Date:

Computational Problem

Problem Statement → Solution 1 → Algorithm = Pseudocode (Paradigms)

Solution 2

Solution 3

Solution N

Pseudocode: Set of Steps, Logic → Comprises. → sequence, selection, iteration

run, exec → m/c

to accomplish some task

(Computational task)

Analysis tools O, Ω, Θ

* Asymptotic Analysis (Notations)
* Matrix Addition Problem.

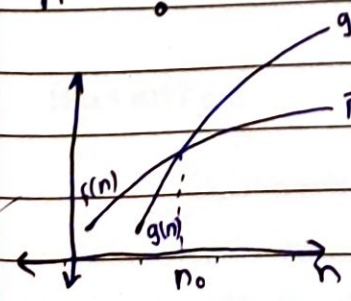
```
MAT_ADD (a[], b[], n){
    for(i=0, i<n, i++)
        for(j=0, j<n, j++){
            c[i][j] = a[i][j] + b[i][j]
        end for
    end for.
```

$n + 2n^2 \Rightarrow O(n^2)$

- Algorithm comparison should be independent of machine
- for this purpose count how many basic steps are there

* Big O Notation:

$F(n) = O(g(n))$ iff there exist positive constants C and n_0 such that $F(n) \leq C \cdot g(n)$ for all $n \geq n_0$.



1. $f(n) = 10n^2 + 5n + 6$

$f(n) \leq 21n^2 \quad n \geq 1$

$\Rightarrow f(n) = O(n^2)$

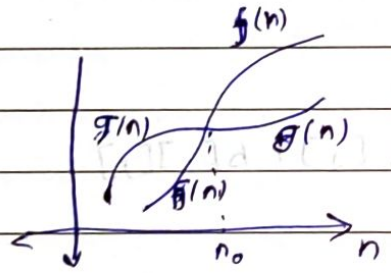
2. $f(n) = 6 \cdot 2^n + 2 \cdot n^2$

$f(n) \leq 8 \cdot 2^n \quad | \; n \geq 1$

$\Rightarrow f(n) = O(2^n)$

* Ω Notation.

- $F(n) = \Omega(g(n))$ iff there exist positive constant C and n_0 such that $F(n) \geq C \cdot g(n)$ for all $n \geq n_0$.
- lower bound.



* Θ Notation

$F(n) = \Theta(g(n))$ iff ~~there exists~~ $F(n) = O(g(n))$ and $F(n) = \Omega(g(n))$

$f(n) = 10n^2 + 5n + 6$

$= \Omega(n^2)$

$f(n) \geq n^2 \quad n \geq 100$

const = 1

$f(n) = \Theta(n^2)$

Note: Dont bother about constants.

$$S(n) = \sum_{i=1}^{N} i = \frac{N(N+1)}{2} \approx \frac{N^2}{2}$$

* $$\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6} \approx \frac{N^3}{3}$$

* $$\sum_{i=0}^{h} i \cdot 2^i = (h+1) \cdot 2^{(h+1)} - 2^{(h+2)} + 2$$

* $$\sum_{i=0}^{N} 2^i = 2^{N+1} - 1$$

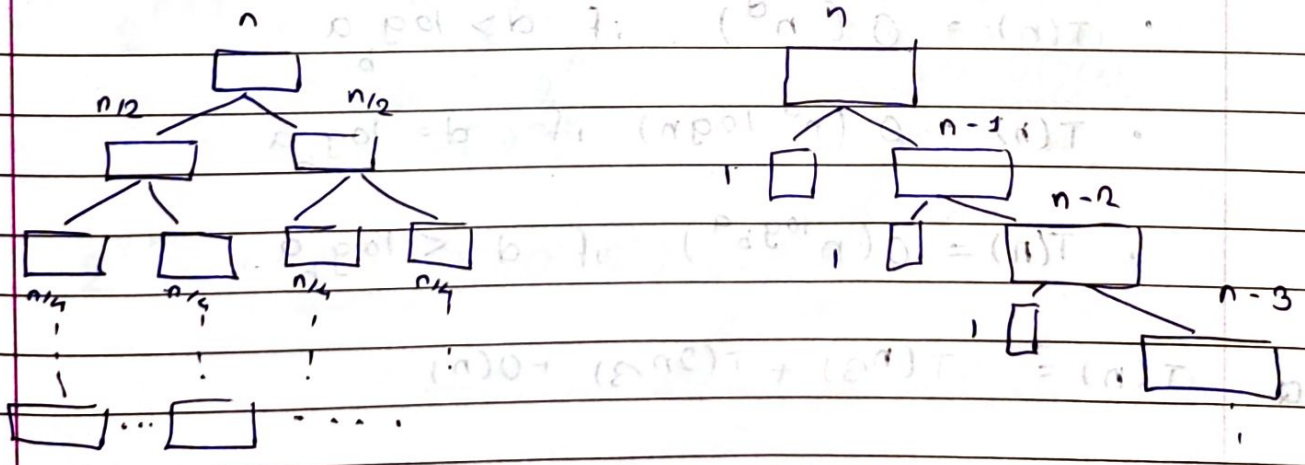
* $$\sum_{i=0}^{N} A^i = \frac{A^{N+1} - 1}{A - 1}$$

* If $0 < A < 1$ then

$$\sum_{i=0}^{N} A^i = \frac{1}{1 - A}$$

* Time Complexity $\propto$ Number of Steps.

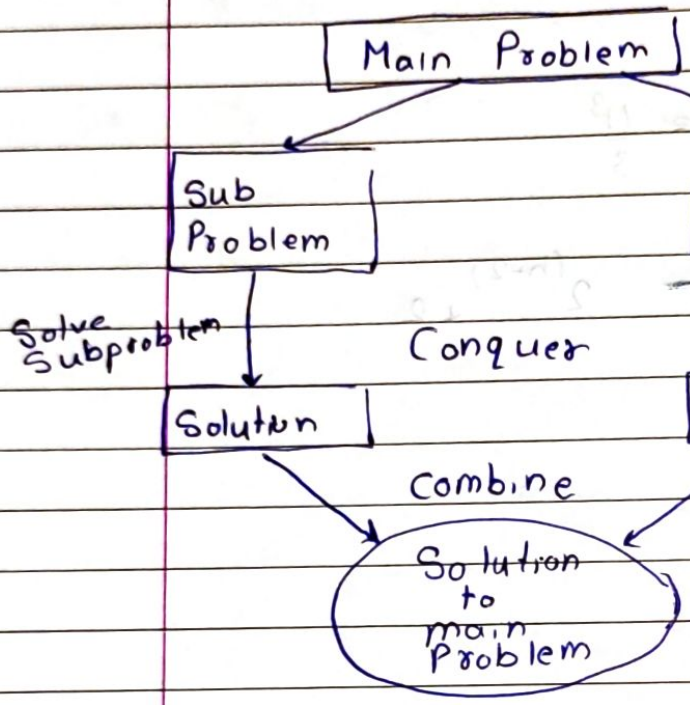
* Balanced & skew



Balanced.

