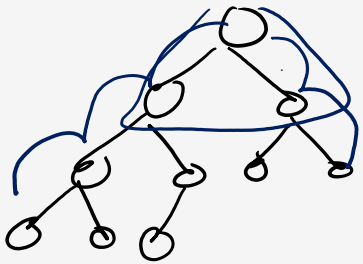
Algorithm

Today's Class Topics

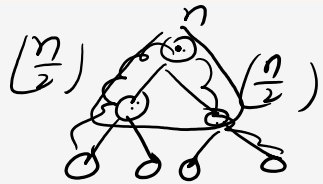
Divide and Conquer

-Binary Search
-Quick Sort
-Merge Sort
-Finding Maxima and Minima
-Selection Procedure
-Power of an element
-Strassen's Matrix Multiplication

Heapify:



$n = 10$



$T(n) = T\left(\left\lfloor \frac{2n}{3} \right\rfloor\right) + T\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + 1$

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

$= O(\log_2 n)$

$T(n) = T(\lfloor 2n/3 \rfloor) + 1$

$T = O(\log n)$

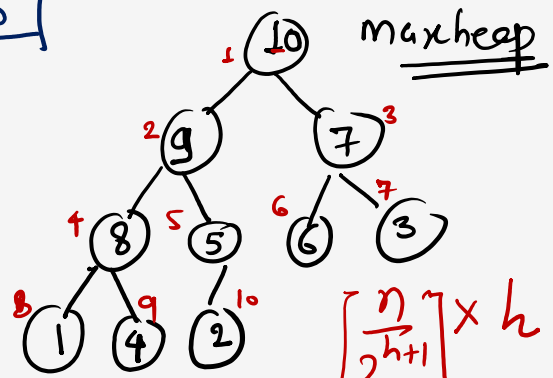
$\Rightarrow T(n) \leq T(2n/3) + 1 \Rightarrow O(\log_{3/2} n) \Rightarrow O(\log n)$

Max:

Build Heap:

A	1	2	3	4	5	6	7	8	9	10

A[6 — 10]



$\Rightarrow$ leafnode = $A[\lfloor n/2 \rfloor + 1 \text{ --- } n]$

Internalnode = $A[1 \text{ --- } \lfloor \frac{n}{2} \rfloor]$

$= A[1 \text{ --- } 5]$

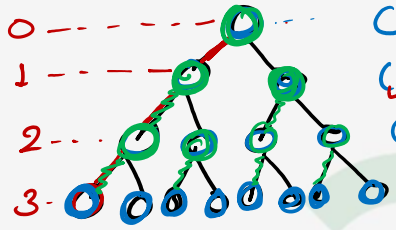
$\left\lceil \frac{n}{2^{h+1}} \right\rceil \times h$

Algo: BuildHeap(A)

1 - Heapsize[A] ← Length[A]

2 - for i ← $\lfloor$ Length[A]/2 $\rfloor$ downto 1

3 - Heapify(A, i)



(1,3) = 1 × 3

(2,2) = 2 × 2

(4,1) = 4 × 1

(8,0)

11

No. of Nodes at Height h = $\left\lceil \frac{n}{2^{h+1}} \right\rceil = \left\lceil \frac{15}{2^{h+1}} \right\rceil \Rightarrow \left\lceil \frac{15}{8} \right\rceil$

$\left\lceil \frac{15}{2^{3+1}} \right\rceil \Rightarrow \left\lceil \frac{15}{16} \right\rceil \Rightarrow 1$

1+1+1+1 +2+2+3

4×1 + 2×2 + 3

$$= \sum_{h=0}^{\log_2 n} \left\lceil \frac{n}{2^{h+1}} \right\rceil \times h = \frac{n}{2} \times \sum_{h=0}^{\log_2 h} \left(\frac{h}{2^h} \right)$$

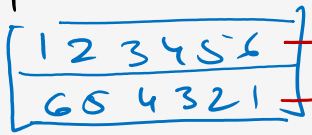
$$\frac{n}{2} \times \left[\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \cdots \right]$$

$$\Rightarrow \frac{n}{2} \times C \Rightarrow \frac{C}{2} \times n \Rightarrow \boxed{O(n)}$$

BuildHeap:

$T =$ $O(n)$ ← BC, WC, Ac.

