Date:
P. No:

Mid-sem - 30%
End-sem - 40%
Quiz(2) - 10%
Assignment - 10%
Class Performance - 10%

Analysis of Algorithm

- time* (limiting parameter) → machine-dependent
- space

Efficiency - Abstract notion of comparison (e.g. no. of steps)
"Steps" can be assignment operation, arithmetic or comparison. Depends on I/p size & other factors.

For a given input size, there's a worst case T.C.

Asymptotic Time Complexity: Ignores constant operations.

e.g.

```
max = A[0]                    -> once
i = 1                         -> once
while i < A.length            -> n times
    if A[i] > max             -> n times
        max = A[i]            -> will be executed 'n' times in worst cas & '1' time in the best one.
```

∴ no. of ~~steps~~ op.s in worst case = n + 2 + n + n
= 3n + 2 ≤ 4n

Date:
P. No:

Sorting Algorithm :-

$O(n^2)$ ⟶

$O(n \log n)$ ⟶

Let CPU speed $= 10^8$ ins./sec

Input size $= 10^9$

$\therefore O(n^2) \rightarrow \frac{10^{18}}{10^8} = 10^{10}$ sec.

$\approx$ 300 years

$O(n \log n) = \frac{10^9 \log 10^9}{10^8} = \frac{3 \times 10^{10}}{10^8}$

$\approx$ 5 min.

$O(1)$ → Independent of input size

$\log n$ →

$\sqrt{n}$

n

n^2

n

} Polynomial

2^n

$n!$

} Exponential

Date:
P. No:

input size	$\log(n)$	$\sqrt{n}$	n	n^2	n^3	2^n	$n!$
10	3	3	10	10^2	10^3	10^6	
10^2	6	10	10^2	10^4	10^6	10^{30}	10^{157}
10^3	9	30	10^3	10^6	10^9		↓
10^4	12	10^2	10^4	10^8	10^{12}		not
10^5	15	3×10^2	10^5	10^{10}	10^{15}		feasible
10^6	18	10^3	10^6	10^{12}	10^{18}		
10^7	21	3×10^3	10^7	10^{14}	10^{21}		
10^8	24	10^4	10^8	10^{16}	10^{24}		
10^9	27	3×10^4	10^9	10^{18}	10^{27}		

Input size :-

e.g. Arithmetics → no of digits in numbers is input size
Checking for prime → magnitude of number is the input size.

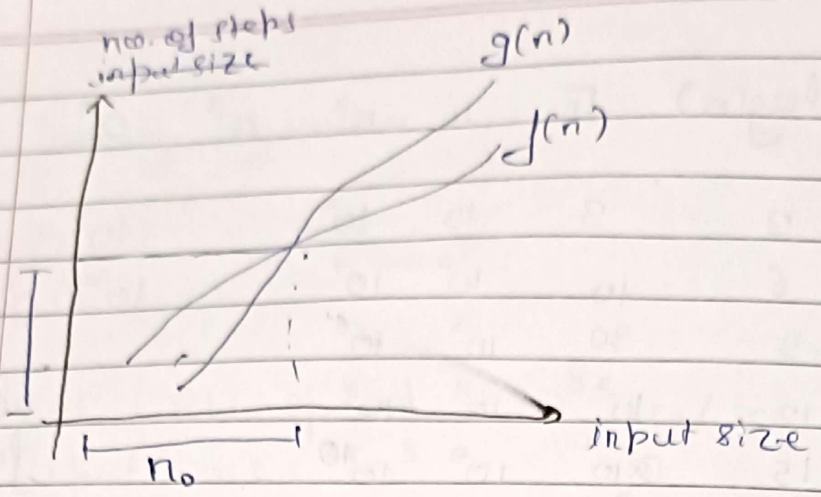
O-notation :-

$$O(g(n)) = \{ f(n) \mid \exists c, n_0 \in \mathbb{Z}^+, \text{ s.t. } 0 \le f(n) \le c \cdot g(n) \ \forall n \ge n_0 \}$$

e.g. $f(n) = n + 3\log n + 5$
$g(n) = n$

$\therefore f(n) \in O(g(n))$

Date:
P. No:



e.g. Show that $f(n) \in O(g(n))$, $f(n) = n + 3\log n + 5$.

$\leq 4n+5$

$n + 3\log n + 5 \leq 5n \quad \forall \quad n \geq 10$ [$\because \log n \leq n \quad \forall\ n \in \mathbb{N}$]

$\therefore c = 5, n_0 = 10$

★ Efficiency is measured using worst case (usually

→ Average-case complexity
→ Upper-bound
→ Lower-bound

Best-case analysis → Suitable for a problem (theoretically) but not an algori

Date:
P. No:

Dominating factor decides order notations:-

Q. $f(n) = 10n^2 + 7n + 8$. Show that $f(n) \in O(n^2)$

$O(g(n)) = \{ f(n) \mid \exists\, c \in \mathbb{R}^+, n_0 \in \mathbb{Z}^+, \forall\, n \geq n_0 \quad 0 \leq f(n) \leq c \cdot g(n) \}$

Proof $f(n) \leq 10n^2 + 7n^2 + 8n^2, \quad \forall\, n \geq 1$

$= 25n^2$

In general, $f(n) = an^2 + bn + c$

$\leq (a+b+c)n^2 \quad \forall\, n \geq 1$

since (n^2) is the dominating factor.

Q. Show that $n^3 \notin O(n^2)$

$\exists\, c, \quad n^3 \leq cn^2 \quad \forall \quad n \geq n_0$

Substitute $n = c+1$

Properties of O-notation:-

$f_1 \in O(g_1(n))$

$f_2 \in O(g_2(n))$

1) $f_1 + f_2 \in O(\max(g_1, g_2))$

Proof $f_1 \leq c_1 g_1, \quad n \geq n_1$

$f_2 \leq c_2 g_2, \quad n \geq n_2$

$n_3 = \max(n_1, n_2) \quad \forall\, n > n_3$

$f_1 + f_2 \leq c_1 g_1 + c_2 g_2 \leq c_3 (g_1 + g_2)$

$\leq 2c_3 \max(g_1, g_2)$

e.g 1 for i = 0 to n-1
    for j = i+1 to n-1
        if A[i] == A[j]
            return false
return true

→

Outer step	Inner iteration count
i = 0	n-1
= 1	n-2
= 2	n-3
⋮	⋮
= n-2	1
= n-1	0

$T(n) = 0 + 1 + 2 + \dots + (n-1)$

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

$\therefore O(n^2)$

e.g 2. count = 1
while n > 1
    count += 1
    n /= 2

→ Let $n = 2^i$

$n \to \frac{n}{2} \to \dots \frac{n}{2^i}$ ∵ 'i' times $(O(\log_2 n))$

Date:
P. No:

e.g 3.