Skew

* Divide and Conquer



Ideally the divde operation should be balanced

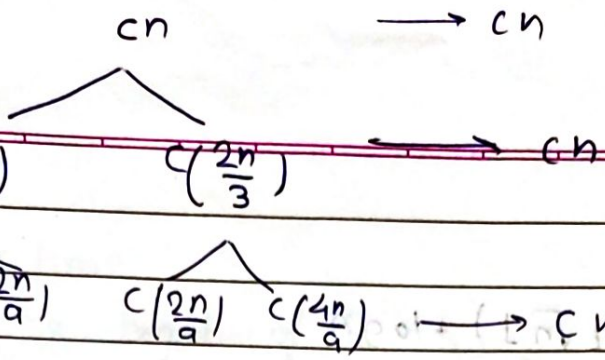
* Master Theorem.

If $T(n) = a \cdot T(n/b) + O(n^d)$

for constants $a > 0$, $b > 1$, $d \geq 0$ then

- $T(n) = O(n^d)$ if $d > \log_b a$
- $T(n) = O(n^d \log n)$ if $d = \log_b a$
- $T(n) = O(n^{\log_b a})$ if $d < \log_b a$

Q $T(n) = T(n/3) + T(2n/3) + O(n)$



It will go till trivial level equal to height of tree =

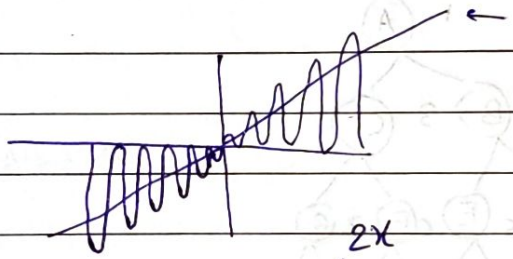
height = $\log_{3/2} n$

⇒ Complexity = $cn \times \log_{3/2} n$

$= O(n \log n)$

Q $f(x) = x(1 + \sin x)$

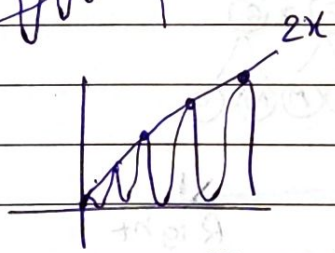
$g(x) = x$.



True or false.?

Q $2^{n+1} = O(2^n)$ ✓

$2^{2n} = O(2^n)$ ×



$2^{n+1} = 2^n \cdot 2^1$

$= 2 \cdot 2^n$

$= O(2^n)$

$f(x) \le c \cdot g(x)$

$f(x) \le 2 \cdot g(x)$

$\Rightarrow f(x) = O(g(x))$

∵ 2 is const.

$2^{2n} \cdot 2^{2n} \neq O(2^n)$

Date:

$O(\log n \log\log n)$

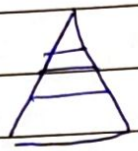
assignment -

$T(n) = 2\,T(\lfloor \sqrt{n} \rfloor) + \log n$

$m = \log n$ i.e $n = 2^m$

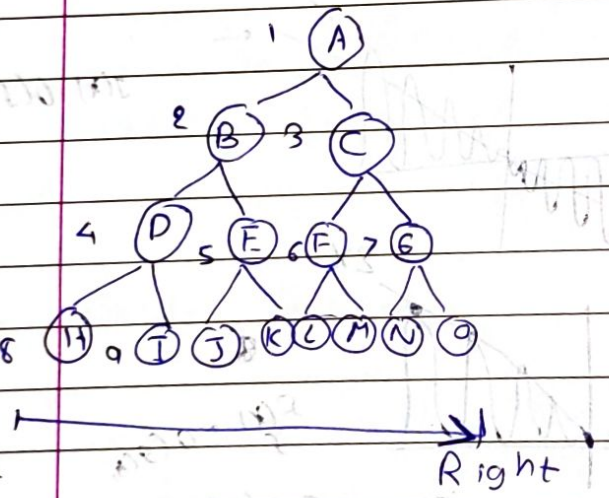
$S(m) \leftarrow$ Solve.

* Heap Structure (using Array) and Heap Sort :



Min or Max.

Data Structure.



i - Index to array A[] (1, 2..., n)

$l(i)$ - left child of node 'i'

$r(i)$ - right child of node 'i'

$P(i)$ - Parent of node 'i'

* Complete Binary Tree :

$l(i) = 2 \cdot i$

$r(i) = 2 \cdot i + 1$

$P(i) = \lfloor i/2 \rfloor$

$\lfloor \rfloor \rightarrow$ floor

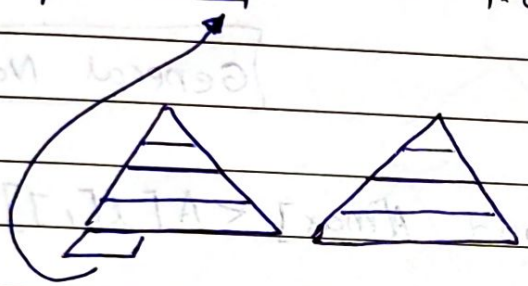
Assume max heap

What is the height of a heap.

assume heap height = h

Claim: $\log(n+1) - 1 \leq h \leq \log n$

$$\sum_{i=0}^{h-1} 2^i + 1 \leq n \leq \sum_{i=0}^{h} 2^i$$



$$\sum_{i=0}^{N} A^i = A\frac{A^{N+1} - 1}{A - 1} \quad (N = h-1)$$

$$2^h + 1 \leq n \leq 2^{h+1} - 1$$

$$2^h \leq n \leq 2^{h+1}$$

$$h \log_2 2 \leq \log h \leq (h+1) \log_2 n$$

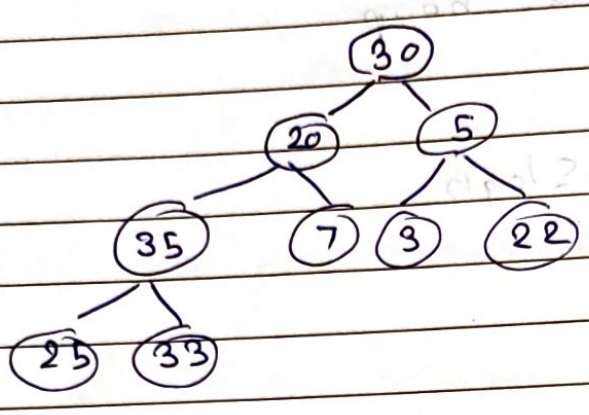
$$h \leq \log n \leq h + 1$$

h is $\Theta(\log n)$

Height of the heap is always $\Theta(\log n)$

* Maintaining heap property:

A[] = 30, 20, 5, 35, 7, 3, 32, 25, 33



General Node:

Procedure downheap(i);
max := i;
if l(i) ≤ heap_Size[A] and A[max] < A[l(i)] then
max = l(i)
end if
if r(i) ≤ heap_Size[A] and A[max] < A[r(i)] then
max = r(i)
end if
if max ≠ i then A[i] ↔ A[max]
downheap(max)
end if

downheap the
For each ^ Total number of downheap calls will be
O(h) or big O logarithmic

