Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS 081
FoE&T Semester: VI Session: 2021-22
Jamia Millia University, New Delhi - 110025

NAME : YASH VINAYVANSHI

PROGRAM NAME : B. TECH (COMPUTER ENGINEERING)

SEMESTER : 6

EXAMINATION ROLL. NO. : 19BCS081

UNIQUE PAPER CODE : CEN - 601

PAPER TITLE : ANALYSIS & DESIGN OF ALGORITHMS

NO. OF PAGES :

DATE OF EXAM : 01/06/2022

TIME OF EXAM : 10AM - 1PM.

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS 081
FoE&T Semester: 6 Session: 2021-22
Jamia Millia University, New Delhi - 110025

Q 1 b (i) $T(n) = 3T(n/4) + n^2$

$T(n) = aT(n/b) + f(n)$. using Master Method

$a = 3, \; b = 4, \; f(n) = n^2$.

$n^{\log_b a} = n^{\log_4 3} \approx n^{0.792}$

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

$n^2 = \Theta(n^{0.792 + \varepsilon}) \rightsquigarrow \varepsilon > 0$.

$\therefore$ case 3

$af(n/b) \leq cf(n)$

$3f(n/4) \leq c \cdot n^2$.

$3 \times \frac{n^2}{16} \leq c \cdot n^2$.

$c \geq 3/16 \rightsquigarrow c < 1$ ✓

$\therefore T(n) = \Theta(f(n)) = \underline{\Theta(n^2)}$.

(ii) $T(n) = 2T(\sqrt{n}) + 1$.

using change of variables.

$m = \log n \rightsquigarrow n = 2^m$.

$T(2^m) = 2T(2^{m/2}) + 1$.

Let $T(2^m) = S(m)$

$S(m) = 2T(m/2) + 1$.

using Master's method,

$T(n) = aT(n/b) + f(n)$

$a = 2, \; b = 2, \; f(n) = n^0$.

$n^{\log_b a} = n^{\log_2 2} = n^1$.

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

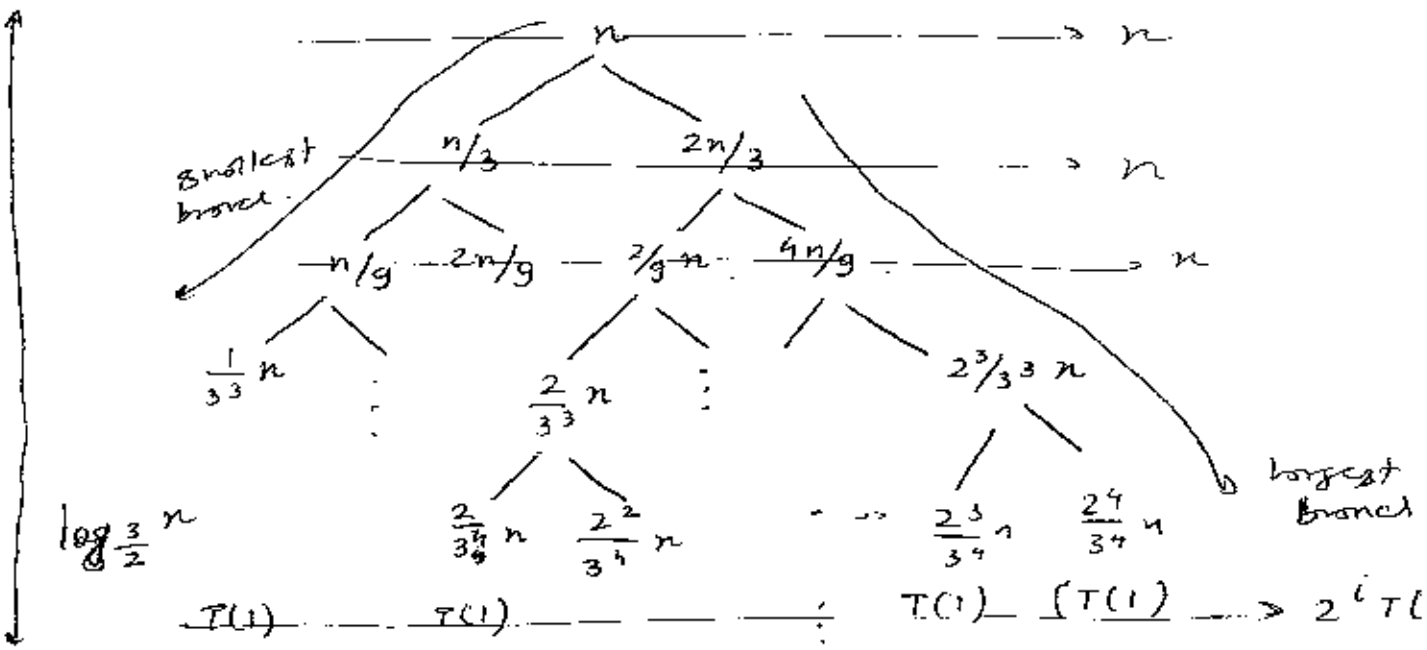
$n^0 = \Theta(n^{1-\varepsilon}) \quad \varepsilon > 0$.

$\therefore$ case 1

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

Q 1(b)(ii) $T(n) = T(n/3) + T(\frac{2n}{3}) + n$

using recursion tree method



$T(n/3^i) = 1$ ← terminates first.

$T(\frac{2^i n}{3^i}) = 1$ ← longest branch.

ht of tree $= \log_{3/2} n$.

$T(n) = n + n + n + \ldots \log_{3/2}$ times $+ (2^i T(1))$

$= n \log_{3/2} n + 2^{\log_{3/2} n}$.

$= n \log_{3/2} n + n^{\log_{3/2}}$ const.