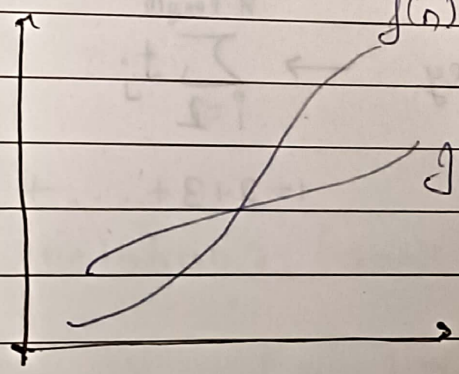
```
for i = 2 to A.length          -> C1 * n
    key = A[i]                 -> C2 x (n-1)
    j = i-1                    -> C3 x (n-1)
    while j >= 1 and key < A[j]  -> sum_{j=1}^{n} Cj x tj
        A[j+1] = A[j]          -> sum C5 x tj - 1
        j--                    -> sum C6 x tj - 1
    A[j+1] = key               -> C7 (n-1)
```

$O(n) \rightarrow O(n^2)$

∴ algorithm shouldn't be analysed using best-case time complexity.

Best-case time complexity :-

Ω-notation: $\Omega(g(n)) = \{f(n) \mid \exists\, c \in \mathbb{R}^+, n_0 \in \mathbb{Z}^+, \text{ s.t. } 0 < c\,g(n) \leq f$



Ω-notation is useful when $O(f(n))$ is also the same. Trivially $g(n)$ = constant is valid.

$2n+5 \leq 3n$

Date:
P. No:

$\therefore f(n) = O(g(n))$ & $f(n) = \Omega(g(n)) \rightarrow f(n) = \Theta(g(n))$

Ω-notation helps analyse theoretical lower-bound / characteristics of a problem.

e.g. Comp. based sorting algo. can't perform better than $O(n \log n)$.

→ When the best & avg. case analysis isn't feasible, amortized analysis is used.

★ $O(g(n)) = \{ f(n) \mid \exists\, n_0 \in \mathbb{Z}^+,\ c \in \mathbb{R}^+ \text{ s.t. } 0 \leq f(n) \leq c\,g(n)\ \forall n \geq n_0 \}$

Insertion sort:

```
for j = 2 to A.length
    key = A[j]
    i = j - 1
    while i > 0 and A[i] > key
        A[i+1] = A[i]
        i--
    A[i+1] = key
```

$\rightarrow \sum_{j=2}^{A.length} t_j$

$1 + 2 + 3 + \ldots +$

Date:
P. No:

Θ-notation :-

$$\Theta(g(n)) = \{f(n) \mid \exists\ n_0 \in \mathbb{Z}^+,\ c_1 \& c_2 \in \mathbb{R}^+ \text{ s.t. } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \quad \forall n \ge n_0\}$$

(*) $f(n) = \Theta(g(n))$ iff. $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

e.g. $\frac{n^2}{4} \le \frac{n^2}{2} - \frac{n}{2} \le \frac{n^2}{2} \rightarrow g(n) = n^2$ & $c_1 = \frac{1}{4}, c_2 = \frac{1}{2}$

Prove (*)

o-notation :- (small-O)

$$o(g(n)) = \{f(n) \mid \exists\ \mathbb{Z}^+ \text{ and } \forall c \in \mathbb{R}^+ \quad 0 \le f(n) < c\,g(n) \quad \forall n \ge n_0\}$$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

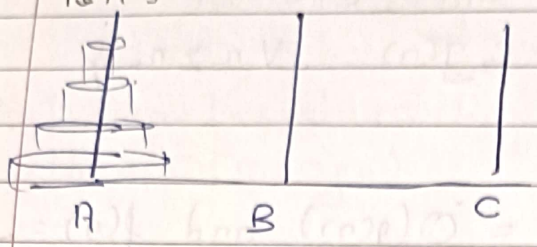
ω-notation :- (small-Ω)

$$\omega(g(n)) = \{f(n) \mid \exists\ \mathbb{Z}^+ \text{ and } \forall c \in \mathbb{R}^+ \quad 0 \le c\,g(n) < f(n) \quad \forall n \ge n_0\}$$

Date:
P. No:

Recursive :-

→ ToH :-



→ Move 'n' disks from A to B.
→ " 'n-1' disks from A to C.
→ " 1 disk from A to B.
→ " 'n-1' disks from C to B.

M(n) = # moves required to transfer 'n' disks.
$M(n) = M(n-1) + 1 + M(n-1)$
$M(1) = 1$

$M(n) = 2M(n-1) + 1$
$= 2[2M(n-2) + 1) + 1$
$= 2^2 M(n-2) + 2 + 1$
$= 2^3 M(n-3) + 2^2 + 2 + 1$
⋮
$= 2^j M(n-j) + 2^{j-1} + 2^{j-2} + \dots + 2 + 1$
$= 2^{n-1} M(1) + 2^{n-2} + 2^{n-3} + \dots + 1$
$= 2^{n-1} + 2^{n-2} + \dots + 2 + 1$
$= 2^n - 1$

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

Date:
P. No:

Recurrence Relations :-

e.g. $T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$ $\quad a \geq 1,\ b > 1,\ f(n)$ is non-negativ

$$\Rightarrow \Theta\left(n^{\log_b a}\right) + \sum_{j=0}^{\log_b(n-1)} a^j f\left(n/b^j\right)$$

$T(1) = \Theta(1)$ → base case

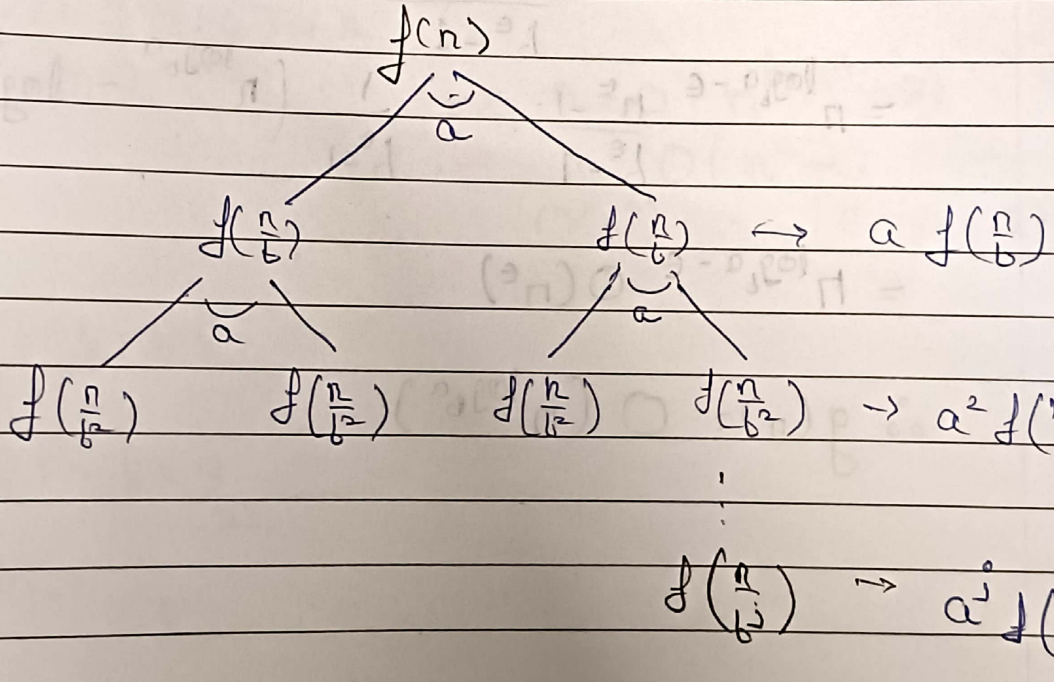
① - if $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for $\epsilon > 0$ $\quad T(n) = O\left(n^{\log_b a}\right)$