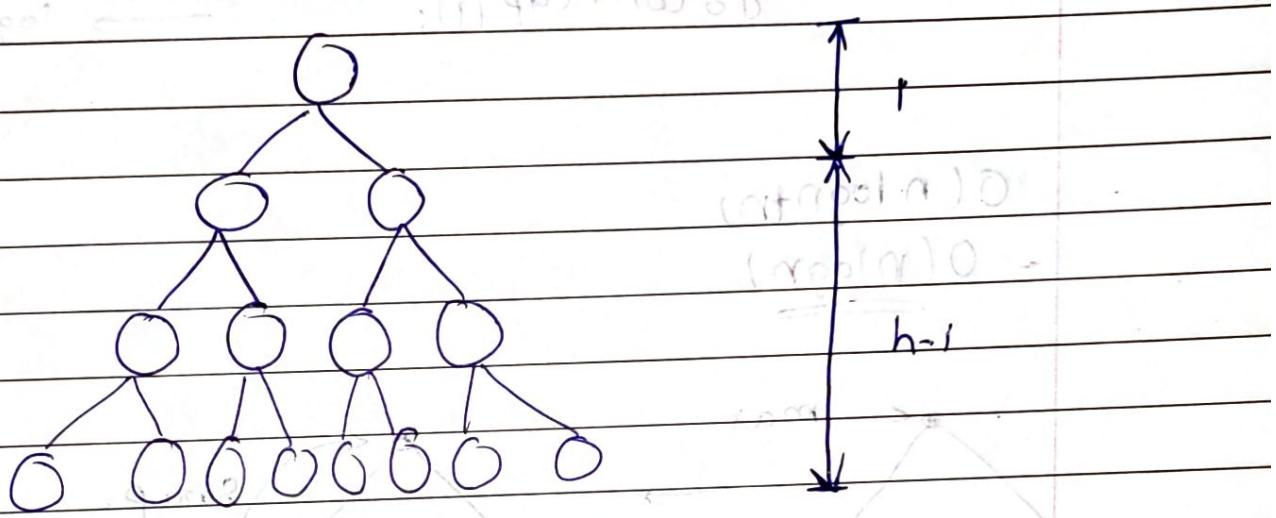
* Building heap.
I/P: Array of unsorted numbers using array A[].
n: Total no of elements as I/p

procedure Buildheap(n);
for i = n down to 1 do
downheap(i);
end f

Complexity of building heap Seems to be $n \log n$ but in reality it is $O(n)$

$$\sum_{i=0}^{h} i \cdot 2^i = (h+1)2^{h+1} - 2^{h+2} + 2$$

$$= (h+1)2^{h+1} - 2^{h+2} + 2$$



Total Complexity: at height i there are 2^i nodes which will go through height $h-i$

$\therefore$ The total complexity will be $\sum_{i=0}^{h} 2^i \cdot O(h-i)$

$$= O\left(h \sum_{i=0}^{h} 2^i - \sum_{i=0}^{h} i \cdot 2^i\right)$$

$$= O\left(h \cdot (2^{h+1} - 1) - (h+1)2^{h+1} + 2^{h+2} - 2\right)$$

$$= O\left(h \cdot 2^{h+1} - h - (h+1)2^{h+1} + 2^{h+2} - 2\right)$$

$$= O\left(2^{h+2} - 2^{h+1} - 2 - h\right)$$

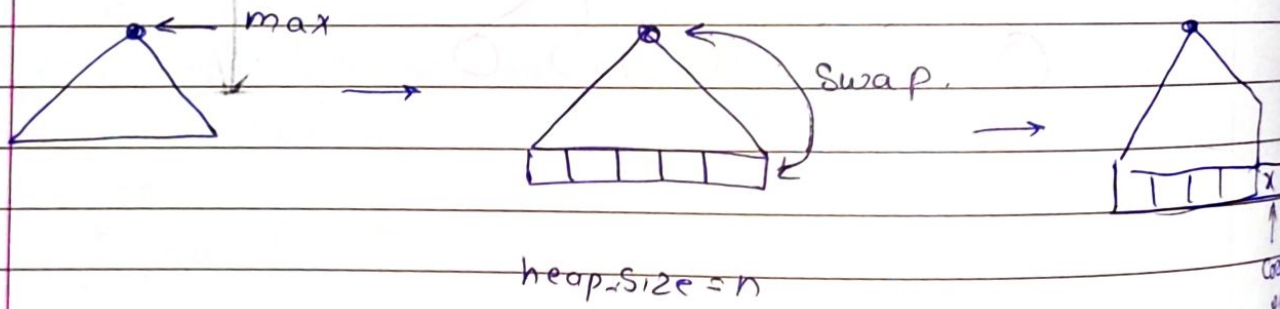
$$= O(2^h) = O(2^{\log_2 n}) \quad (\because h = O(\log n))$$

$$= O(n)$$

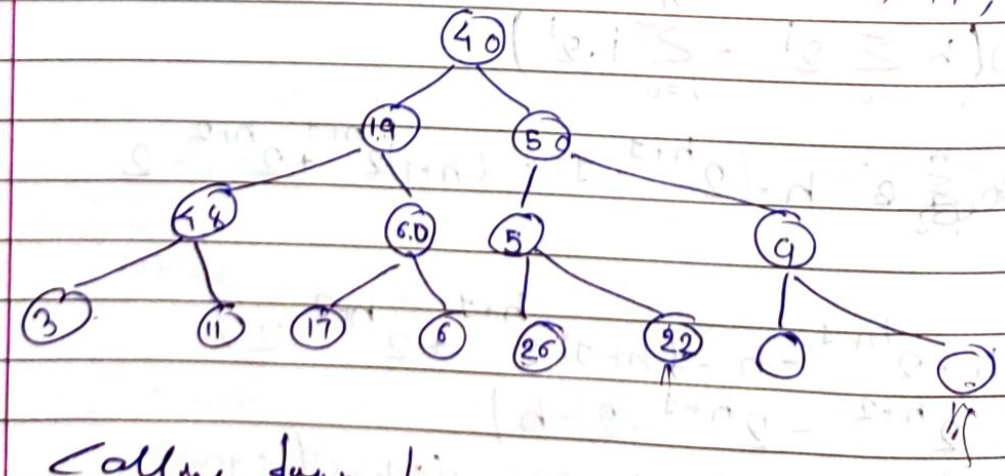
* Heap Sorting:

Procedure heapsort(n)
Buildheap(n) heap_size[A]=n; → n
For i:=n dow to 2 do → n
A[1] ⟷ A[i];
heap_size[A]:=i-1;
downheap(1); → log n.

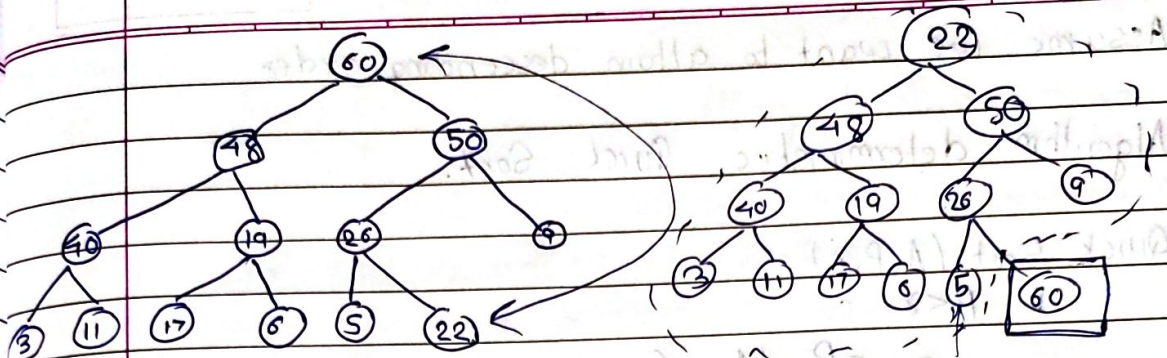
O(n log n + n)
- O(n log n)