$\boxed{T(n) = O(n \log n)}$

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS 084
FoE&T Semester: 6 Session: 2021-22
Jamia Millia University, New Delhi - 110025

Q1 c) (i) RAM (random access Model) of computation and reference model for Algorithm analysis:

→ instructions executed serially, no concurrency
→ Each
  Arithmetic (+, −, *, /, %, ⌊ ⌋, ⌈ ⌉, ...)
  Data movement (load, store, copy)
  control (conditional, unconditional branch, subroutine call & return)
  Instruction take a constant amount of time
→ Data types are integer & floating point.
  word size ~ c log n bits ($c \geq 1$) for inputs of size n.
→ other instructions such as exponentiation, shift left etc in RAM graduate.
→ memory hierarchy is not considered.

```
array Function (A, n)
I/p : Array A of integers
O/p : A.
1.  a = 0.
2.  for i = 0 to n-1 do.
3.      a = a + A[i].
4.      A[i] = a/(i+1).
5.  return A.
```

primitive operations.

1 (control jump)
3 (=, jump, ++) × n
3 (+, =, [i]) × n
4 (+, =, /, [i]) × n
1.

Total primitive operations.

$= 1 + 3n + 7(n-1) + 1$

$= \boxed{10n - 5}$

Q1 c(ii) Binary Search Recurrence equation.

```
BinSea (A, l, r, key) {
    if (l > r) return;
    mid = l+r/2
    if A[mid] == key
        return mid;
    if (A[mid] < key)
        return BinSea(A, l, mid, key);
    return BinSea(A, mid+1, r, key);
}
```

→ We match the key with middle element of sorted array A, if it matches we found the key → return mid index

→ If key < A[mid] → the key shall exist in first half of array, otherwise in second half of array

→ With each recursive call, we reduce the search space by half.

∴ Divide : To compare A[mid] with key → O(1)
Conquer : In each iter^n, search space halves → T(n/2)
Combine : No combination → 0 cost.

∴ recurrence rel^n for binary search is

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T(n/2) + \Theta(1) & \text{otherwise} \end{cases}$$

→ $\boxed{T(n) = \Theta(\log n)}$

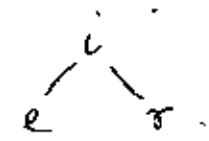
→ The best case complexity of bin search is $\boxed{\Omega(1)}$ when mid of array matches with the key in first step

→ The worst case complexity occurs when either the key is not found or found at the last recursion. At of recursion tree in bin search is $\log n$ in worst case → $\boxed{T(n) = O(\log n)}$

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No : 19BCS081
FoE&T Semester: 6 Session: 2021-22
Jamia Millia University, New Delhi - 110025

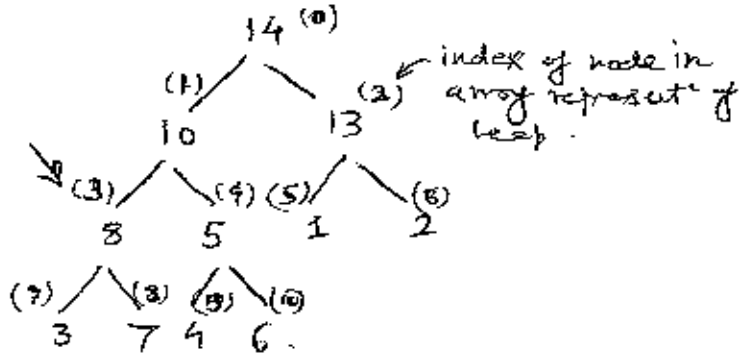
Q2 a) (i)

```
MAX-HEAPIFY (A, i)
1.  l = LEFT (i).          // return index of left child of i
2.  r = RIGHT (i)
3.  if l ≤ A.heapsize & A[l] > A[i]
4.        largest = l
5.  else largest = i
6.  if r ≤ A.heapsize & A[r] > A[largest].
7.        largest = r.
8.  if largest ≠ i
9.        exchange A[i] with A[largest].
10.       MAX-HEAPIFY A(largest)
```



finds largest of i, l, r
if largest ≠ i → heap property violated → exchange i with largest of its children & max-heap in subtree rooted at new root

Step 1



→ The Algorithm terminates after step 1 bec subtree rooted at index 3 is already a max heap.

i = 3
l = left (3) = 7 (2i + 1)
r = right (3) = 8. (2i + 2)
A[7] = 3 > A[3] = 8 No → largest = 3.
A[8] = 7 > A[largest] = A[3] = 8 No → largest = 3.
largest = i = 3 → No swap required.

Applying MAX-HEAPIFY in bottom up way for all internal nodes.

2a (ii)

BUILD-MAX-HEAP(A)
1. A.heapsize = A.length.
2. for i = ⌊A.length/2⌋ down to 1.
3. MAX-HEAPIFY(A, i)

Analysis

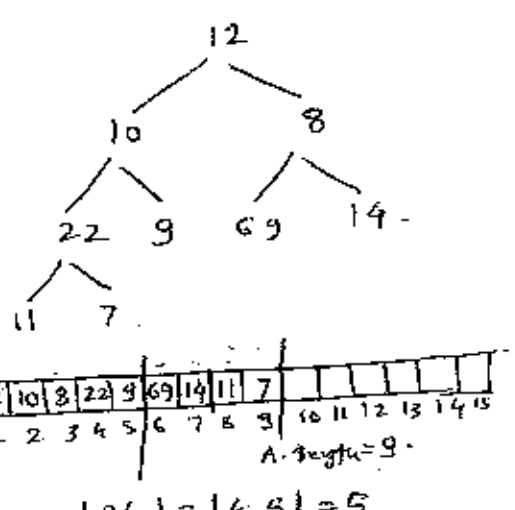
$$T(n) = \sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h).$$

(No. of levels in a heap; No. of Nodes at level h; time required to maxheapify a node at level h)

$$= O\left(n \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h}\right) \quad \text{removed ceil factor}$$

$$\leq O\left(n \sum_{h=0}^{\infty} h\left(\tfrac{1}{2}\right)^h\right) \qquad \sum_{k=0}^{\infty} k x^k = \frac{x}{(1-x)^2}$$

