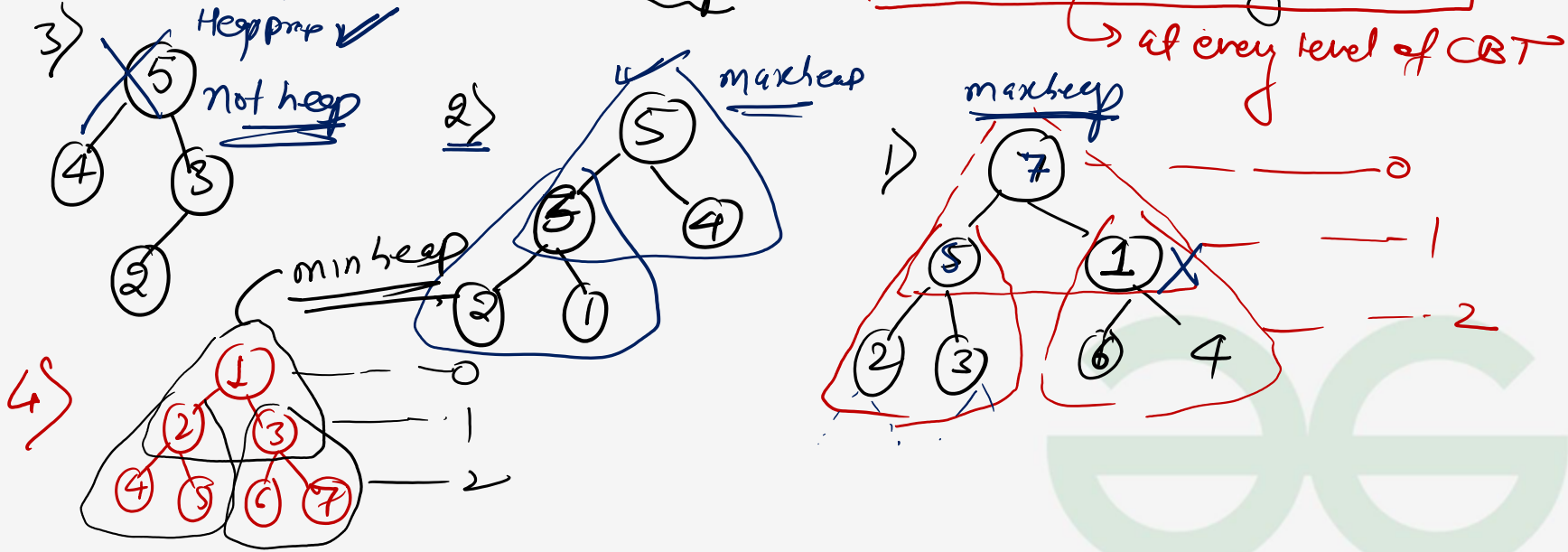
$\Rightarrow$ Break $\Rightarrow$ 10 mins

11:22

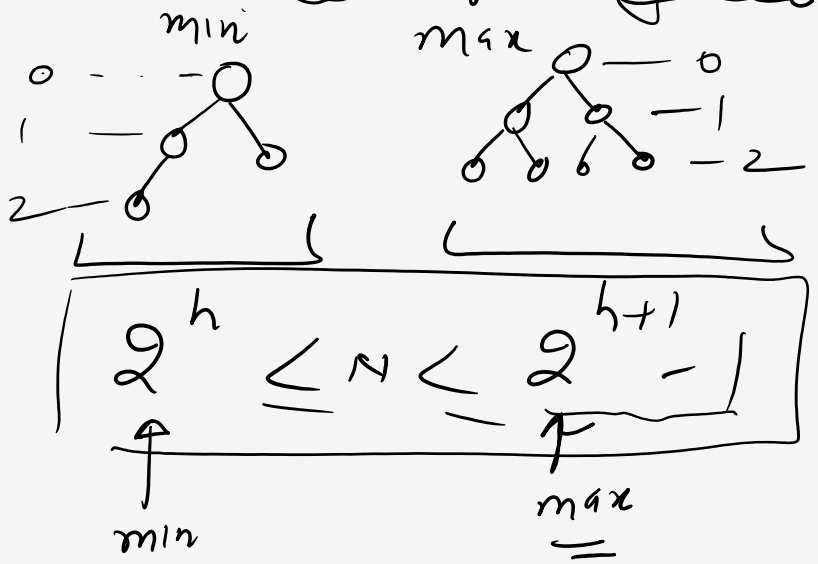
Heap :- CBT with Heap property.