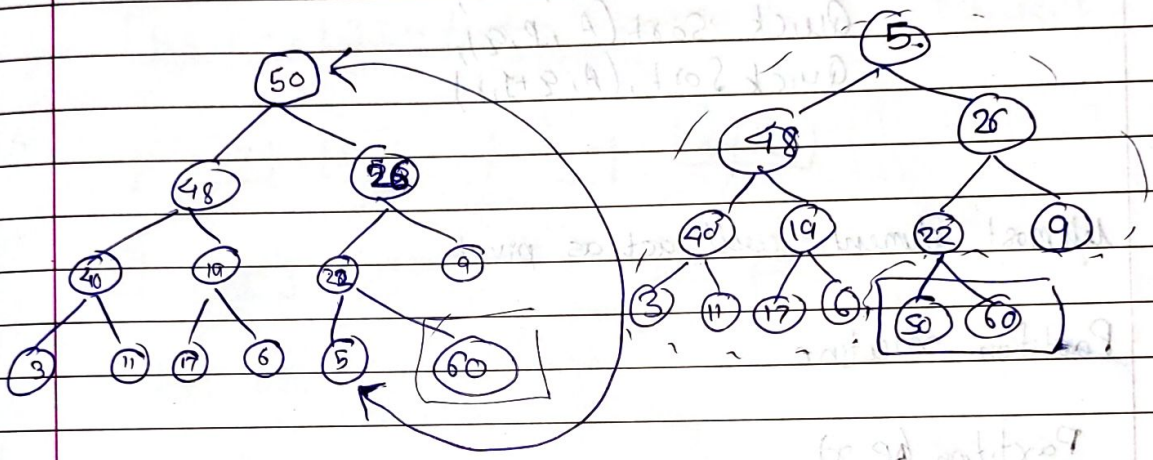


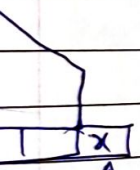
A[] 40, 19, 50, 48, 60, 5, 9, 3, 11, 17, 6, 26, 22.



Calling down heap

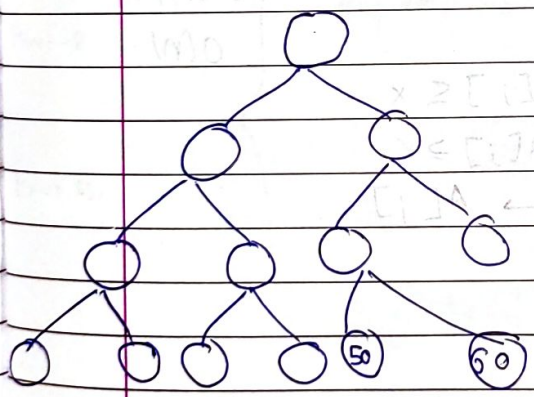






Correct place.

$r = n-1$.



★ Quick Sort:

This paradigm is based on divide and conquer

$O(N^2)$ 1) Deterministic Quick Sort. → Pivot → Left / Right most

$O(N \log N)$ 2) Randomized Quick Sort. → Random Pivot

Heap is used for priority queue.

A[P, r]

P r

Assume we want to attain descending order

Algorithm deterministic Quick Sort.

```
Quick_Sort (A, P, r)
    if P<r
        q := Partition (A, P, r);
        Quick_Sort (A, P, q);
        Quick_Sort (A, q+1, r)
```

leftmost element will act as pivot.

Partition routine

```
Partition (A, P, r)
    x = A[P]; i = P-1; j = r+1;
loop
    repeat j = j-1 until A[j] ≤ x
    repeat i = i+1 until A[j] ≥ x
    if i < j then A[i] ⟷ A[j]
    else partition = j;
        exit;
```

O(2n) ≈ O(n)

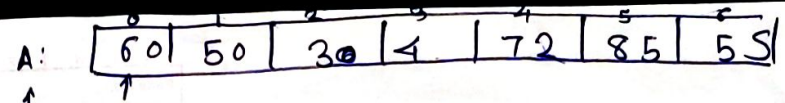
Complexity Analysis of deterministic.

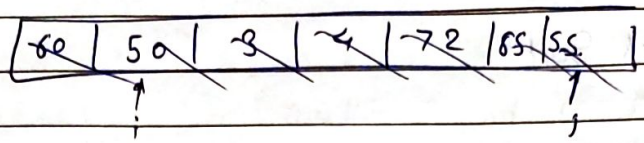
Worst case behaviour

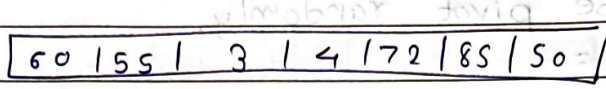
$T(n) = T(q) + T(n-q) + O(n)$

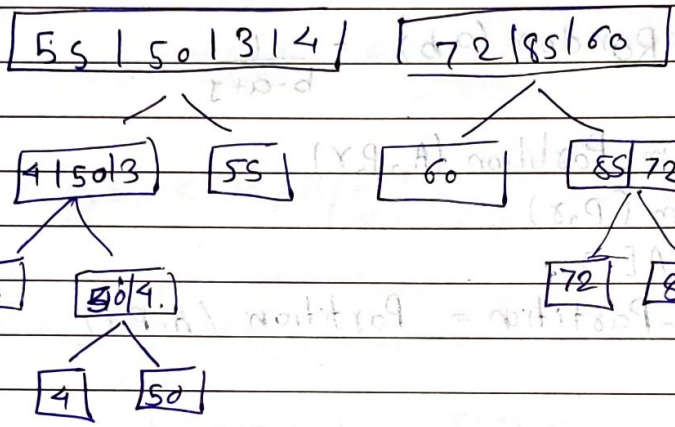
$T(1) = 1$

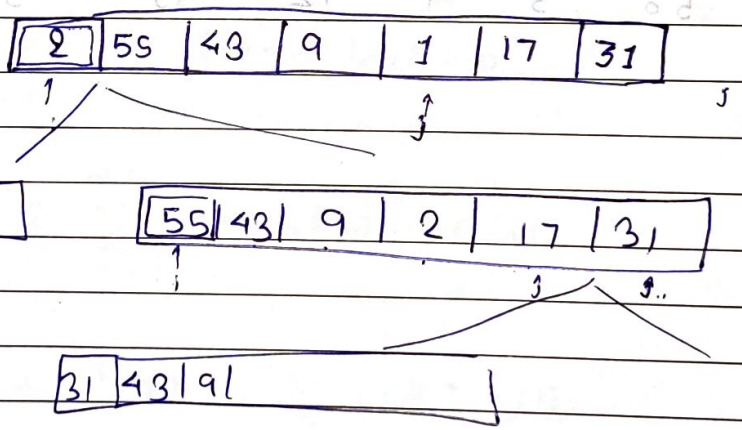
If partitions are balanced the height of divide logic is $O(\log n)$











Pivot-2.

Pivot 55.

Randomized Quick Sort:

~~Deterministic~~ Choose pivot randomly.

Random (a, b) =

Generate a random number between 'a' an

Probability [i = Random (a, b)] = $\frac{1}{b-a+1}$

```
Function Random_Partition (A, P, r)
    i = random (P, r)
    P[P] ⟷ A[i] ;
    Random -Partition = Partition (A, P, r)
    end
```

60 50 3 4 72 85 55

* Complexity Analysis of randomized Quick Sort.