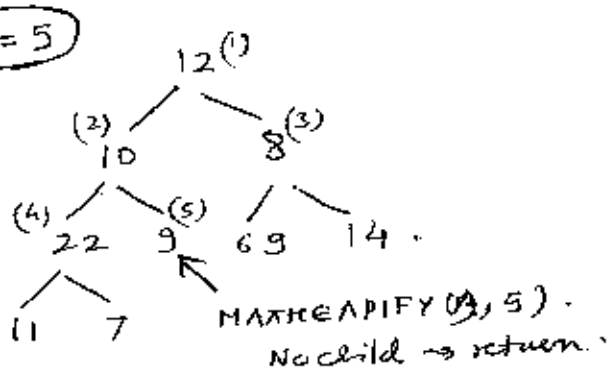
$$= O\left(n \frac{1/2}{(1-1/2)^2}\right) = \boxed{O(n)}$$



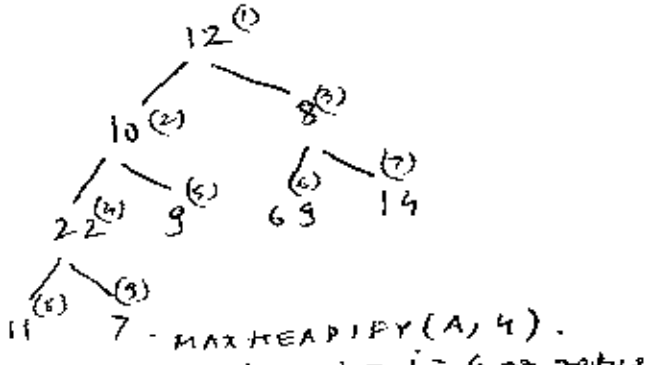
A: | 12 | 10 | 8 | 22 | 9 | 69 | 14 | 11 | 7 | | | | | | |
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
A.length = 9.

⌊9/2⌋ = ⌊4.5⌋ = 5.

i = 5

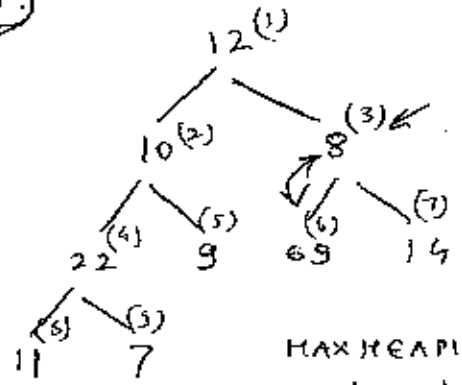


i = 4



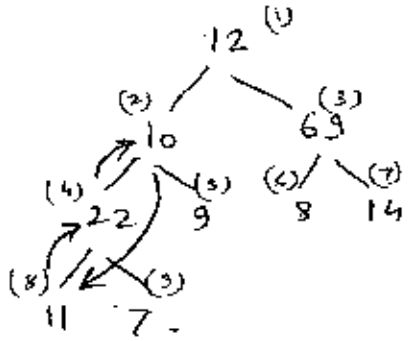
Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS081
FoE&T Semester: 6 Session: 2021-22
Jamia Millia University, New Delhi - 110025

i = 3.



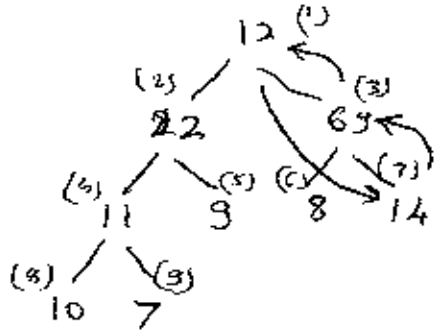
MAXHEAPIFY (A, 3)
largest = 6 ≠ 3 → swap A[3], A[6].
MAXHEAPIFY (A, 6).
largest = 6 = i → return

i = 2

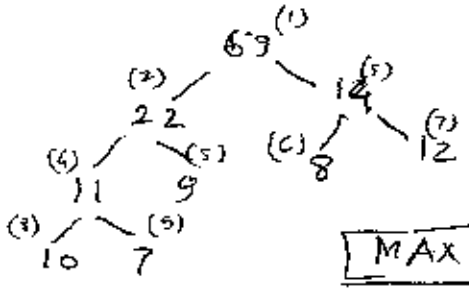


MAXHEAPIFY (A, 2)
largest = 4 ≠ 2 → swap (A[2], A[4])
MAXHEAPIFY (A, 4).
largest = 8 ≠ 4 → swap (A[4], A[8]).
MAXHEAPIFY (A, 8)
No children → return.