CBT AND heap property

- max heap :- Parent ≥ Left & Right child (at every level of CBT)
- min heap :- Parent ≤ Left & Right child (at every level of CBT)

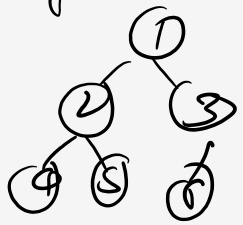


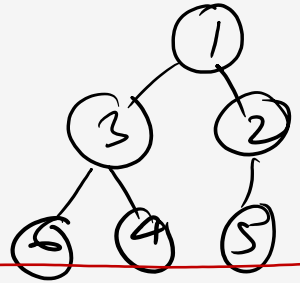
## min & max elements in heap on height (h)



1) Ascending order → Always min heap

1 2 3 4 5 6

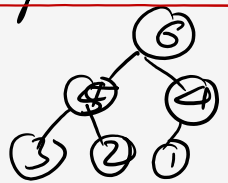


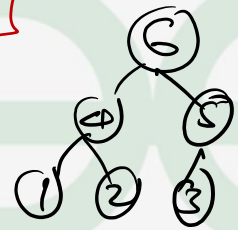




1 3 2 6 4 5

2) Descending order → Always max heap

6 5 4 3 2 1



## Representation of Heap:

int Heap[10];

Heap

| 10 | 14 | 10 | 8 | 7 | 9 | 3 | 2 | 4 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

Array:

=> Algo: Parent(i) → index

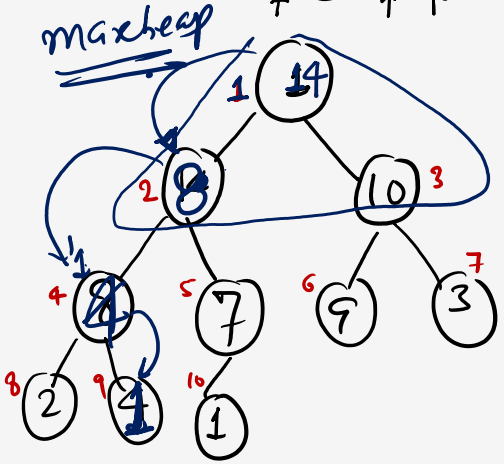
- return $\lfloor i/2 \rfloor$

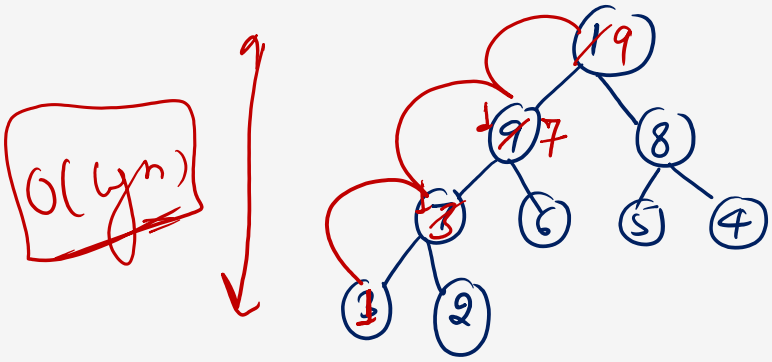
Left(i)

- return $2i$

Right(i)

- return $2i+1$




O(log n)
19
9
7
8
3
6
5
4
1
2

**Heapify:-** To maintain Heap property (max heap)

Heapsize[A] = 9

Algo:- Heapify (A, i)