  * Average Case Analysis:

$$T(n) = \frac{1}{n}\left(T(1) + \dots T(n-1) + \sum_{q=1}^{n-1} T(q) + T(n-q)\right)$$

* There are 2 worst cases when the pivot = 1st Smallest and if pivot = 2nd Smallest

| 3 | 5 | 1 | 7 | 8 |

1 — 3, 5, 7, 8.

Assume by induction $p(n) \leq a.n\log n + b$ where a, b are constants.

Tool. $\sum_{k=1}^{n-1} k \log k \leq \log n\left(\frac{n^2}{2}\right) - \left(\frac{n^2}{8}\right)$

$$T(n) = \frac{2}{n} \sum_{k=1}^{n-1} T(k) + O(n)$$

Assume by induction $T(n) \leq an\log n + b$ where a, b are const.

Hypothesis: for $k < n \rightarrow T(k) \leq ak\log k + b$. true.

$$T(n) \leq \frac{2}{n} \sum_{k=1}^{n-1} (ak\log k + b) + O(n).$$

$$\leq a.n\log n - \frac{an}{4} + 2b + O(n).$$

$$\leq an\log n + b + O(n) + b - \frac{an}{4}$$

(as $a >> b$ we can ignore few terms).

$$\leq O(n\log n).$$

2	5	7	8	10	12	17	20

↑ Pivot.

Approach 1: Deterministic.

{2, 5, 7, 8, 10, 12, 17, 20}

{2} {5, 7, 8, 10, 12, 17, 20}

{5} {7, 8, 10, 12, 17, 20}

{7} {8, 10, 12, 17, 20}

{8} {10, 12, 17, 20}

{10} ⋮

Approach 2: Randomised.

i = Random (1, 8).

assume 3 index.

Swap (A[i], A[p]).

2	5	7	8	10	12	17	20

↓

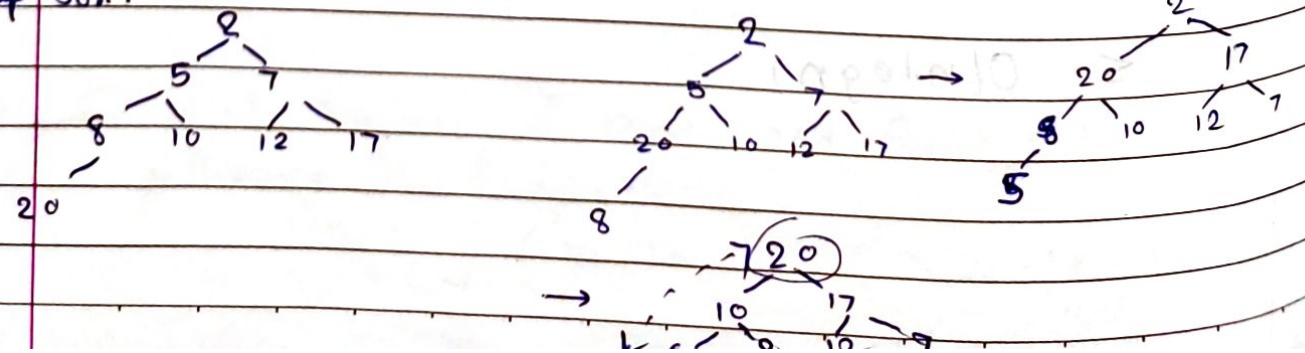
7	5	2	8	10	12	17	20

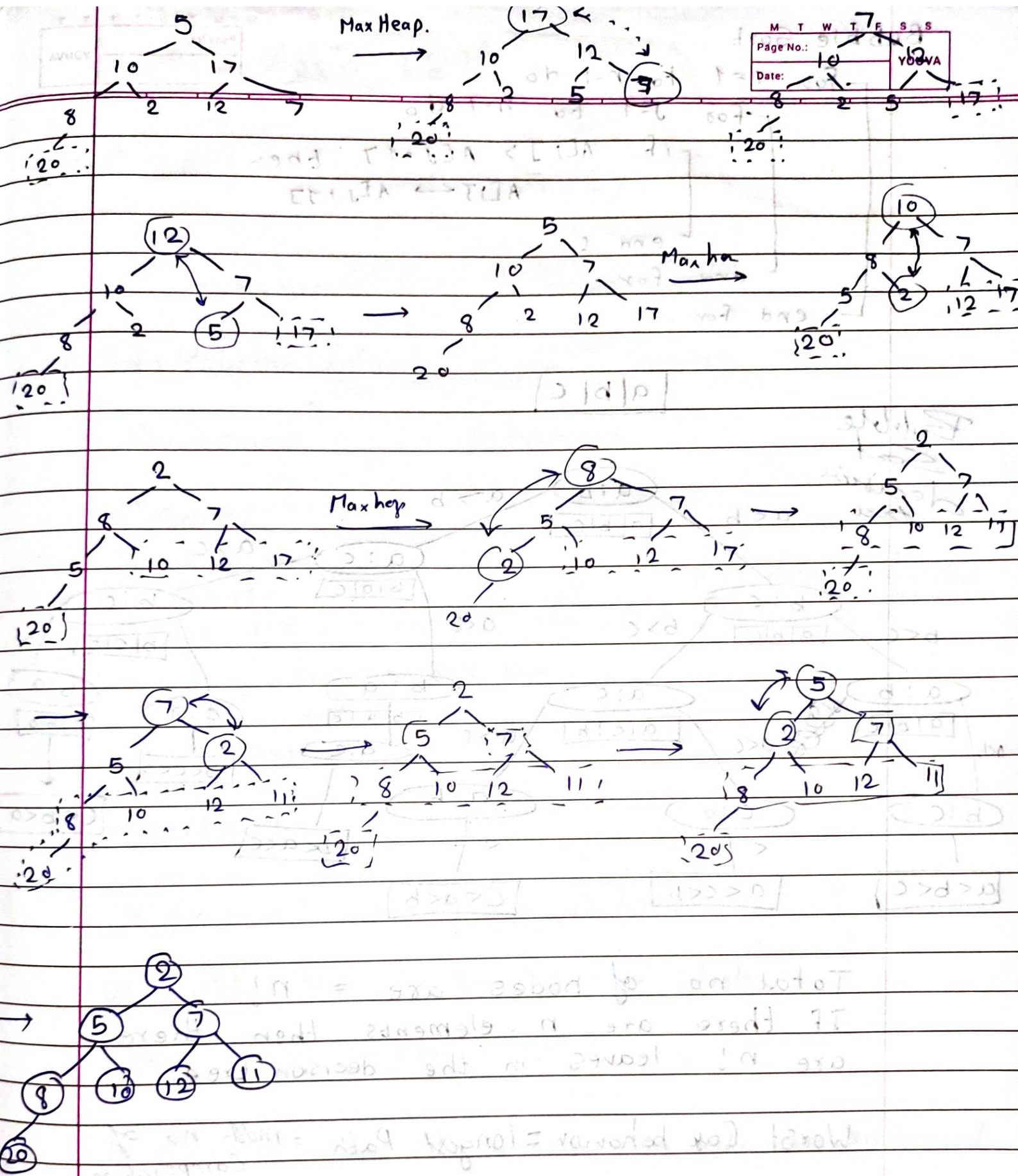
{2, 5} {7, 8, 10, 12, 17, 20}

↓

Random (

Heap Sort.