i = 1

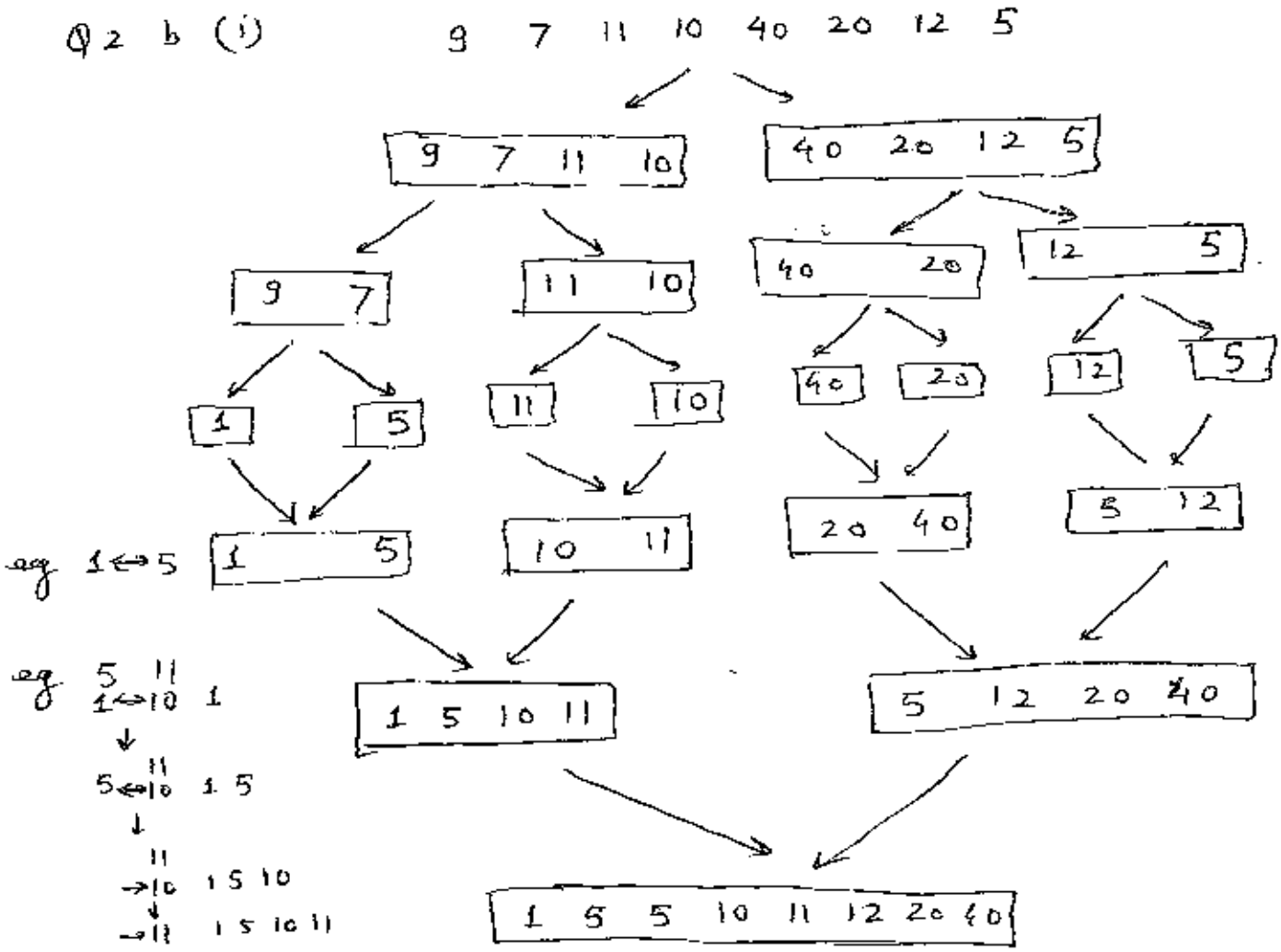


MAXHEAPIFY (A, 1).
largest = 3 ≠ 1 → swap A[1] & A[3]
MAXHEAPIFY (A, 3).
largest = 7 ≠ 3 → swap A[3] & A[7]
MAXHEAPIFY (A, 7)
No children → return



MAX HEAP obtained

Q2 b (i)



eg 1 ⇔ 5

eg 5 11
1 ⇔ 10 1
↓
5 ⇔ 11 1 5
↓
11
→ 10 1 5 10
↓
→ 11 1 5 10 11

eg. 1 ⇔ 5
5 12
10 20
11 40 1

5 ⇔ 5
5 12
10 20 1 5
11 40

5
10 ⇔ 12
20
11 40 1 5 5

12
10 20 1 5 5 10
11 40

12
11 ⇔ 20 1 5 5 10 11
40

12
20
40 1 5 5 10 11 12 20 40.

Best case complexity of Merge Sort.

$T(n) = 2T(n/2) + n$ (n no of comparisons instead of $n+m$)

$T(n) = aT(n/b) + 1$.

$a = 2, b = 2, f(n) = n^1$.

$n^{\log_b a} = n^1$. $\epsilon = 0$.

∴ case 2

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

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS081
FoE&T Semester: 6 Session: 2021-22
Jamia Millia University, New Delhi - 110025

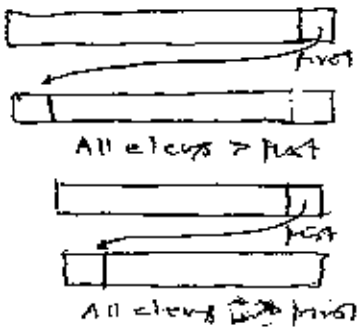
Q2 b) (ii)

```
QUICKSORT (A, l, r).
1. if p < r
2.     q = PARTITION (A, l, r)
3.     QUICKSORT (A, l, q-1)
4.     QUICKSORT (A, q+1, r).

PARTITION (A, l, r).
1.  x = A[r]
2.  i = l-1
3.  for j = l to r-1.
4.      if A[j] ≤ x
5.          i = i+1.
6.          exchange A[i] with A[j]
7.  exchange A[i+1] with A[r]
8.  return i+1
```

Worst case complexity of quicksort.

worst case partitioning : unbalanced partitioning at each recursive call.



ie Array is reverse sorted.

$T(n) = \cancel{T(n-1)}^{\text{divide}} + \cancel{T(0)}^{\text{conquer}} + \cancel{\Theta(n)}^{\text{combine}}$.

$= T(n-1) + \Theta(n)$.

(using iterative method).

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

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

$\vdots$

$= T(k) + n(n-k+1) \ldots n-2 + n-1 + n$

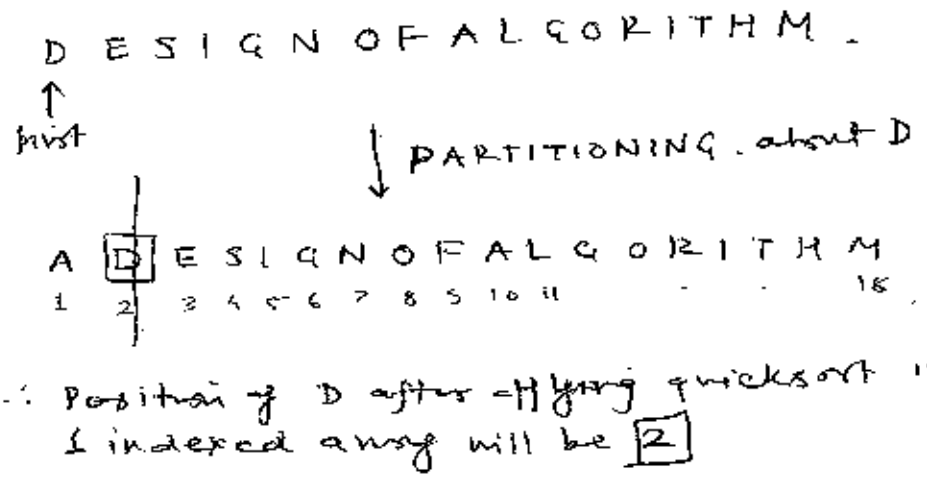
$k = 1$

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

$= 1 + 2 + 3 + \ldots n$

$= \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ or

$= \boxed{O(n^2)}$



∴ Position of D after performing quicksort in 1 indexed array will be 2

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS 081
FoE&T Semester: 6 Session: 2021-22
Jamia Millia University, New Delhi - 110025

Q 3 a) (i) DFS (depth first search) is a graph traversal algorithm, which starting from a source vertex s, penetrates the graph as deep as possible until a dead end emerges due to No more further choices to extend, which can occur if either the degree of current vertex is 1 or all its neighbours are visited. Once a dead end emerges, the DFS backtracks the path it has traced until further choices to extend appear. This process is repeated until all vertices are ~~visited~~ discovered.