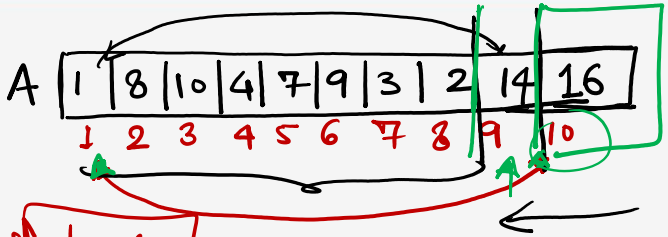
Heapsort:-

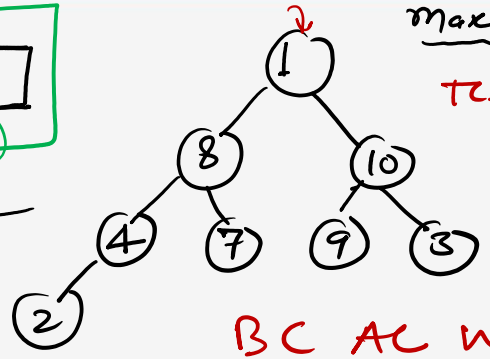


t_1

t_2

$t_1 = t_2$





Maxheap

TC = O(n)

Algo:- Heapsort(A)

1- BuildHeap(A) → O(n)

2- for i ← Length[A] downto 2 → (n-1)

3- do Exchange A[1] ↔ A[i]

4- Heapsize[A] ← Heapsize[A]-1

5- Heapify(A,1) → log n × (n-1)

BC AC WC

⇒ O(n log n)

$\log n + \log(n-1) + \log(n-2) + \cdots \log 2 + 1$

$\log n \times (n-1) \times (n-2) \times \cdots (2) \times 1 \Rightarrow \log n! \leq \log n^n \neq O(n \log n)$

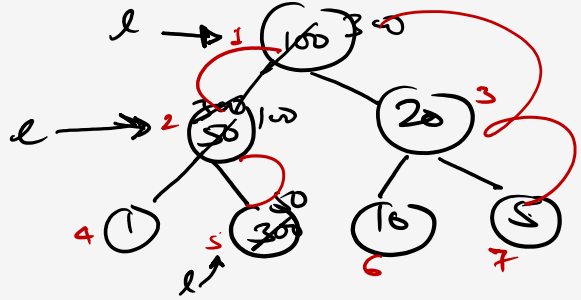
Heap operations:

WC

1) HeapIncreaseKey: $\Rightarrow O(\log n)$

Algo: HeapIncreaseKey(A, i, key)

```
{ if(key < A[i])
      "Error"
  A[i] = key;
  while(i >= 1 and A[i/2] < A[i])
  { Exchange A[i] <-> A[i/2]
    i = i/2;
  }
}
```

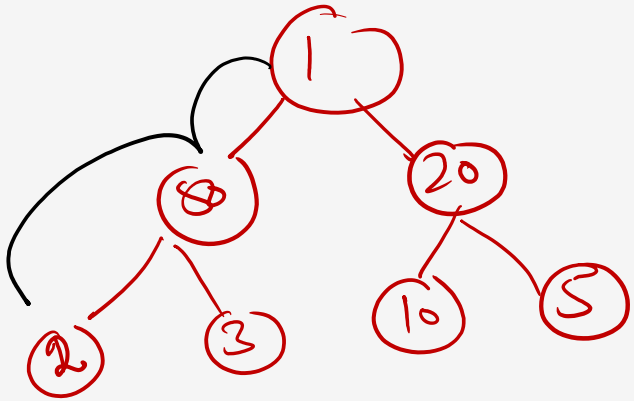


$\Rightarrow O(\log n)$

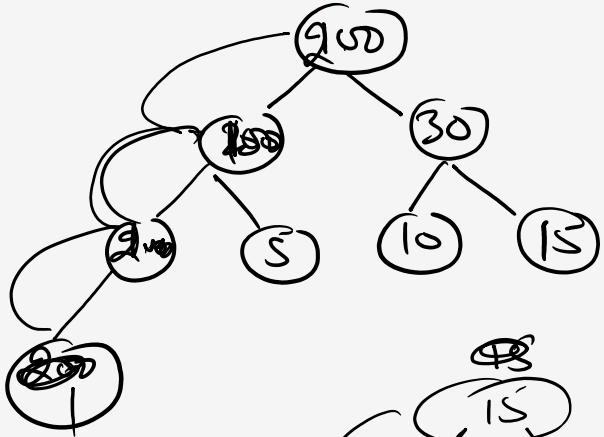
A[5] = 300 (Key)