② - $f(n) = \Theta\left(n^{\log_b a}\right)$ then $\quad T(n) = \Theta\left(n^{\log_b a} \log_b n\right)$

③ - $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for $\epsilon > 0$ for $a f\left(\frac{n}{b}\right) \leq c f(n)$ for $c < 1$

then $T(n) = \Theta(f(n))$

Proof



j goes down to $\log_b n$

$\therefore a^{\log_b n} f\left(\frac{n}{b^i}\right)$

Date:
P. No:

(1). $g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$

$f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$

$g(n) = O\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon}\right)$

$= n^{\log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} \left(\frac{a b^\epsilon}{b^{\log_b a}}\right)^j$

$= n^{\log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} (b^\epsilon)^j$

$= n^{\log_b a - \epsilon} \cdot \frac{b^{\epsilon \log_b n} - 1}{b^\epsilon - 1}$

$= n^{\log_b a - \epsilon} \cdot \frac{n^\epsilon - 1}{b^\epsilon - 1} = \frac{1}{b^\epsilon - 1}\left(n^{\log_b a} - \log^{\log_b a - \epsilon}\right)$

$= n^{\log_b a - \epsilon} O(n^\epsilon)$

$\therefore g(n) = O(n^{\log_b a})$

Date:
P. No:

1. $T(n) = T(n-1) + n$
2. $T(n) = T(\lceil n/2 \rceil) + 1$
3. $T(n) = 9\,T(n/3) + n$
4. $T(n) = T(2n/3) + 1$
5. $T(n) = 3\,T(\frac{n}{4}) + n\log(n)$
6. $T(n) = 2\,T(\frac{n}{2}) + n\log(n)$

Ans ① - $T(n) = T(n-1) + n$

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

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

$= T(1) + 2 + 3 + \dots + n$

$= T(1) + 2 + 3 + \dots + n$

$= O(n^2)$

Ans ② - Put $n = 2^i$

Ans ③ - $T(n) = 9\,T(n/3) + n$

$f(n) = O(n^{\log_b a - \epsilon}) \rightarrow n = O(n^{\log_3 9 - \epsilon}) \rightarrow n = O(n^{2-\epsilon})$

$\therefore O(n^2)$ (ϵ is 1)

$a \geq 1$ & $b > 1$

Ans ④ - $a = 1, \; b = \frac{3}{2}$

$n^{\log_b a} = 1$

$f(n) = \Theta(n^{\log_b a})$

$\therefore T(n) = \Theta(\log_{\frac{3}{2}} n)$

Ans (5) $T(n) = 3T(\frac{n}{4}) + n\log(n)$

$a=3,\ b=4,\ \log_b a = \log_4 3 = 0.79$

$\therefore \Theta(n^{0.79})$ is the first term.

$n\log(n) = \Omega(n^{0.79+\epsilon})$

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

$a\,f(\frac{n}{b}) < c * f(n)$

* $3(\log\frac{n}{4})\frac{n}{4} < c(\log n)n$ for some $c < 1$.

$\boxed{c = \frac{3}{4}}$

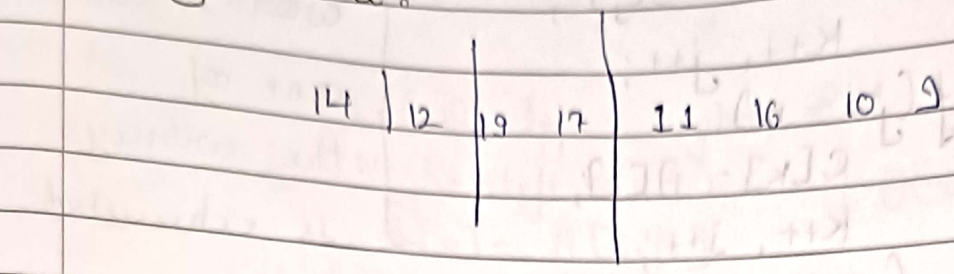
(6) $\Theta(n)$ can't find ϵ.

$n^{\log_b a + \epsilon} \rightarrow \frac{n^{\epsilon}}{\log n}$ (not asymptotically bounded.

Date:
P. No:

Divide and Conquer

Merge Sort :-




```
merge (A, m, B, n, C):
    i=0, j=0, k=0
    while (k < m+n)
        if (j==n or A[i] ≤ B[j]
            C[k++] = A[i++];
        else if (i==m or A[i] > B[j]):
            C[k] = B[j]
            k++, j++
```

D & C → divide into disjoint sub-prob., solve individually & merge to get sol^n of main problem.

- Merge Sort
- Quick Sort
- Max. array summation

→ Merging step: Use 3 integers.

```
Merge (A, m, B, n, C):
    i=0, j=0, k=0,
    while (k < m+n):
```

```
if (i == m)
    C[k] = B[j];
    k++, j++;
else if (j == n)
    C[k] = A[i];
    k++, i++;
else if ( A[i] ≤ B[j])
    C[k] = A[i]
    k++, i++;
else if (A[j] > B[j]
    C[k] = B[j]
    k++; j++;
```

works when one of the arrays is exhausted.

ex: (i) 5 7 9 (j=0)
2 4 6 (j=0)

Complexity of merge ops: $\Theta(m+n)$ → only one of the 4 comparisons is performed.

→ worst case: 7 operations
best case: 4 operations.

Prove $\Theta(m+n)$ (best & worst case).

Date:
P. No:

Inplace_merge :-
Increases time complexity due to shifting ops.

```
Merge_Sort (A, left, right, B)
    if (right - left == 1)
        B[0] = A[left]                    } O(1)
    if (right - left > 1)
        mid = (left) + (right - left)/2
T(n/2) -->  Merge Sort (A, left, mid, L)
T(n/2) -->  Merge Sort (A, mid, right, R)
O(n)   -->  Merge (L, (mid - left, R, right - mid, B)
```

arrays go from 'left' to 'right - 1'

Recurrence - relation :

- $T(n) = 2T(\frac{n}{2}) + O(n)$

Use - cases :-

- Distributed systems. (when the problem is unsolvable for one memory, it can be divided into multiple memories & recombined?
- Not useful for less memory

Since divide step precedes conquer step, merge-sort is always recursive.