The predecessor subgraph of a DFS forms a depth first forest.

DFS colors vertices during the search to indicate their state and also timestamps each vertex.

v.color = White (initially)
grey (when discovered)
black (when finished i.e its adj list is examined completely).
v.d = time when v is first discovered
v.f = time when v is completely examined.

DFS has several applications
- Finding connected components.
- Topological sort on DAGs.
- Finding strongly connected components.
- Finding Articulation pts in a graph
- Finds bridges in a graph, etc
⋮

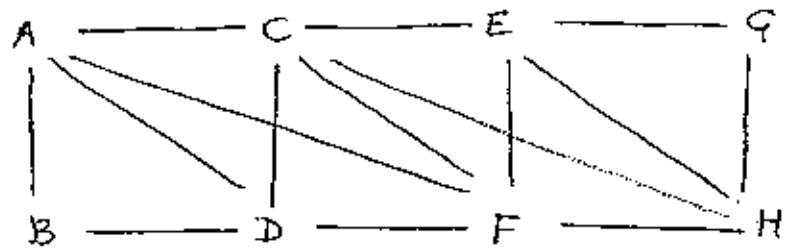
DFS classifies the edges of a graph in 3 categories.

```
DFS (G)
1.  for each vertex u ∈ G.V
2.        u.color = white.
3.        u.π = NIL
4.  time = 0.
5.  for each vertex u ∈ G.V.
6.      if color == WHITE
7.          DFS-VISIT(G, u).
```

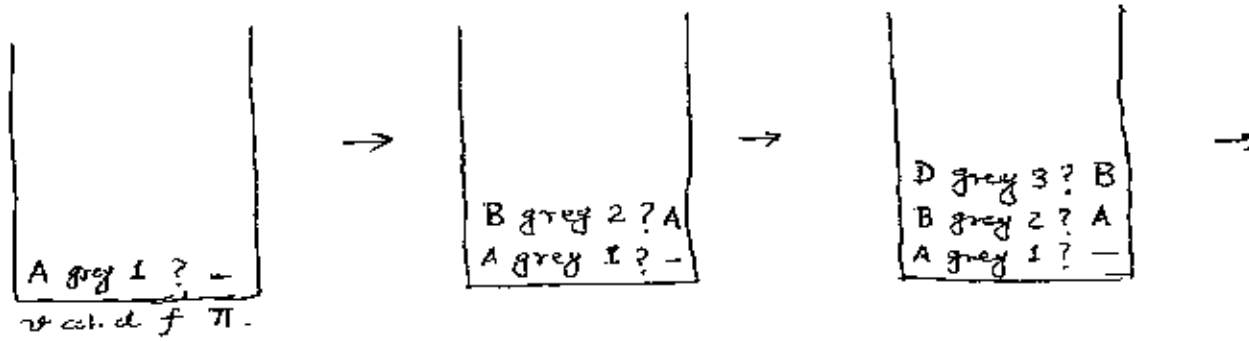
Analysis
$T = O(V+E)$.
$S = O(V)$
At max V vertices can be stack
(if graph contains hamiltonian path)

```
DFS-VISIT(G, u).
1.  time = time + 1.
2.  u.d = time.
3.  u.color = grey.
4.  for each v ∈ G.Adj[u].
5.      if v.color = WHITE.
6.          v.π = u.
7.          DFS-VISIT(G, v).
8.  u.color = BLACK
9.  time = time + 1.
10. u.f = time
```

This is the recursive version of DFS, we can show its implementation by using an implicit stack