Key = 300

$\left\lfloor \frac{5}{2} \right\rfloor = 2$

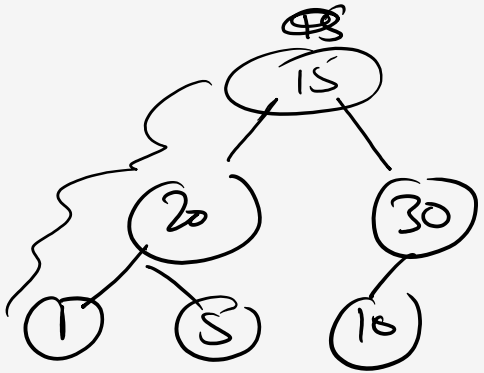


$\Rightarrow O(\log n)$



insert = 200

WC = O(log h)

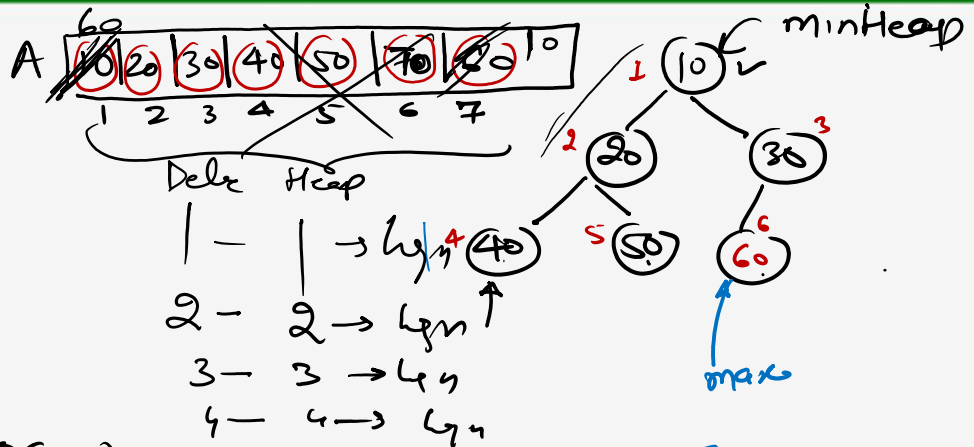


15, 20, 30, 1, 5, 10,

⇒ O(log n)

MinHeap :-

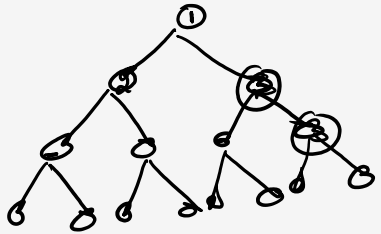
1) Search min element = $O(1)$
2) Search max element = $O(n)$
3) Find n^{th} min element = $O(n)$
4) Search $(n-1)^{th}$ min element = $O(n)$
5) Search $(\frac{n}{2}+1)^{th}$ min element = $O(n)$
6) Search 7th min element = $O(1)$ ✓
7) Delete min element = $O(\log n)$
8) Delete max element = $O(n)$
9) Delete n^{th} min = $O(n)$
10) Delete 7th min = $O(\log n)$



3rd min

$\Rightarrow 2^3 - 1 \Rightarrow 7$

min...