Since partition step precedes yo call, quick-sort can be iterative.

Date:
P. No:

Quick-sort :-

1. Pivot
2. Partition
3. Complexity

pivot

→ | 20 | 10 | 16 | 25 | 19 | 46 | 81 | 7 |

PI UPI

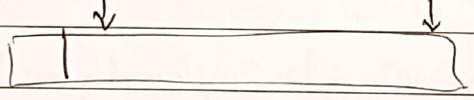
PI → partition indicator
UPI → unprocessed partition indicator

PI → indicates that left-sides have < elements.
UPI → indicates that left side is already processed.
Swapping → constant-time operation

∴ O(n). for scanning complexity.

Alternative

lower → ← upper



Complexity: If 'pivot' is the median, it divides the array in exactly equal halves.

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

However, finding median needs array to be sorted.

∴ Worst case: $T(n) = T(n-1) + O(n)$

$$= O(n^2)$$

e.g.

Date:
P. No:

e.g. → 5 7 2 4 } same relative position
→ 15 17 12 14 }
3 4 1 2

∴ n! possibilities of arrangement

Probability of one particular sequence: $\frac{1}{n!}$

∴ expected complexity = O(nlogn)

→ Almost sorted array is very infrequent.

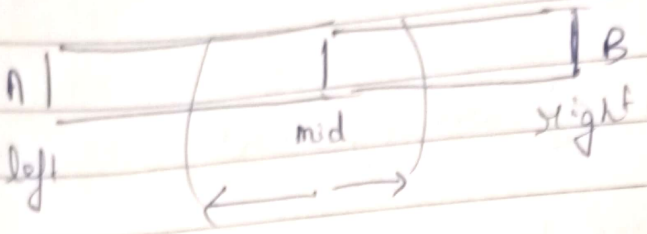
→ ~~divisions~~ partitions such as $\frac{n}{3}, \frac{2n}{3}$ are also nlogn.

∴ quicksort is fast in general.

Constant factors affect practical applications & Q.C. has low constant factor.

Date:
P. No:

Maximum sum subarray :-



3 cases: max-sum on left
max-sum on right
max-sum crossing mid.

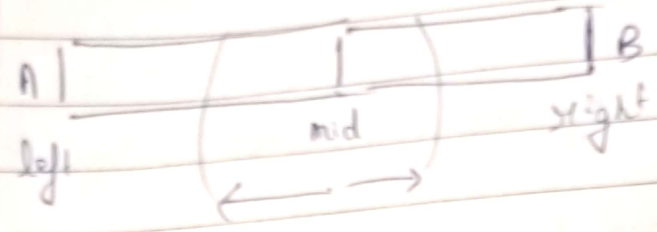
$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

Heap Sort →

1. Heap data structure :
a) Binary heap *
b) Binomial heap *
c) Fibonacchi heap *

Properties :-
→ Full binary tree → Levels upto l
(Structural property) ⇒ till (l-1) the tree is full and at l^{th} level, it is left full
→ Min-heap → Value at parent node less than value in child nodes.

Maximum sum subarray :-



3 cases: max-sum on left
max-sum on right
max-sum crossing mid.

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

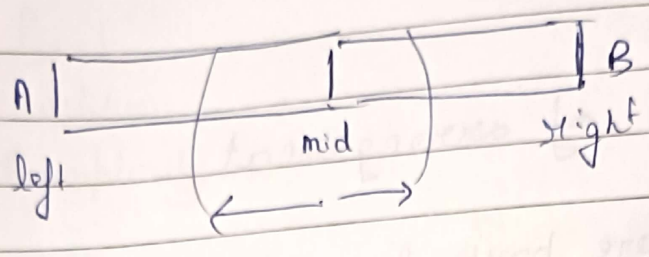
Heap Sort →

Q. Heap data structure :
a) Binary heap ★
b) Binomial heap ★
c) Fibonacchi heap ★

Properties :-
→ Full binary tree → Levels upto l
(Structural property) ⇒ till (l-1) the tree is full and at l^{th} level, it is left full
→ Min-heap → Value at parent node less than value in child nodes.

Date:
P. No:

Maximum sum subarray :-



3 cases: max-sum on left
max-sum on right
max-sum crossing mid.

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

Heap Sort →

1. Heap data structure :
a) Binary heap *
b) Binomial heap *
c) Fibonacchi heap *

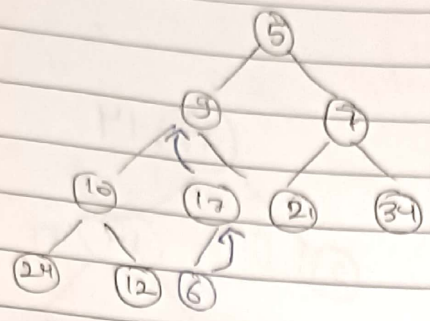
Properties :-

→ Full binary tree → Levels upto l
(Structural property) ⇒ till (l-1) the tree is full and at l^{th} level, it is left full

→ Min-heap → Value at parent node less than value in child nodes.

Date:
P. No:

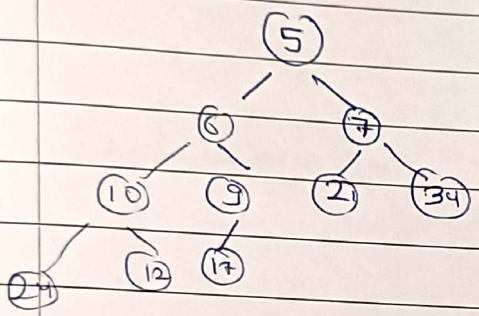
e.g



3	9	7	10	17	21	34	24	12	
1	2	3	4	5	6	7	8	9	10

child
parent : $2i, 2i+1$
parent : $\lfloor \frac{i}{2} \rfloor$

Array's used in place of L.L. since array allows random-access.



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

Insertion: $O(\log(n))$

$h = \lfloor \log(n) \rfloor$

Building heap from scratch: $\log 1 + \log 2 + \dots + \log n$

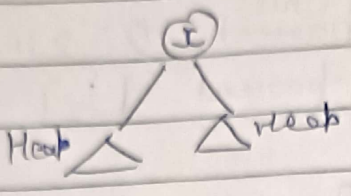
$= \log n!$

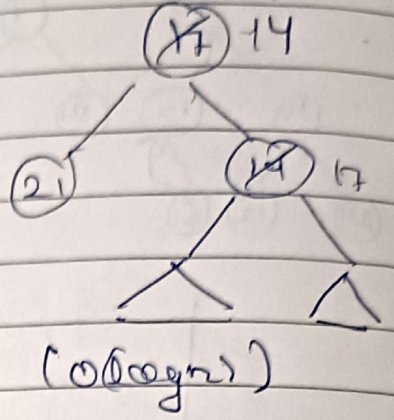
$= n \log(n)$

level	# nodes	more than half nodes are leaf $\rightarrow \frac{n}{2}\log(n)$
0	2^0	
⋮	2^1	
i	2^i	$2^{i+1}-1$ → # nodes in i^{th} level.