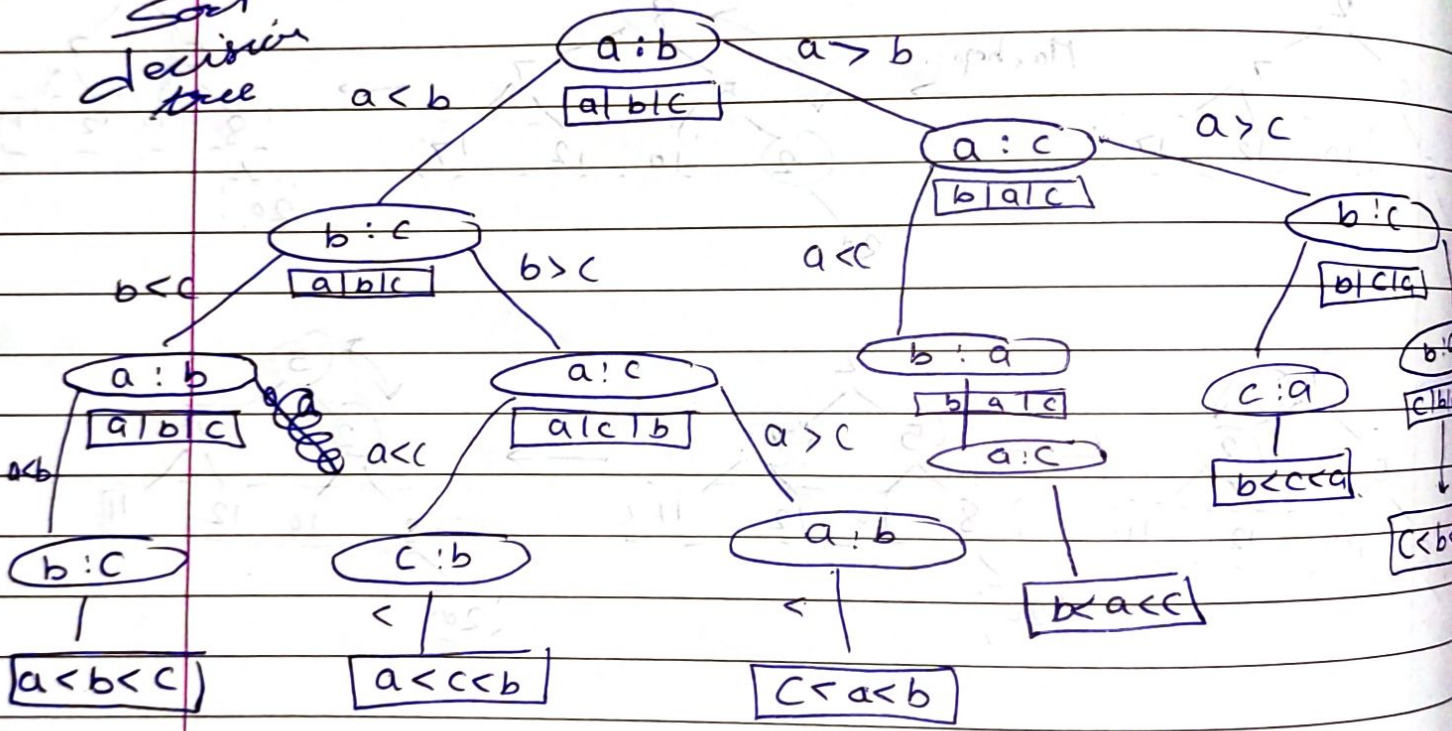
Any Sorting algorithm based on comparator operator will have lower bound of $\Omega(n \log n)$.

Bubble Sort

```
for i = 1 to n-1 do
    for j = 1 to n-1 do
        if A[j] > A[j+1] then
            A[j] ↔ A[j+1]
        end if
    end for
end for
```

a | b | c

Bubble Sort decision tree



Total no of nodes are = $n!$

If there are n elements, then there are $n!$ leaves in the decision tree.

Worst Case behavior = longest Path = most no of Comparables.

A binary tree of height h can have at most $\leq 2^h$ leaves.

$$2^h \geq n!$$

$$h \log_2(2) \geq \log_2 n! \Rightarrow h \geq \log n!$$

* Stirling Formula:

$$n! \geq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$\Rightarrow h \geq \frac{1}{2}\log_2(2\pi n) + n\log_2\left(\frac{n}{e}\right)$$

$$h = \Omega(n \log n) \quad \text{(lower Bound)}$$

* Order Statistics / Selection problem.

Randomised. Deterministic.

K^{th} Smallest

Function min — find lowest Element

```
min = 1;
For i = 2 to n do
    if A[min] > A[i] then
        min = i;
    end if
end for
min = A[min];
```

$O(n)$

To find k^{th} Smallest $\rightarrow O(kn)$

or Sort array and then find $\rightarrow O(n \log n)$

Best = $\min(O(kn), O(n \log n))$

In place of these 2 we have an algorithm randomised Selection problem.

```
Random_Select (P, r, i)
    if P = r then return P;
    else
        q = Randomized_partition (P, r);   → O(n)
```

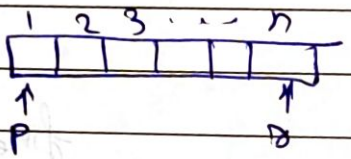
Reccurence relation for

$\sum_{k=1}^{n-1} T(k) + T(n-k)$

Page No.:
Date:

```
k = q - p + 1;   // No of elements in left subpart
if i ≤ k then return
Random_Select (p, q, i);
else return
Random_Select (q+1, r, i-k);
end if
end if.
```

Initially:



$p = 1$, $r = n$

Overall Complexity → $O(n)$

Q A = [30, 25, 3, 40, 43, 33, 6, 12, 31, 18, 1.]

Q B = [ ]

Find 5th

30, 25, 3, 40, 43, 33, 6, 12, 31, 18, 1.

p r

Partition (1, n, A)

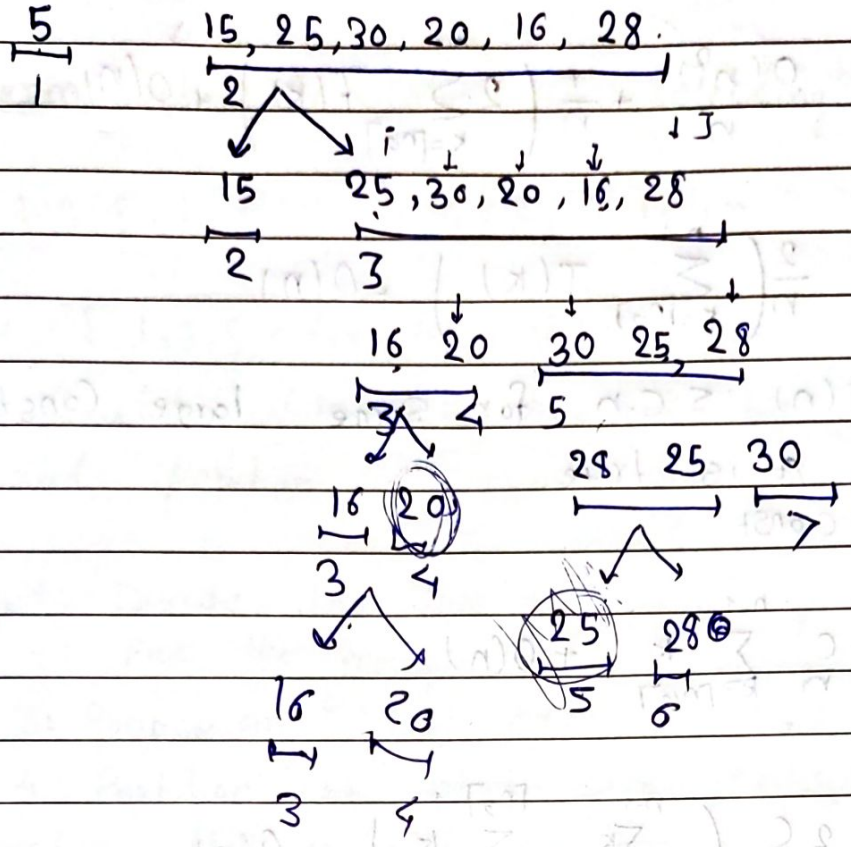
1, 25, 3, 18, 12, 6 | 33, 43, 31, 40, 30