i.e. $(2^i - 1 \Rightarrow 3)$

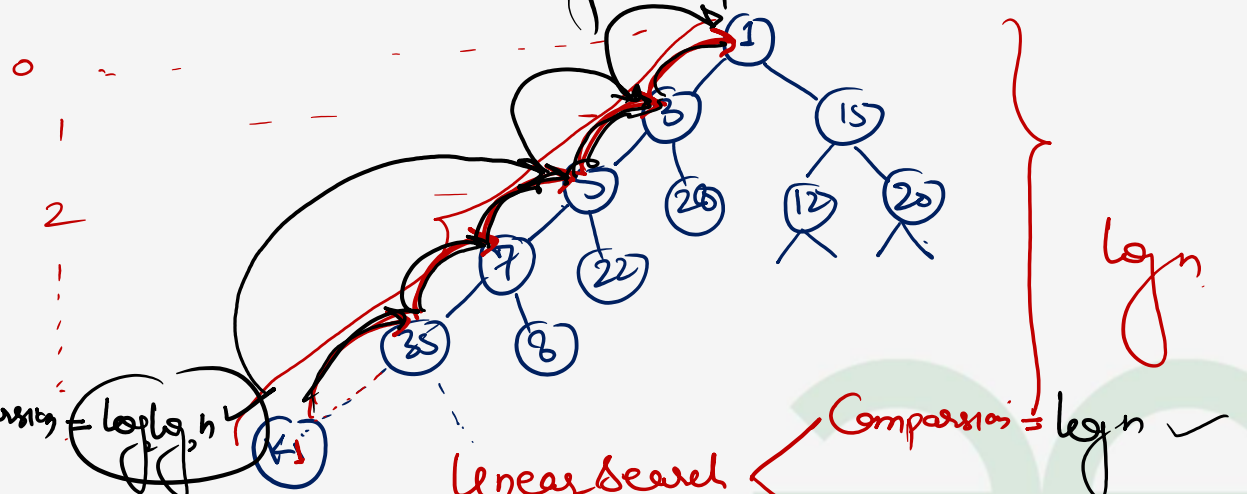
Question: Consider the process of inserting an element into a ~~max~~ min heap where the ~~max~~ min heap is represented by an Array. Suppose we perform a binary search on a path from the new leaf to the root to find the position for the newly inserted element; The number of Comparisons Performed is

a) $\Theta(\log_2 n)$

b) $\Theta(\log_2 \log_2 n)$ ✓

c) $\Theta(n)$

d) $\Theta(n \log_2 n)$



Binary Search
- Comparisons = $\log \log_2 n$
- Swap = $\log n$

Linear Search
- Comparisons = $\log n$ ✓
- Swap = $\log n$ ✓

Sorting By distribution:

1) Counting Sort
2) Radix Sort.

Counting Sort:-

- Not inplace sort
- Stable sort

$K \approx O(n)$

10, 24 ans

Algo:- CountSort(A, B, K) — Aux, max

1- for i ← 1 to K ⇒ O(K)

2- C[i] ← 0

3- for j ← 1 to length[A] ⇒ O(n) // Count.

4- C[A[j]] ← C[A[j]] + 1

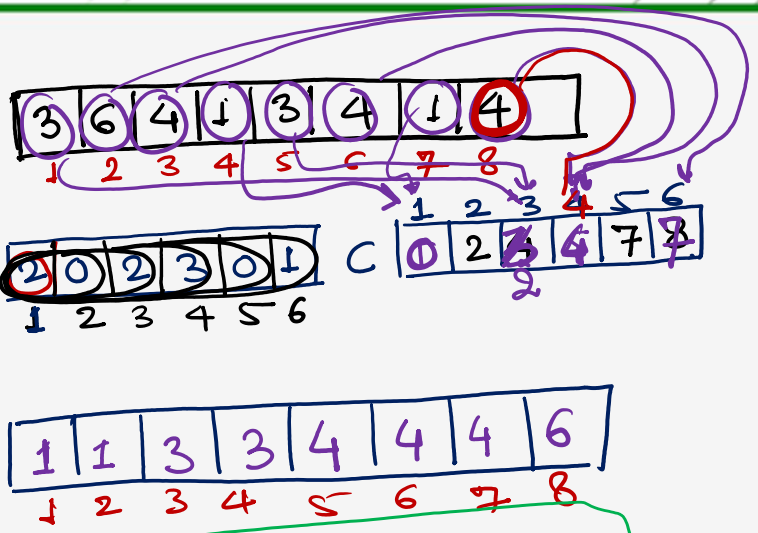
5- for i ← 2 to K ⇒ O(K)

C[i] = C[i] + C[i-1]

6- for j ← length[A] down to 1 → O(n)

7- B[C[A[j]]] ← A[j]

8- C[A[j]] ← C[A[j]] - 1



SC: $O(n + k) \Rightarrow O(n)$

$\Rightarrow O(2n + k)$ → max ka

$\Rightarrow O(n + k)$

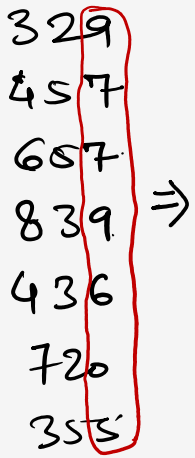
BC, WC, AC

Radix Sort :- Stable Sort.