Date:
P. No:

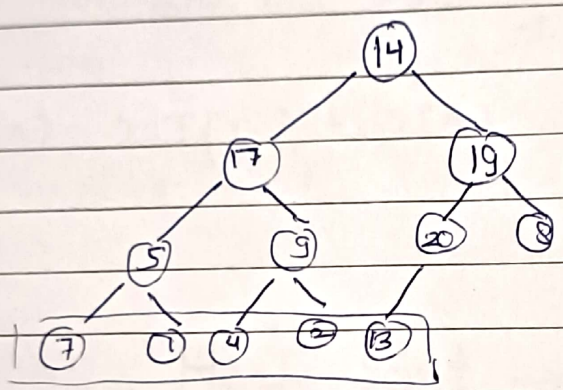
Heapify :-

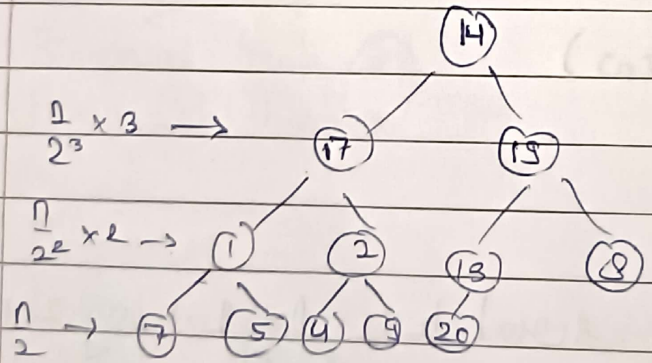




$(O(\log n))$

e.g:



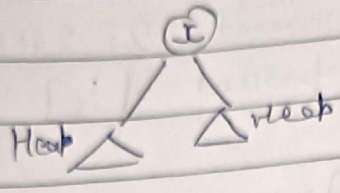


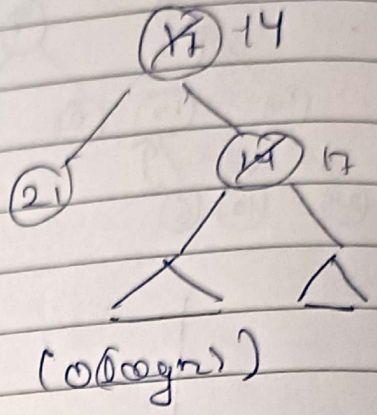
$$\frac{n}{2} + \frac{n}{2^2} \cdot 2 + \frac{n}{2^3} \cdot 3 + \cdots \frac{n}{2^i} \cdot i \qquad i = \log n$$

$$= n \sum_{i=1}^{\log n} \frac{i}{2^i}$$

Date:
P. No:

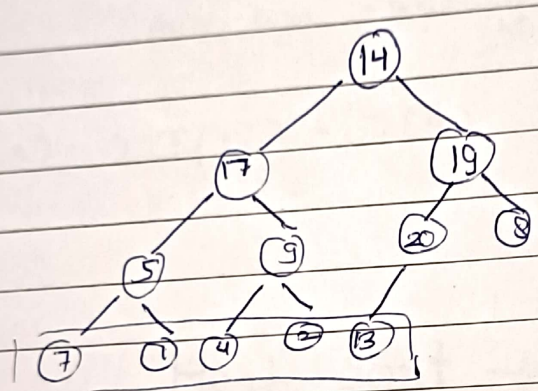
Heapify :-

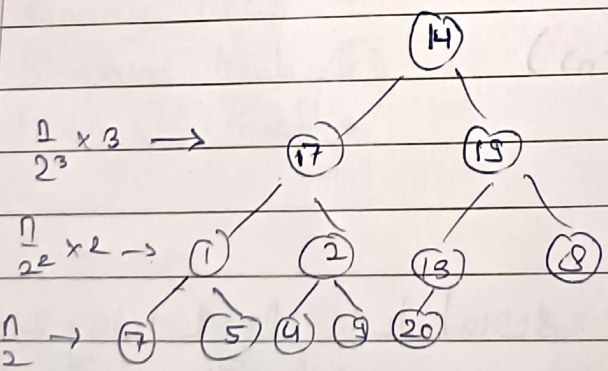




$(O(\log n))$

e.g.





$$\frac{n}{2} + \frac{n}{2^2} \cdot 2 + \frac{n}{2^3} \cdot 3 + \dots \frac{n}{2^i} \cdot i \qquad i = \log n$$

$$= n \sum_{i=1}^{\log n} \frac{i}{2^i}$$

Date:
P. No:

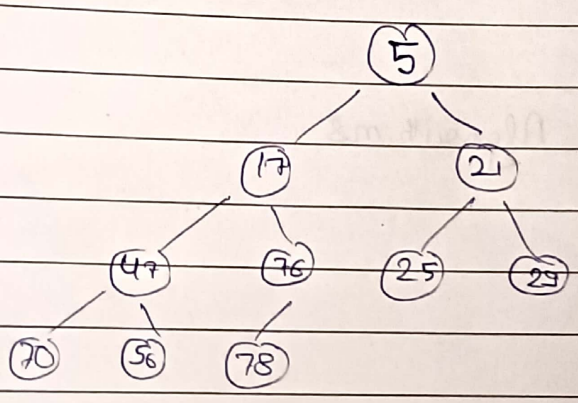
$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x} \quad [|x| < 1]$$

$$\sum_{i=1}^{\infty} i x^{i-1} = \frac{1}{(1-x)^2} \Rightarrow \sum_{i=1}^{\infty} i x^i = \frac{x}{(1-x)^2}$$

$$x = \frac{1}{2} \rightarrow \sum_{i=1}^{\infty} \frac{i}{2^i} = \frac{1/2}{1/4} = 2$$

$$\therefore \boxed{n \times \sum_{n=1}^{\log n} \frac{i}{2^i} \leq O(n)}$$

Delete-min :-



find-min → O(1)
find-max → O(n)
(need to search $\frac{n}{2}$ leaf nodes).

- Swap first and last → reduce heap-size by 1 → call heapify(0, n-1).
- Complexity → O(log n).

Heap sort :-

Build-heap O(n) + O(n log n).

Date:
P. No:

Preferably, quick-sort > heap-sort since bus-heap takes 2n ops.

Also, heap sort not suitable for distributed systems like merge sort.

Complexity of the problem: For any algorithm, complexity can't be better than $f(n)$.