1 | 25 3 18 12 6 | 30 31 43 40 33

6 3 18 12 | 25

3 | 6 18 12

5 , 15 , 25, 30 , 20 , 16 , 28 . 4th smallest.



* Avg Complexity Analysis:
There are two cases.

$$T(n) \leq \frac{1}{n}\left(T\left(\max(1, n-1)\right) + \sum_{k=1}^{n-1} T\left(\max(k, n-k)\right)\right) + O(n)$$

$$T(n) \leq \frac{1}{n}\left(\underbrace{T(n-1)}_{O(n^2)} + 2 \cdot \sum_{k=\lceil n/2 \rceil}^{n-1} T(k)\right) + O(n)$$

10 - elm

1 9	9 1
2 8	8 2
3 7	7 3
4 6	6 4
5 5	5 5.

All term Occur 2 times So

$$T(n) \leq \frac{O(n^2)}{n} + \frac{1}{n}\left(2\sum_{k=\lceil n/2 \rceil}^{n-1} T(k)\right) + O(n)$$

$$T(n) \leq \frac{2}{n}\left(\sum_{k=\lceil n/2 \rceil}^{n-1} T(k)\right) + O(n)$$

Assume $T(n) \leq C \cdot n$ for some large Constant C if $k < n$ it is true

$T(1) = \text{const}$

$$T(n) = 2 \cdot \frac{C}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} k + O(n)$$

$$T(n) \leq 2\frac{C}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil n/2 \rceil - 1} k\right) + O(n)$$

$$T(n) \leq 2\frac{C}{n}\left(\frac{(n-1)(n)}{2} - \frac{\left(\frac{n}{2}-1\right)\frac{n}{2}}{2}\right) + O(n)$$

$$\left[\frac{n}{2}\right]-1 \times \frac{\left\lceil \frac{n}{2} \right\rceil}{2}$$

$$\left\lceil \frac{n}{2} \right\rceil - \frac{1}{2} \times$$

$$\leq \frac{2C}{n}\left(\frac{n}{2}-1\right)\frac{n}{2}$$

$$\leq \frac{C}{2}\left(\frac{n}{2}-1\right) + O(n)$$

$$\leq \frac{3}{4} C \cdot n + O(n)$$

$$T(n) \leq \frac{3}{4} Cn + C_1 \cdot n$$

Assume $C \geq 4C_1$

$$\leq O(n)$$

$$C_1 \leq \frac{1}{4} C$$

11 Time complexity of randomized Selection Sort is n.

Page No.: YOUVA
Date:

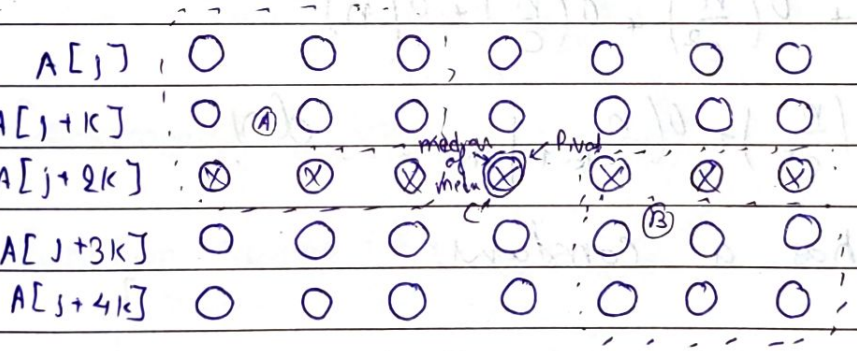
* Deterministic Approach of selecting problem.

A- { 3, 8, 1, 6, 10, 11, 5, 12, 7, 14, 13 }

Sorted = { 1, 3, 5, 6, 7, 8, 10, 11, 12, 13 }

we would choose median of all no so that partition is equal.

Step 1: Divide the nos in group of 5 elements $k = \lceil \frac{n}{5} \rceil$
Step 2: Find the median of each group (using insertion Sort)
Step 3: Recurse on $k = \lceil \frac{n}{5} \rceil$ median to find
Step 4: Partition the array with median of medians (pivot) (call partition function)
Step 5: if $i \leq q$ find the i^{th} key on left side
else find $(i-q)^{th}$ key on right side



This paradigm is called as Prune and Search.

$A \leq C \leq B$

$$T(n) \leq C \cdot \frac{n}{5} + C \cdot \frac{7n}{10} + O(n)$$

Assum $T(n)$ is order of n for $k<n$ we can say $T(k) \leq ck$

$T(n) = O(n)$
$T(n) \leq ck \quad k<n$

$T(n) \leq \frac{cn}{5} + c \cdot \frac{7n}{10} + O(n)$

$\leq c\frac{9}{10}n + O(n)$

$\leq c\frac{9}{10}n + c_1 n$

$\leq O(n)$

2nd Way :

$T(n) = T\left(\frac{n}{c}\right) + O(n)$ where $c > 1$.

$= T\left(\frac{n}{c^2}\right) + O\left(\frac{n}{c}\right) + O(n)$

$= T\left(\frac{n}{c^3}\right) + O\left(\frac{n}{c^2}\right) + O\left(\frac{n}{c}\right) + O(n)$

$= T(1) + O\left(\frac{n}{c^k}\right) + O\left(\frac{n}{c^{k-1}}\right) + \cdots O(n)$

Each O has a constant.

$\leq O\left(n + \frac{n}{c} + \frac{n}{c^2} + \frac{n}{c^3} + \cdots + \frac{n}{c^k}\right)$

$\leq O\left(\frac{n}{1 - \frac{1}{c}}\right)$

$\leq O(n)$

Start, Jump Step, End.

Procedure Insertion_Sort (P, k, n)

last = P + k;

while last ≤ n do

pos = last;

```
        while pos > p and A[pos] < A[pos-k] do
            A[pos] ⟷ A[pos-k]
            pos = pos - k.
        end while
        last = last + k
    end while
```

* A[ ] : | 30 | 20 | 15 | 35 | 40 | 10 |

insertion_Sort (1, 1, 6)

last = 2.

iterat; 1 :

pos = 2.