1) Very large Numbers
2) Queue data structure used

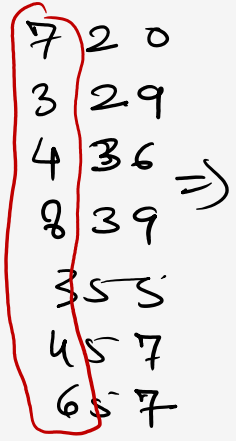
Algo: RadixSort(A, d)

1 - for i ← 1 to d (Counting Sort)
2 - do use a stable sort to sort an Array A on digit i

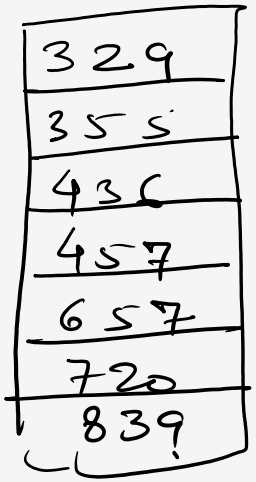


⇒

⇒



⇒



TC = $O(d(n+k))$ → max = 9 ⇒ $O(n*d)$

$TC = O(n * d) \Rightarrow O(n \log_r n) \Rightarrow O(n \lg n)$

$n = 4$ (00, 01, 10, 11)

$d = \log_r n = \lg_2 4 = 2$

$d = \lceil \lg \frac{3}{2} \rceil = 2$

$TC = O(n \log_r n)$

for linear Time $r = ?$

$r = n$

$= O(n \lg_n n)$

$\Rightarrow O(n)$

$n = 10$ (0 ——— 9)

$d = \log_{10} 10 \Rightarrow 1$

$d = \lceil \lg_{10} 9 \rceil = 1$

$d = \log_r n = \log_r k$

Q-1) Sort n-numbers in the range $\{0 - n^2\}$ and find the time Complexity using

a) Counting sort = $O(n+K) = O(n+n^2) = O(n^2)$

b) Radix sort = $O(n*d) = O(n*\log_k K) \Rightarrow O(n*\log_n n^2)$

$\Rightarrow O(2n \log n)$

$\Rightarrow O(n \log n)$

Question: if we used Radix sort to sort n integers in the range $(n^{K/2}, n^K)$ for some $K > 0$ which is independent of n, the time taken would be

a) $\Theta(n)$

b) $\Theta(Kn)$

c) $\Theta(n \lg n)$

d) $\Theta(n^2)$

$d = \log_r n^K \Rightarrow K \lg n$

$TC = \Theta(n * d)$

$= \Theta(n * K \lg n)$

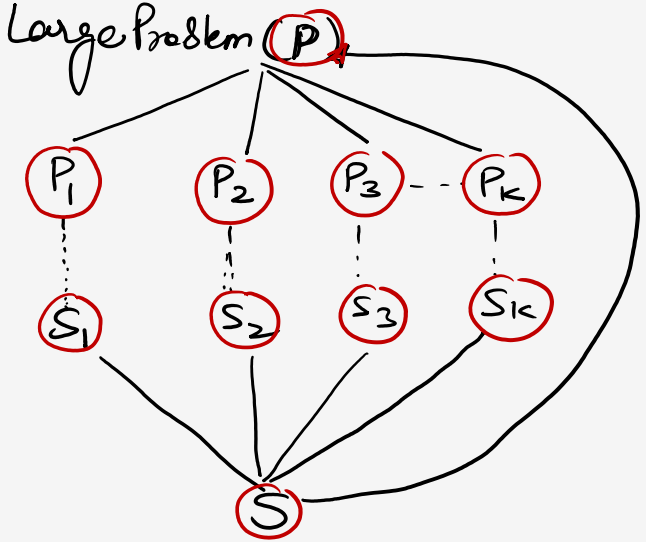
$\Rightarrow \Theta(n \lg n)$

Break: 10 mins

11:16

Divide & Conquer (DAC):-

1. **Divide:-** Break the given Problem into Subproblems of same type
2. **Conquer:-** Recursively Solve these Subproblems
3. **Combine:-** Combine the solution of all Subproblems



⇒ 1) Binary Search
2) Quick Sort
3) merge Sort
4) Finding an element
5) selection Procedure
6) Strassen's matrix multip'

Binary Search:- Array Should be Sorted

Algo: BinarySearch(a, i, J, x) → T(n)

```
{
    int mid;
    if(i == J) // Smallest problem
    {
        if (a[i] == x)
            return i;
        else return -1;
    }
    else
    {
        mid = (i+J)/2;
        if (a[mid] == x)
            return(mid);
        elseif(a[mid] > x)
            BinarySearch(a, i, mid-1, x);
        else
            BinarySearch(a, mid+1, J, x);
    }
}
```