Topics Left : • lower bound of sorting
• tutorial-session

Graph Algorithms

→ Map coloring
→ Route planning
→ Social-media groups
→ Problem solving
→ Graph Convolutional Network

$G = (V, E)$
$E \subseteq V \times V$

directed : $(u, v) \neq (v, u)$
weighted : $G = (V, E, W)$
$W: V \to R$ or $E \to R$

Missionary - Cannibal problem :-

(3, 3, 0)
↓ ↓ ↓
move cann. transition

possible transitions :
- (2,0), (1,0), (1,1), (0,1), (0,2)

goal : (0,0,1)

Here, the graph is implict - finite or infinite.

Explicit graph : (V, E)
↓
finite set of vertices given explicitly.

Can be represented using adj. list or matrix.

list → for sparse
mat → for dense

e.g. to find whether 'i' is the neighbour of 'j'.
adj. ~~list~~ mat. gives O(1).

To find all nbr. of 'i', adj. mat. gives O(N).

Adj matrix : Useful for vectorized operation (fast)

Breadth first search :-

adj. list : O(V+E) linear (w.r.t E) (every node will be checked twice)
adj. matrix : O(V²) quad. w.r.t. V.

$E \in [V-1 \rightarrow V^2]$

$O(b^d)$ for implicit graphs

Space complexity: Explicit → O(V)
Implicit → O(b^d)

Queue has a level at some point.

→ BFS gives optimal no. of edges

Depth First Search:-

Explicit or system stack can be used for implementation
recursion

```
1. visited[i] = 1
2.
3. DFS(i)
4.     visited[i] = 1
5.     for (i,j) ∈ E
6.         if visited[j] = 0
7.             visited[j] = 1
8.             DFS[j]
```

Time Complexity: Adj. mat. → O(V²)
Adj. list: O(V+E)

Space Complexity: Explicit: O(V)
Implicit: O(bd)

Disadvantages:
- Won't give optimal path length.
- Gets stuck for infinite graph.
→ Incomplete algorithm.

Date:
P. No:

Completeness of an algorithm → always provides a solution if ∃ one.

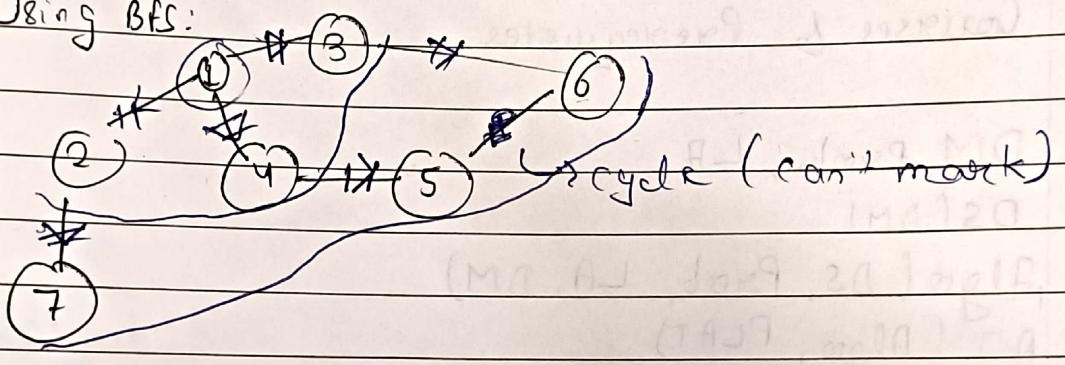
Advantages: Low space complexity.

→ Connectivity
→ Cycle

Cycle

e.g:

Using BFS:



DFS number: → pre_number
post_number

initially, count = 0

```
pre[i] = count++;
:
:
:
post[i] = count++;
```

Date:
P. No:

→ pre-post number → can detect cycle
→ to find articulation / cut point
→ to find the bridge

cut-point: removing the node disconnects the graph
bridge: remove the edge to disconnect the graph

Topological Sorting

e.g: Courses & Prerequisites

DM, Prob., LA
DS(DM)
Algo(DS, Prob, LA, DM)
AI(Algo, FLAT)
ML(AI, Algo, Prob, LA, DM, DC)

This one's a Directed Acyclic Graph (DAG).

To find 'longest path of the graph'.

Use queue
Push indegree[0] in queue.
Reduce indegree by 1.

Topological sort works in DAG.

Date:
P. No:

$LP(i) = 1 + \max[LP(j)] \quad \forall j$ s.t. $A[j,i] = 1$.

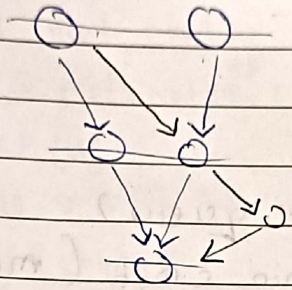
Time complexity: Adj. list. → $O(V+E)$

Adj. matrix → $O(V^2)$

$(i,j) \in E$:

indegree − − = 1

$LP[j] = \max(LP[j], 1 + LP[i])$;



Covers a level first

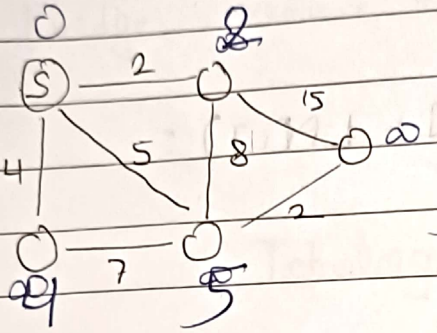
Longest path methods make sense in DAG since we're interested in dependencies.

Shortest path → normal graph since we can approach from any node.

Date:
P. No:

Shortest Path Algorithms.

→ Single source all pair shortest path algorithm



→ Use min-heap (Pr. queue).
→ Pop the v with min. cost (make sure to relax its neighbours).

↑ Dijkstra Algorithm

Given a weighted and directed graph $G=(V,E)$ and $W: E \to \mathbb{R}$, a path $P_{v_0 v_k}$ from v_0 to v_k is defined as

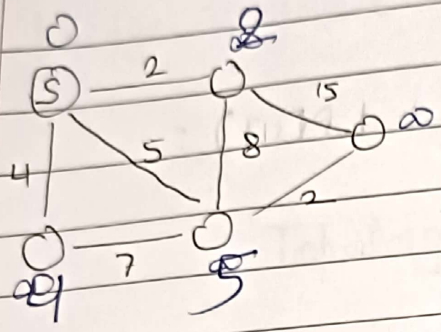
$$W(P_{v_0 v_k}) = \sum_{i=0}^{k-1} \omega(v_i, v_{i+1})$$

Shortest path $S(v_0, v_k) = \begin{cases} \min\{\omega(P_{v_0 v_k})\} \text{ if } \exists \text{ a} \\ \text{path from } v_0 \text{ to } v_k \end{cases}$

Date:
P. No:

Shortest Path Algorithms

-> Single source all pair shortest path algorithm



-> Use min-heap (Pr. queue).
-> Pop the v with min. cost (make sure to relax its neighbours).