20 < 30

| 20 | 30 | 15 | 35 | 40 | 10 |

Deterministic Selection.

```
Function Select (p, r, i)
    if r - p + 1 = 50 then          (const)
        insertion_sort (p, 1, r)
        return A[p + i - 1];
    else
        k = ((r - p + 1) + 4) div 5;
        for j = 0 to k - 1 do
            insertion_Sort (p + j, k, r);
        end for
        med_med = Select (p + 2k, p + 3k - 1, k div 2);
        A[p] ⟷ A[med_med];
        q = Partition (p, r);
        if i ≤ q - p + 1 then
            return Select (p, q, i)
```

~~end if~~

else

return Select (q+1, r, i-[q-p+1])

end if

end if

$T(n) = a\,T(n/b) + O(n^d)$

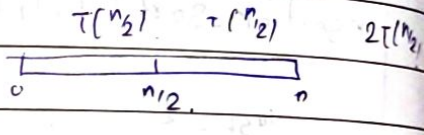
$n^{\log_b a}$ compare with n^d.

if $a < b$ (Overlap)

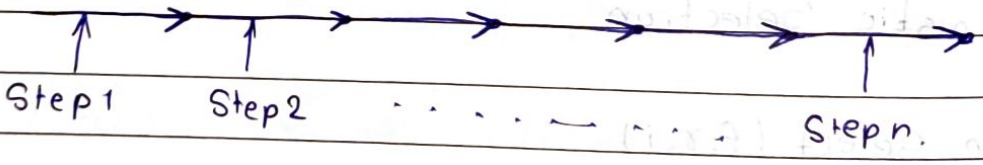
if $a = b$ (No Overlap)

if $a > b$ (Prune And Search) $T(n) = 2T(n/4) + O(n)$

(discard some part)

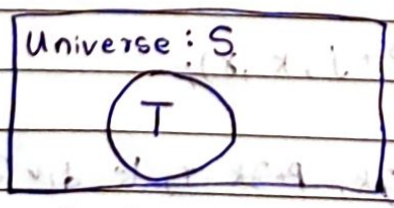


* Greedy Method → Search for Local Optimization.



Choose a subset T of S of objects A_i such that $\sum_{A_i \in T} w_i \leq W$ and $|T|$ is maximized.

$|T|$ = Cardinality (No of element in T).



$S = \{A_1, A_2 \ldots A_n\}$

each object has weight w_i

$A_1, A_2 \ldots A_n$

-- time Complexity → nlogn

Algo:

Sort So that

$\omega_i < \omega_j$ if $i < j$

$T = \phi$, $\omega(T) = 0$;

$i = 1$

while $\omega(T) + \omega_i \leq \omega$ do

$T = T \cup \{A_i\}$

$\omega(T) = \omega(T) + \omega_i$;

$i = i + 1$;

end while.

$A = \{A_1, A_2, A_3, A_4\}$

$\omega_1 = 20$, $\omega_2 = 5$, $\omega_3 = 25$, $\omega_4 = 1$.

Sorting as per weights.

A_4 A_2 A_1 A_3

$\omega_4 < \omega_2 < \omega_1 < \omega_3$

1 5 20 25

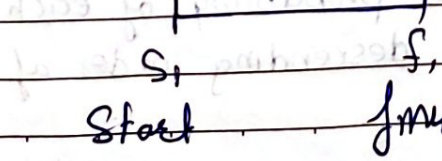
Scheduling Problem:

$S = \{1, 2, 3, \ldots, n\}$.

S = Set of Activities. ~~and finish~~

Each activity has a starting time & finishing time.

Constraint: We are having only one CPU which can run 1 process at a time.

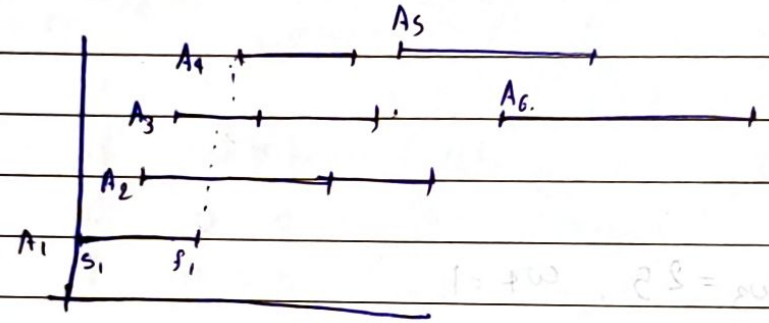


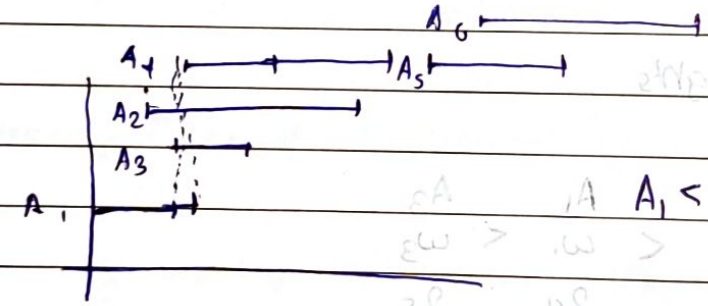
O(n log n)

Algo:

Sort so that $f_i \leq f_j$ if $i < j$

```
T = {1}; last = 1;
for i = 2 to n do
    if last ≤ s_i then
        T = T ∪ {i};
        last = i;
    end if
end for.
```





$A_1 < A_4 < A_3 < A_2 < A_5 < A_6$

$T = \{A_1, A_4, A_5\}$

Q. Write greedy algorithm to pair numbers that minimises maximum sum

[20, 1, 30, 15, 25]

1 36 20
30 45 45
(65) (45) (55)

$^6C_2 = \frac{6!}{2!\,4!} = \frac{6 \times 5}{2} = 15$

* Huffman Codes:

1. Count each symbol and find probability of each symbol
2. Arrange these symbols in descending order of probability / frequency.

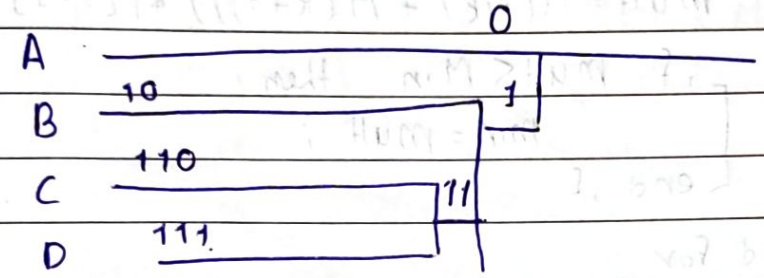
It is a lossless Compression.

Greedy Approach

3. Construct tree Structure to encode all Symbols

Input: C A B A A A B B A C D C B

Probabl- Frequency: A - 5/13, B - 4/13, C - 3/13, D - 1/13. } Sort. A - 5/13, B - 4/13, C - 3/13, D - 1/13.



* Dynamic Programming:

Exploding problem → Polynomial Time (n^c)

Catlan No = $\frac{1}{n}\binom{2n-2}{n-1}$

Matrix Chain Multiplication Problem.

Memory element is used to keep records

It is parentization problem

Given: $A_1, A_2 \ldots A_n$ (Input Matrices)

Parenthesize these matrices such that total no of multiplications will be minimum for all possible parenthesization.

$A_i \longrightarrow P_{i-1} \times P_i$