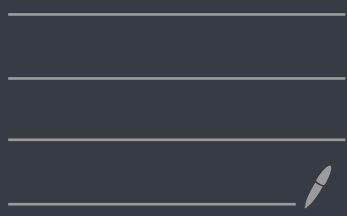


1-e $T(n) = \sqrt{n}\, T(\sqrt{n}) + n$

cost of each level = n

no. of levels → log log n

$n = 2^{2^k}$

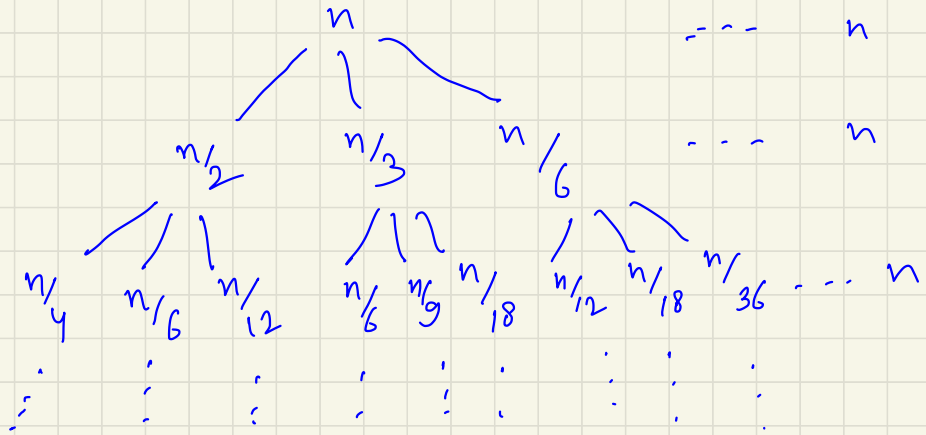
$\sqrt{n} \rightarrow 2^{2^{k-1}} \rightarrow 2^{2^{k-2}} \rightarrow \cdots \rightarrow 2^{2^0}$ (k)