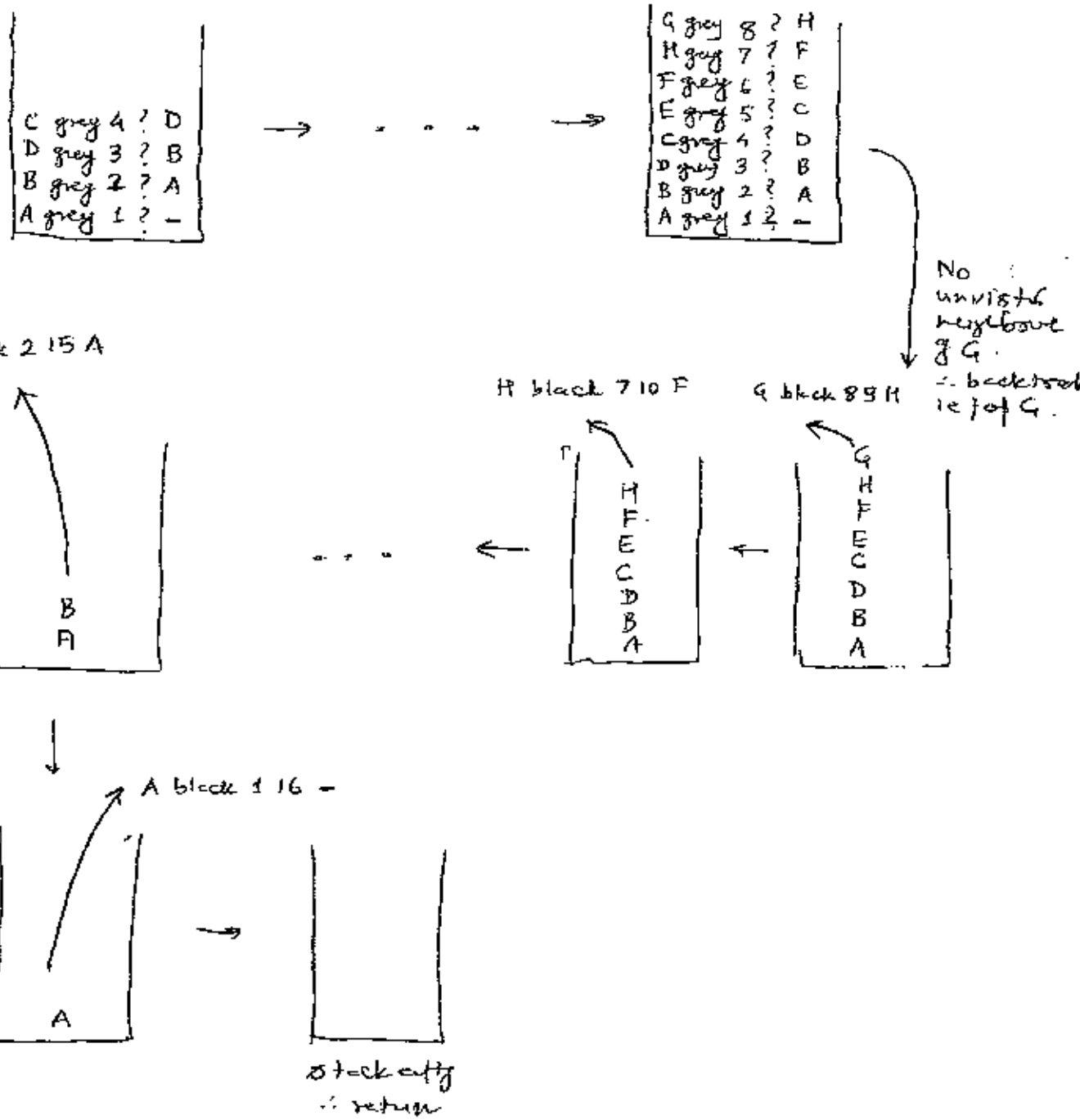


stack.

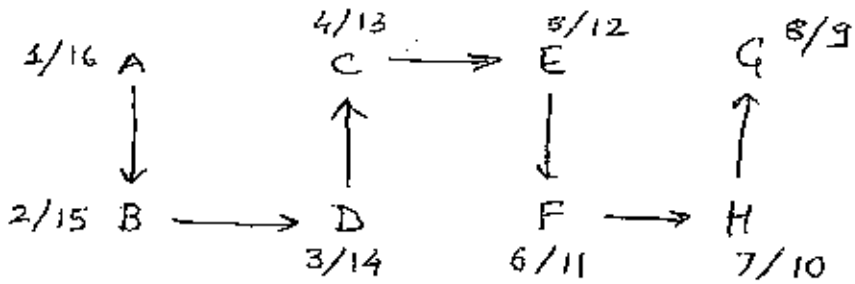


v col. d f π.

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS 081
FoE&T Semester: 6 Session: 2021-22
Jamia Millia University, New Delhi - 110025



The depth first tree obtained is.

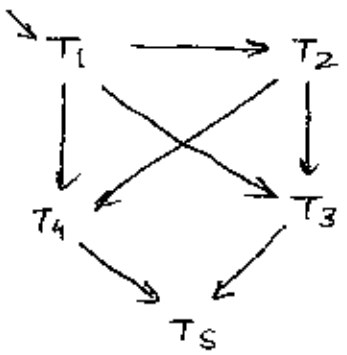


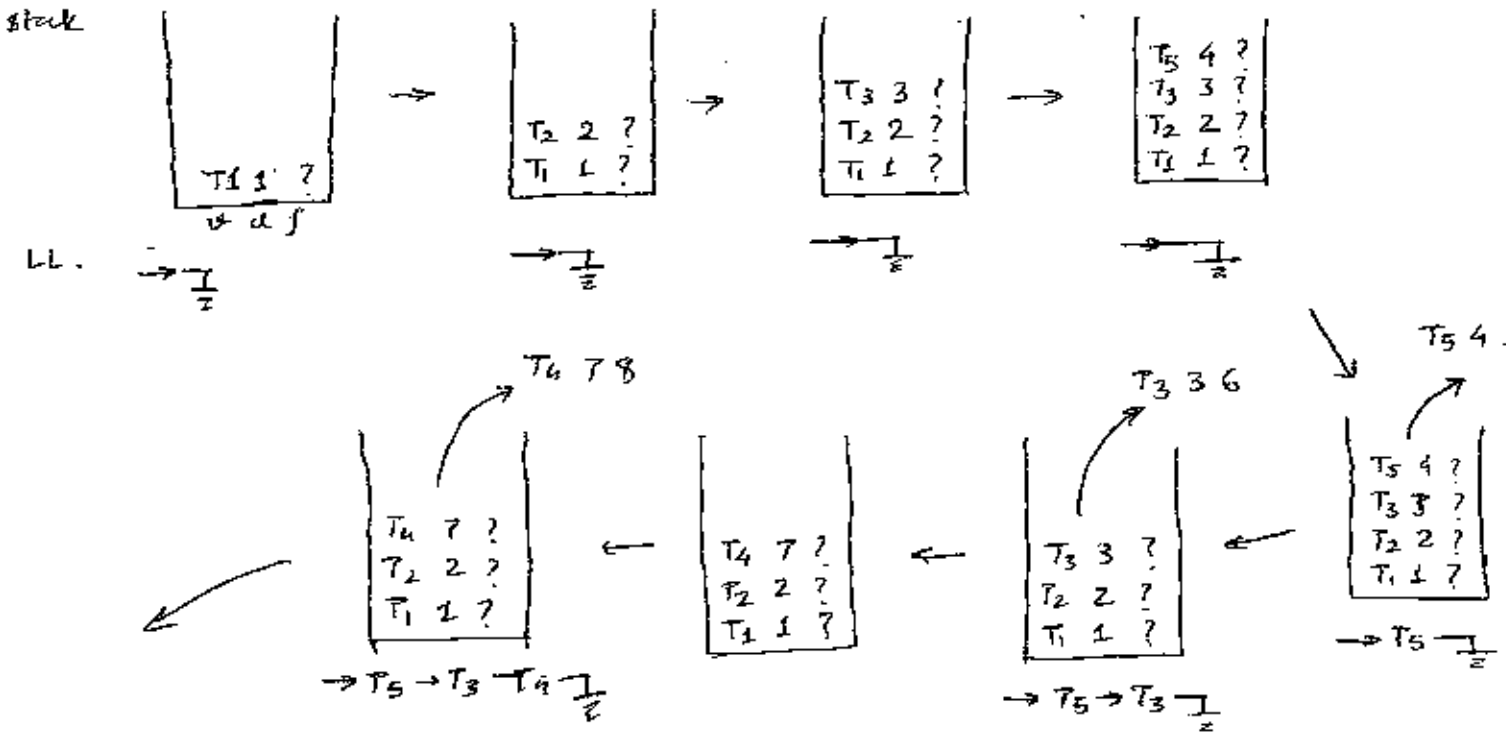
Q3 a (ii) TOPOLOGICAL-SORT(G).