↑ Dijkstra Algorithm

Given a weighted and directed graph $G=(V,E)$ and $W: E \to \mathbb{R}$, a path $P_{v_0 v_K}$ from v_0 to v_K is defined as

$$W(P_{v_0 v_K}) = \sum_{i=0}^{K-1} \omega(v_i, v_{i+1})$$

Shortest path $S(v_0, v_K) = \begin{cases} \min\{\omega(P_{v_0 v_K})\} \text{ if } \exists \text{ a path from } v_0 \text{ to } v_K \\ \infty \text{ otherwise} \end{cases}$

Date:
P. No:

Single Source Shortest path

SSP(G, ω, s)

for each vertex $v \in G.V$

$v.d = \infty$

$v.\pi = \phi$

$s.d = 0$

$Q = G.V$

$S = \phi$

while Q is not empty

$u \leftarrow Q.extractMin()$ //greedy

$S = S \cup \{u\}$

for all $v \in G.adj(u)$

if $v \notin S$

if $v.d > u.d + \omega(u,v)$ } Relaxation

$v.d = u.d + \omega(u,v)$

$v.\pi = u$

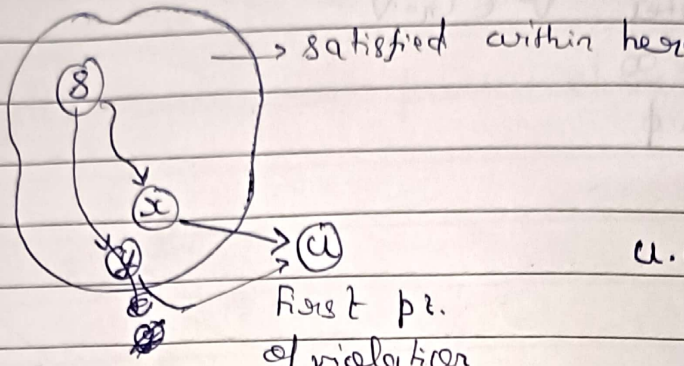
(1). Greedy Algorithm → Optimization problem (minimization)

(2). Correctness → Loop invariant

(3). Analysis

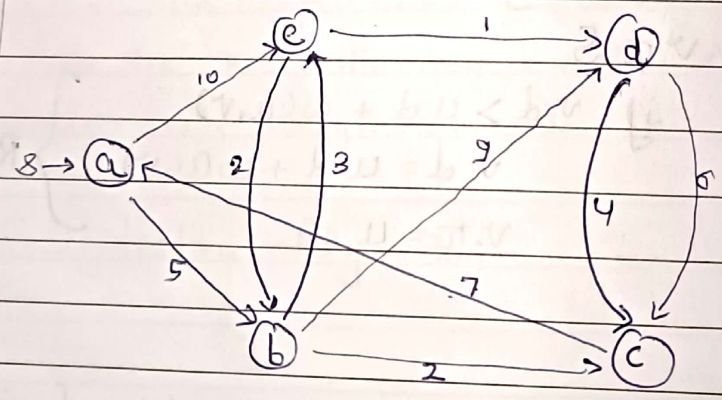
Date:
P. No:

Correctness → Upon extracting a node, it should has the shortest path until now.



$u.d = \delta(s,x) + w(x,y)$

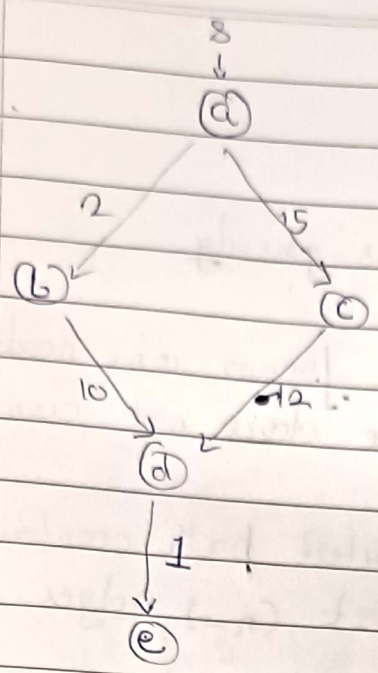
Ques.



a	b	c	d	e
0	∞	∞	∞	∞
0	5	∞	∞	10
0	5	7	14	8
			13	8
			9	

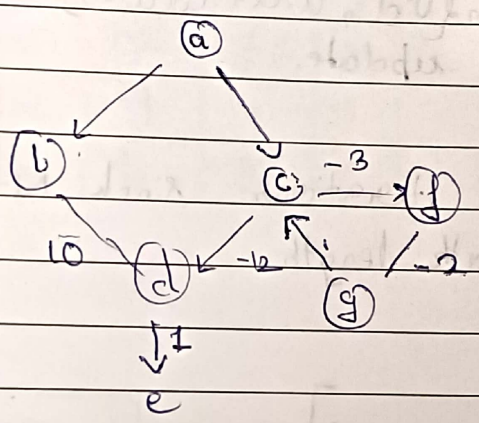
Date:
P. No:

Ques



a	b	c	d	e
0	∞	∞	∞	∞
	2	15	∞	∞
		15	12	∞
		15		13
		15		

Negative Cycle :-



Time complexity of Dijstra :- $n(\log(n)) + E \log(n)$

$\Rightarrow V(\log V) + E \log V$

$= (V+E) \log V$

$V \log V \rightarrow$ extract min

$E \log V \rightarrow$ relaxation

Fibonacci heap $\rightarrow O(|E| + |V| \log |V|)$

Date:
P. No:

Bellmanford Algorithm :-

Dijkstra may fail cause its greedy.

Bellmanford: ① Shortest path from one node to another node does not contain any cycle.

Corollary: Shortest path contains at-most $(n-1)$ edges.

② $s \longrightarrow v_1 \longrightarrow v_2 \cdots v_m \longrightarrow t$

$s \rightsquigarrow t$ is shortest $\Rightarrow s \rightsquigarrow v_i$ is also shortest.

$v.d = \cancel{v.d} \min\{v.d, u.d + c(u,v)\}$

→ spurious update.

① Needs atmost $N-1$ iterations since $N-1$ is maximum possible path length.

```
for each v ∈ G.V                          ┐
    v.d = ∞                               │
    v.π = ϕ                               │ initialization {O(n)}
                                          │
s.d = 0.                                  ┘

for i = 1 to |N-1|  → O(n³) or O(n|E|)
    for (u,v) ∈ G.E                       ┐
        if v.d > u.d + c[u,v]             │ O(|V|²)
            N.d = u.d + c[u,v]            │ or
                                          │ O(|E|)
```

Date:
P. No:

Bellman ford Algorithm :-

Dijkstra may fail cause its greedy.

Bellmanford: (1). Shortest path from one node to another node does not contain any cycle.

Corollary: Shortest path contains at-most $(n-1)$ edges.

(2). $s \longrightarrow v_1 \longrightarrow v_2 \cdots v_m \longrightarrow t$

$s \rightsquigarrow t$ is shortest $\Rightarrow s \rightsquigarrow v_i$ is also shortest.