O(1) — if(i == J)

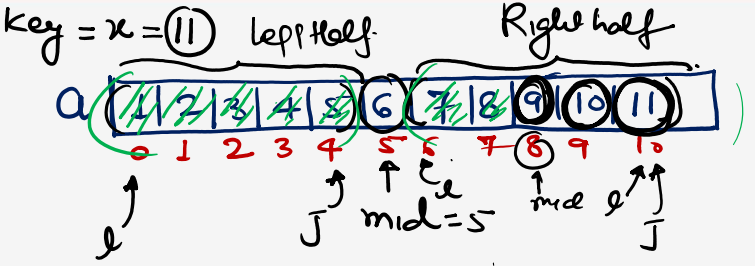
O(1) — else

O(1) — mid = (i+J)/2;

O(1) — if (a[mid] == x)

T(n/2) — BinarySearch(a, i, mid-1, x);

T(n/2) — BinarySearch(a, mid+1, J, x);



$mid = \frac{i+J}{2} = \frac{0+10}{2} = 5$

if(x == a[mid])

Recurrence Relation:-

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + k & \text{if } n > 1 \end{cases}$$

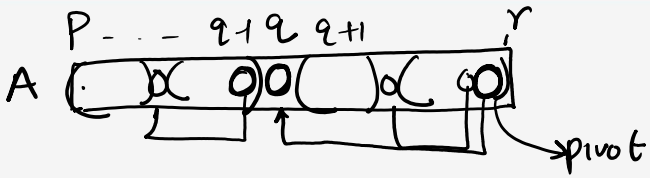
$T(n) = T(n/2) + k$

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

WC / AC: $TC = O(\log_2 n)$

BC = O(1)

Quicksort:- A[P, r]



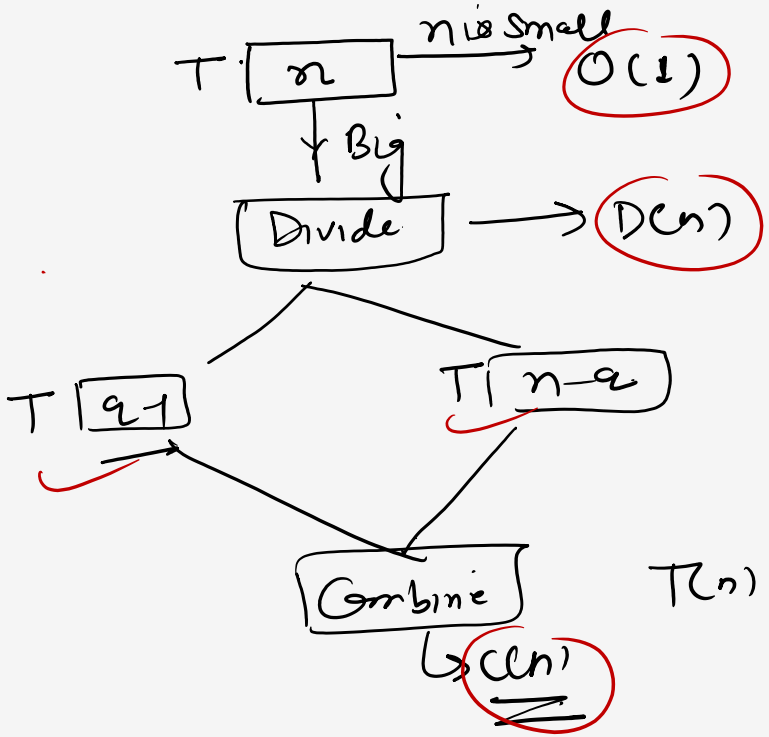
1) Divide & Conquer
2) Not stable sort
3) Not inplace sort

1) **Divide:** A[P, r] → A[P -- q-1], A[q+1 -- r]

Such that $A[P, q-1] \leq A[q+1 -- r]$

2) **Conquer:** The two subarray A[P -- q-1] and A[q+1 -- r] are sorted by recursive calls to Quicksort

2) **Combine:** Combine



$$T(n) = \begin{cases} 1, & \text{if } n \text{ is small} \\ T(a-1) + T(n-a) + D(n) + C(n) & \text{if } n \text{ is Big} \end{cases}$$

Thank You !