1. call DFS(G) to compute finishing times $v.f$ for each vertex v.
2. as each vertex is finished, insert it in front of a linked list.
3. return the LL of vertices.

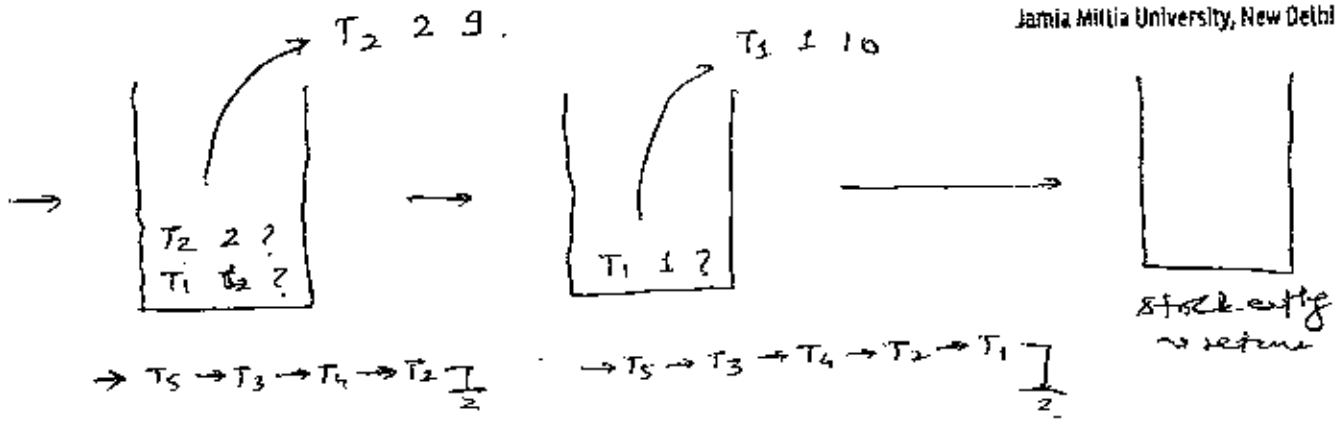
Topologically sorts vertices effectively in reverse order in LL.

$T = O(V+E)$ [$O(V+E)$ for DFS & $O(V)$ insertions in LL of cost $O(1)$ each





Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS 081
FoE&T Semester: 6 Session: 2021-22
Jamia Millia University, New Delhi - 110025



$\rightarrow T_5 \rightarrow T_3 \rightarrow T_4 \rightarrow T_2 \rightarrow$ (2) $\rightarrow T_5 \rightarrow T_3 \rightarrow T_4 \rightarrow T_2 \rightarrow T_1 \rightarrow$ (2)

∴ One of the Topological sorts of given DAG is,

$$\boxed{T_1 \rightarrow T_2 \rightarrow T_4 \rightarrow T_3 \rightarrow T_5}$$

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS 08 1
FoE&T Semester: 6 Session: 2021-22
Jamia Millia University, New Delhi - 110025

Q 3 c)

```
MST-PRIM(G, w, s)
1.  for each u ∈ G.V
2.      u.key = ∞
3.      u.π = NIL
4.  s.key = 0.
5.  Q = G.V
6.  while Q ≠ ∅.
7.      u = EXTRACT-MIN(Q)
8.      for each v ∈ G.adj[u]
9.          if v ∈ Q & w(u, v) < v.key.
10.             v.π = u
11.             v.key = w(u, v).
```

s : arbitrary source vertex.

G : graph $G = (V, E)$ — V: set of vertices; E: set of edges, $E \subseteq V^2$

u, v : vertex. has two attributes
1. value key
2. parent vertex π.

Q : minimum priority queue (implemented using min heap)

w : weight function $w : V \times V \to \mathbb{R}$ or $E \to \mathbb{R}$.

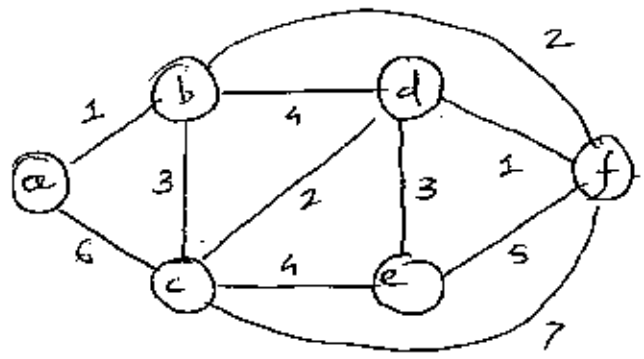
Analysis.