$k = \log \log n$

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

1-f

$$T(n) = T(n/2) + T(n/3) + T(n/6) + n$$



max height → $\log_2 n$

min height → $\log_6 n$

So $T(n) = O(n \log n)$

$T(n) = \Theta(n \log n)$

1-g $T(n) = T(n/2) + 2T(n/3) + 3T(n/4) + n^2$

$T(n) = \Omega(n^2)$

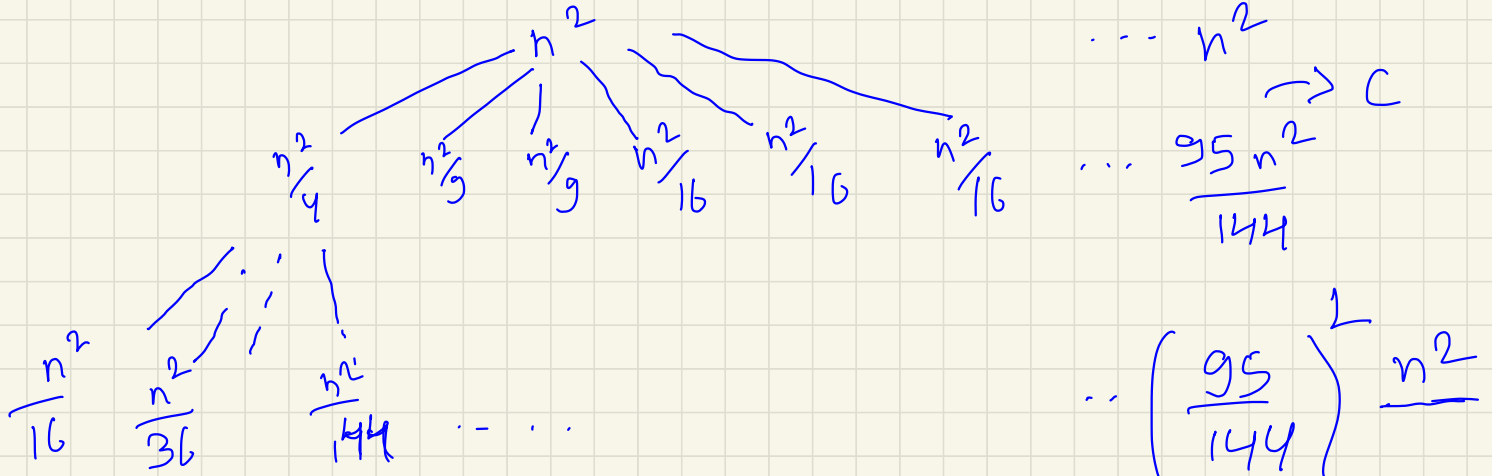
Let $T(n) = cn^2$

Then $\frac{cn^2}{4} + \frac{2cn^2}{9} + \frac{3cn^2}{16} + n^2 = cn^2\frac{95}{144} + n^2 \leq cn^2$

for large enough c.

So, $T(n) = O(n^2)$

Also, Try to solve it using Recursion Tree.



$$\epsilon = \frac{95}{144} < 1$$

$$T(n) \leq n^2 \left(\epsilon + \epsilon^2 + \cdots \epsilon^N\right)$$

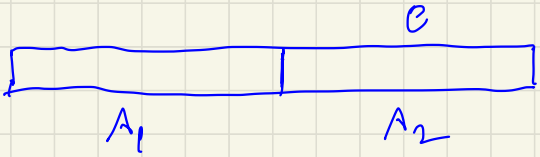
$$\leq \frac{n^2}{1-\epsilon}$$

Problem Set 2

1 → Simple Brute force → $O(n^2)$

Try using Divide & Conquer.

A :→



Observation No. of inversions in A = No. of inversions due to elements in A_1 + (in A_2)

= No. of inversions in A_1 + No. of inversions in A_2

+ $\sum_{e \in A_2}$ No. of elements in A_1 greater than e.

For unsorted A_1 and A_2, it requires $O(n^2)$ operations

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

$T(n)$ is still $O(n^2)$.

How to compute 3rd part faster?

What if A_1 and A_2 are sorted?

Let N_A denote the no. of inversions in array A.

Algorithm: Count_Inversion

Input: Array A

Output: Sorted A, N_A

1. If len(A) == 1, return A, 0
2. mid = $\lfloor len(A)/2 \rfloor$
3. Sorted A_1, N_{A_1} = count_Inversion (A[0 : mid-1])
4. Sorted A_2, N_{A_2} = count_Inversion (A[mid :])

5. Sorted A, N = Modified_merge (sorted A_1, sorted A_2)

6. $N_A = N_{A_1} + N_{A_2} + N$

7. return sorted A, N_A

Algorithm: Modified_merge

Input: Two sorted arrays A_1 and A_2

Output: Sorted array A including elements of A_1 and A_2, $\sum_{e \in A_2}$ No. of elements in A_1 g.t. e

1. $i_{A_1} = 0$, $i_{A_2} = 0$, $i_A = 0$, $N = 0$

2. While ($i_{A_1} < len(A_1)$ or $i_{A_2} < len(A_2)$)

    1. If $i_{A_1} == len(A_1)$,

        $A[i_A] = A_2[i_{A_2}]$, i_A++ , i_{A_2}++

    2. Else if $i_{A_2} == len(A_2)$,

        $A[i_A] = A_1[i_{A_1}]$, i_A++ , i_{A_1}++

3. Else if $A_1[i_{A_1}] > A_2[i_{A_2}]$,

$A[i_A] = A_2[i_{A_2}]$, i_A++, i_{A_2}++, $N = N + len(A_1) - i_{A_1}$

4. Else,

$A[i_A] = A_1[i_{A_1}]$, i_A++, i_{A_1}++

3. return A, N

$len(A_1) - i_{A_1}$ → No. of elements in A_1 s.t. $A_2[i_{A_2}]$

Proof of Correctness

Proof by induction on length of A.