$v.d = \min\{v.d, u.d + c(u,v)\}$

→ spurious update.

(1). Needs atmost $N-1$ iterations since $N-1$ is maximum possible path length.

```
for each v ∈ G.V
    v.d = ∞
    v.π = ϕ
s.d = 0.
```

} initialization {O(n)}

```
for i = 1 to |V|-1        → O(n³) or O(n|E|)
    for (u,v) ∈ G.E
        if v.d > u.d + w[u,v]
```

} O(|V|²) or

Bellmanford Algorithm :-

Dijkstra may fail cause its greedy.

Bellmanford: ① Shortest path from one node to another node does not contain any cycle.

Corollary: Shortest path contains at-most (n-1) edges.

② (s) → (v_1) → (v_2) ... (v_m) → (t)

(s) ⇝ (t) is shortest ⇒ (s) ⇝ (v_i) is also shortest.

$v.d = \min\{v.d, u.d + c(u,v)\}$

→ spurious update.

① Needs atmost N-1 iterations since N-1 is maximum possible path length.

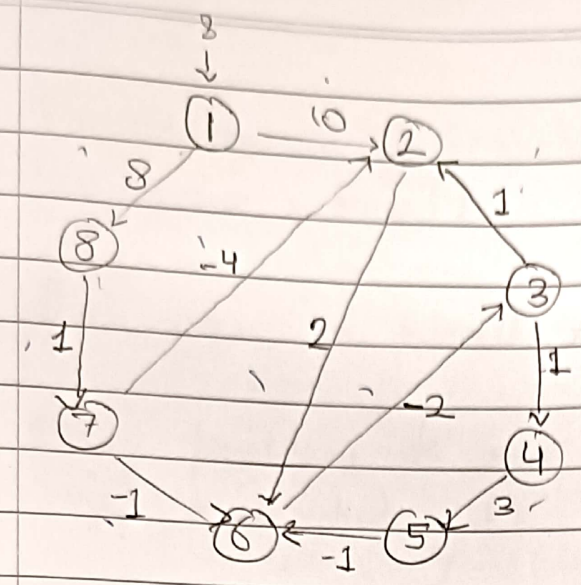
```
for each v ∈ G.V                          ┐
    v.d = ∞                               │
    v.π = ϕ                               │ initialization {O(n)}
                                          │
s.d = 0.                                  ┘

for i = 1 to |V|-1  → O(n³) or O(n|E|)
    for (u,v) ∈ G.E                       ┐
        if v.d > u.d + w[u,v]             │ O(|V|²)
            v.d = u.d + w[u,v]            │ or
            v.π = u                       │ O(|E|)
```

$\infty + \infty = 16$

Date:
P. No:

Eg:



v \ it.	1	2	3	4	5	6	7
1	0	0	0	0			
2	10	10	5	5			
3	∞	10	5	5			
4	∞	∞	11	6			
5	∞	∞	14	9			
6	∞	12	~~8~~ 7	7			
7	∞	9	9	9			
8	8	8	8	8			

Correctness :-

Use induction → Let's assume its correct till $u_1, u_2 \ldots u_i$ which are the ~~neighbours~~ direct parents of v.

∴ It'll be correct for 'v' in the next step.

N-1 iterations are needed since N-1 is max. path length for shortest path.

Time Complexity :-

→ $O(|V||E|)$ adj. list

→ $O(|V|^3)$ adj. matrix

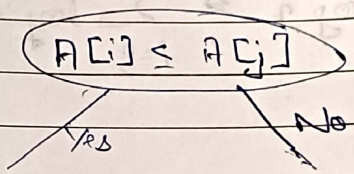
All pair shortest path :

Eric Demon MIT OCW

Date:
P. No:

(1). Lower Bound of Searching and Sorting :-
(Comparison Based)

- Best case = Worst Case → Avg.
- Many problems do not have well-defined lower bound (not researched yet).
- Comparison-based (swapping is constant time) sorting is analogous to decision tree.



Decision Tree	Algorithm
→ Internal node	Comparison
→ Leaf node	Solution
→ path from leaf to node	One execution of the algorithm.
→ path length	Running time
→ Height of tree	Worst case running time

Leaf of decision tree for Sorting ~~heap~~ like $A[5] \leq A[2] \leq A[1] \leq A[4] \leq A[3]$ for 5 values.

∴ n! leaf nodes.

$\log(n!) \rightarrow O(n \cdot \log n)$

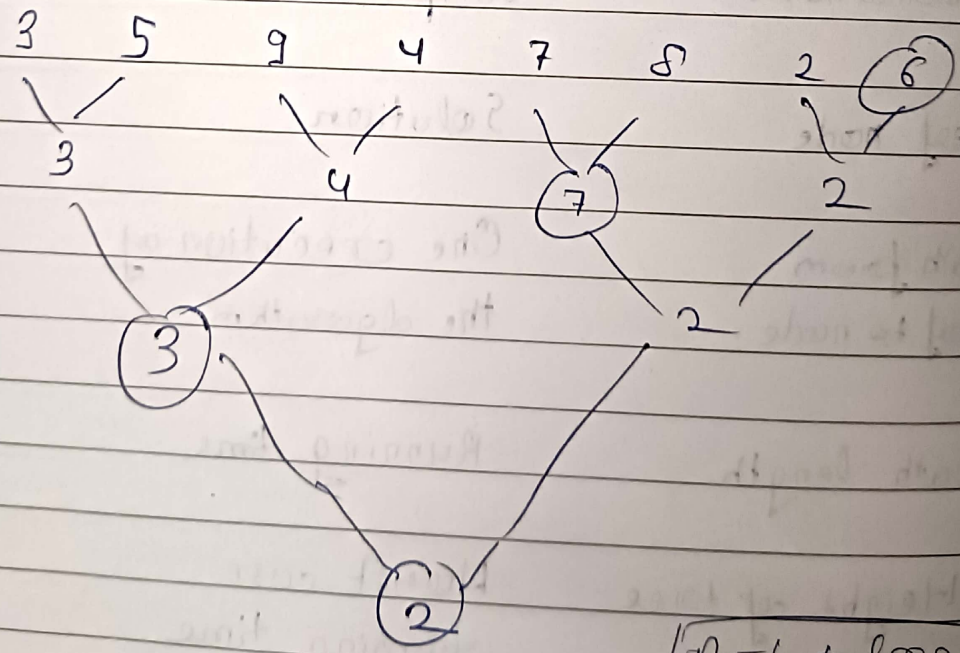
$\rightarrow \log(n(n-1)(n-2)\cdots 1)$

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

$\rightarrow \sum_{i=1}^{n} \log i \geq \sum_{i=\frac{n}{2}}^{n} \log \frac{n}{2}$

Second Minima :-

1. Tournament solution:



$n-1+\log n$

$O(n+\log n)$

Date:
P. No:

② . K^{th} minima :-

Quick select

~~Max-heap~~

③ . Articulation point :-