1 - $l \leftarrow$ Left(i) // 2i // 2×1 = 2

2 - $r \leftarrow$ Right(i) // 2i+1 // 2×1+1 = 3

3 - if $l \leq$ heapsize[A] and A[l] > A[i]

4 - largest $\leftarrow l$

5 else largest $\leftarrow i$

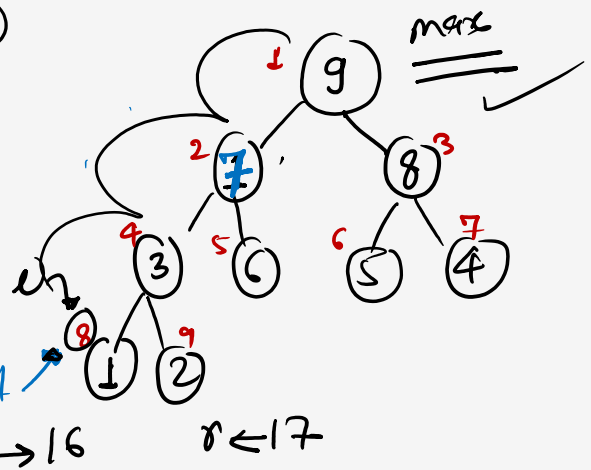
6 if $r \leq$ heapsize[A] and A[r] > A[largest]

7 - largest $\leftarrow r$

8 - if largest $\neq i$

9 - [ then Exchange A[i] $\leftrightarrow$ A[largest]

10 - Heapify (A, largest);




A

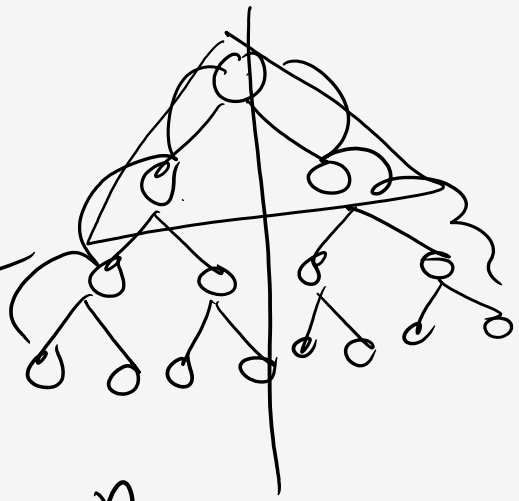
| 1 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

A

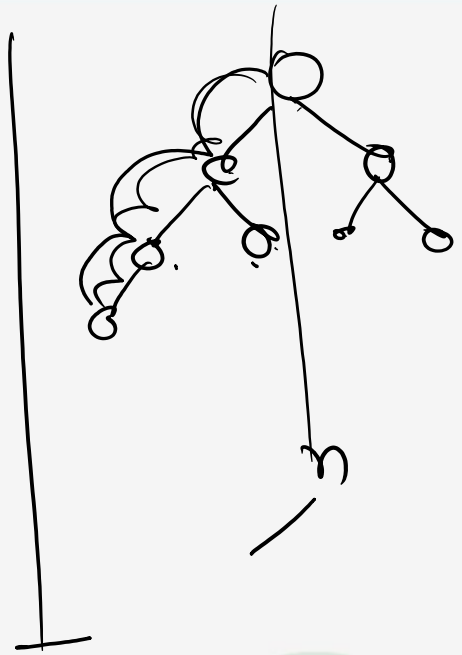
| 9 | 7 | 8 | 3 | 6 | 5 | 4 | 1 | 2 |
|---|---|---|---|---|---|---|---|---|

$T(n) = T(2n/3) + 1$

$\Rightarrow O(\log_{3/2} n)$



$\Rightarrow$



$n = 8$

$\frac{2n}{3} + \frac{n}{3}$

$\frac{2 \times 8}{3} + \left\lceil \frac{8}{3} \right\rceil$


$\Rightarrow$

$T(n) = 1 + T(n/2)$

$T(n) = T(n/2) + 1 \Rightarrow O(\log n)$

# Thank You !