Base Case: $len(A) = 1$. Trivial

Now, suppose algorithm returns correct ans for any array of length $\leq k$.

For an input array of length $k+1$, we can say that algorithm gives correct answer because of above observation, induction hypothesis and correctness of Modified_Merge.

Time Complexity

Similar to Merge Sort.

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

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

[2] what do they mean by fastest ?

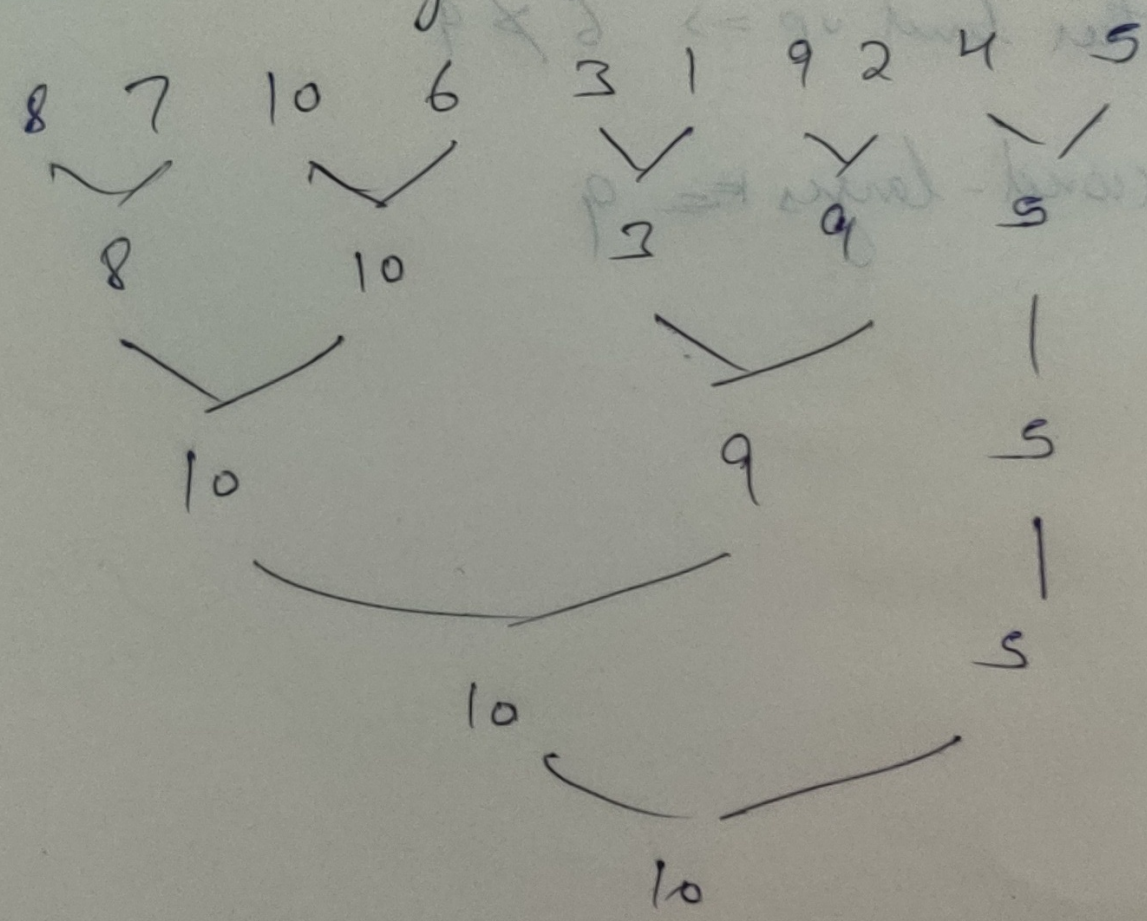
Time complexity or comparison complexity

if Time complexity

```
largest = A[1]
second-largest = -∞
for i = 2 to n
    if A[i] > second-largest :
        if A[i] > largest :
            second-largest = largest
            largest = A[i]
        else
            second-largest = A[i]
```

Time : O(n)

Comparison Complexity : Tournament method



Comparisons to find the largest = $n-1$

Second largest must have to lose from the largest.

So, we will go up along the tree

Comparisons to find second_largest = $\log n - 1$

eg. we go up

second_largest -> 5

another level up => $9 > 5$

second_largest = 9

another level up => $8 \not> 9$

another level up => $6 \not> 9$

$\therefore$ second_largest = 9

=> To convert to code, will have to use clever data structures

3→ Given an input array A, the goal is to find the maximum element in the array.

For an algorithm and a input, consider the following graph.

Vertices → 1 to n

Edges → (i, j) if A_i and A_j are compared at anytime during the algorithm.

No. of edges ≤ No. of comparisions

Observation → no. of comparisions $< n-1 \Rightarrow$ the graph has more than 1 connected component.

Suppose there is an algorithm which gives index of max. element using less than $n-1$ comparisions on all inputs.

For a fixed input array A, let $C_1, C_2, \ldots C_k$ be the connected components in the graph w.r.t the algorithm. $k \geq 2$

Wlog, Let the algorithm outputs u which belongs to C_1.

Consider another input array A' such that,

$$A'[i] = A[i] + N \quad \text{if } i \in C_2$$
$$= A[i] \quad \text{otherwise.}$$

Take N very large.

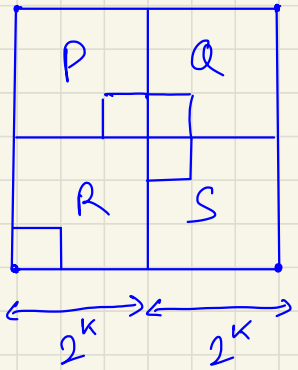
observation → Graph w.r.t. A' is same as Graph w.r.t to A.

Hence the algorithm doesn't make any difference for inputs A and A' and will return same index.

A contradiction as max of $A' \in C_2$

4 → Proof by induction on k.

Suppose it is true for k.



→ Fill P, Q, R, S using induction hypothesis.

→ Then fill the middle gap using building block A.

proved for $k+1$

5 Modified binary search

i i+1 i i+1

1 2 3 5 8 10 7 6 3 2

→ MBS (A, n)

p = 1, r = n

while (p < r):

$q = \left\lfloor \frac{p+r}{2} \right\rfloor$

//check neighbour elements of q

if A[q-1] < A[q] & A[q] > A[q+1]:

return A[q]

elif A[q-1] ≥ A[q]:

r = q-1

else

p = q+1

6 → Algorithm (binary search)

```
FindIndex (A, start_index, last_index) {
    if (last_index < start_index)
        return False.
    if [start_index == last_index) {
        if [A[start_index] = start_index)
            return True
        return False }
    mid = ⌊(start_index + last_index) / 2⌋
    if (A[mid] = mid)
        return True
else if (A[mid] > mid)
        return FindIndex (A, start_index, mid-1)
else return FindIndex (A, mid+1, last_index) }
```

Base Case

Proof of Correctness

We will show that the algorithm returns true iff there exist an start_index $\leq i \leq$ last_index such that $A[i] = i$.

Proof by induction on l = last_index − start_index.

Base case $l \leq 0$. Trivial

Suppose the algorithm works correctly for $l \leq k$.

Suppose $l = k+1$.

1st Case $A[mid] = mid$

2nd Case $A[mid] > mid$

Claim: In array A with distinct elements if $A[i] > i$, then

$\forall j \geq i \quad A[j] > j$

Proof: Try to do it yourself

Hence, the answer is true iff there is an startIndex $\leq i <$ mid s.t. $A[i] = i$.

The algorithms does this correctly. {By Induction hypothesis}

3rd Case Similar to 1st Case.

Time Complexity

Same as binary search.