1. line 1 to 3 initializes each vertex	$O(V)$
2. line 4 is single assignment	$O(1)$
3. line 5 is implemented using BUILD-MIN-HEAP() func^n	$O(V)$.
4. line 7 runs EXTRACT-MIN() func^n on heap which is logarithmic in size of heap & since while loop runs (V) no. of times	$V * O(\log V)$
5. line 8 runs in total for $\sum_{v=0}^{V-1} \lvert adj[v_i] \rvert = O(E)$.	$O(E)$.
6. line 11 is implemented using DECREASE-KEY() taking log time & it runs in total $O(E)$ times as in for loop on line 8.	$E * O(\log V)$

∴ Total time = $O(V \log V + E \log V) = \boxed{O(E \log V)}$

Assuming $E = O(V^2)$ in worst case (for complete graph)

$$\boxed{T(n) = O(E \log V) = O(V^2 \log V)}$$



Minimum spanning tree

	a	b	c	d	e	f
key	∞	∞	∞	∞	∞	∞
π	–	–	–	–	–	–

key	0	∞	∞	∞	∞	∞
π	–	–	–	–	–	–

$u = \min(0, \infty, \infty, \infty, \infty, \infty) = a$

1 vs ∞ 6 vs ∞

key	1	6	∞	∞	∞	a
π	a	a	–	–	–	

$u = \min(\infty, 1, 6, \infty, \infty, \infty) = b$

6 vs 4 ∞ vs 4 ∞ vs 2

key	4	4	∞	2	a 1 b
π	b	b	–	b	

$u = \min(\infty, \infty, 4, 4, \infty, 2) = f$

4 vs 7 4 vs 1 ∞ vs 5

key	4	1	5	a 1 b 2 f
π	b	f	f	

$u = \min(\infty, \infty, 4, 1, 5, \infty) = d$

4 vs 2 5 vs 3

key	2	3	a 1 b 2 f, f 1 d
π	d	d	

$u = \min(\infty, \infty, 2, \infty, 3, \infty) = c$

4 vs 3

key	3	a 1 b 2 f, f 1 d, d 2 c
π	d	

$u = \min(\infty, \infty, \infty, \infty, 3, \infty) = e$

key	Q empty.	a 1 b 2 f, f 1 d, d 2 c, c 3 e
π		

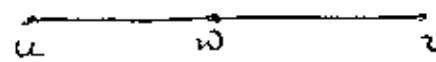
Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS 081
FoE&T Semester: 6 Session: 2021-22
Jamia Millia University, New Delhi - 110025

Q 4) a) (i) Characteristics of Dynamic programming.

1. Optimal Substructure.

A problem exhibits optimal substructure if optimal solution to the problem contains within it the optimal solutions to subproble

eg. in a graph, shortest path problem has optimal substr



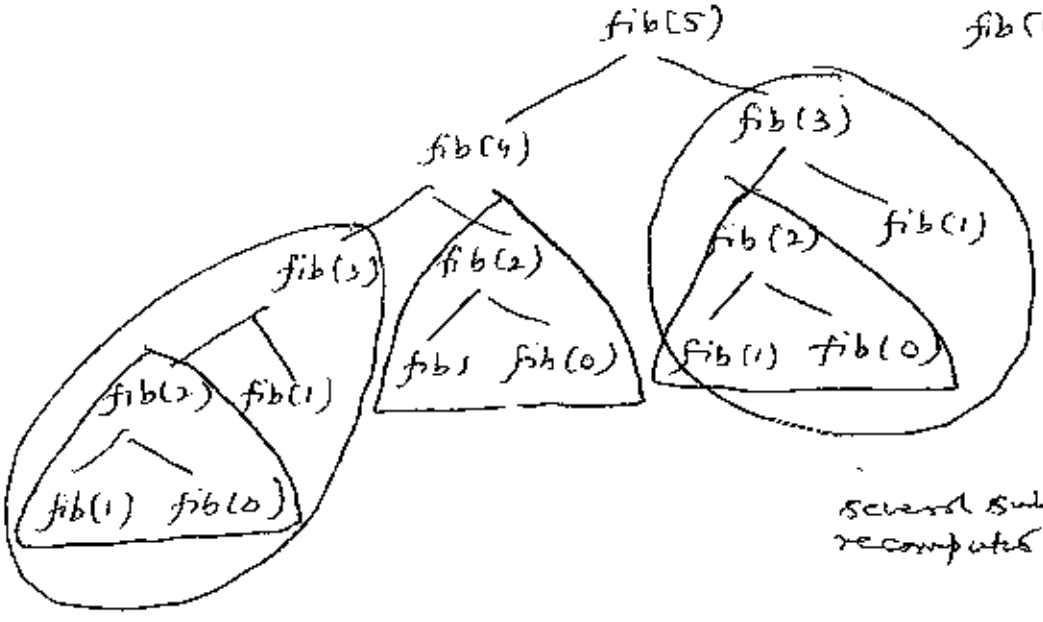
if $p^*(u, v)$ is shortest path from u, v & $w \in p^*(u, v)$ then $p(u, w)$ & $p(w, v)$ are shortest paths from u, w & w, v as well.

2. overlapping subproblems

The space of subproblems must be small in the sense that a recursive algorithm for problem solves the same subproblems over & over rather than always generating new subproblems.

eg. recursive generation of fibonacci nos

$fib(n) = \begin{cases} 1 & \text{if } n = 0 \\ fib(n-1) + fib(& \text{otherwise} \end{cases}$



Several subproblems are recomputed eg fib(3) 2 times fib(2) 3 times

Limitations of Dynamic programming

→ In combinatorial problems which do not obey optimal substructure like longest path problems, dynamic programming cannot yield a polynomial time solution.

4a)(ii) Fibanacci series using bottom up DP (Tabulation)
(1 dimensional DP)

```
int fib(int n){
    int dp[n+1];
    dp[0] = 0;
    dp[1] = 1;
    for(int i=2; i<=n; i++)
        dp[i] = dp[i-1] + dp[i-2]
    return dp[n]
}
```

T = O(n) for loop runs n-1 times, body is O(1)
S = O(n) for dp array.

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS085
FoE&T Semester: 6 Session: 2021-22
Jamia Millia University, New Delhi - 110025

Q 4 c) **Problem stmt**

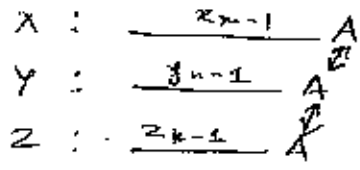
LCS pblm : Given two sequences $X = \langle x_1, x_2, \ldots, x$ and $Y = \langle y_1, y_2, \ldots, y_m \rangle$, find the maximum length subsequence of X &

Problem characterisⁿ

(1) LCS has optimal substructure property.

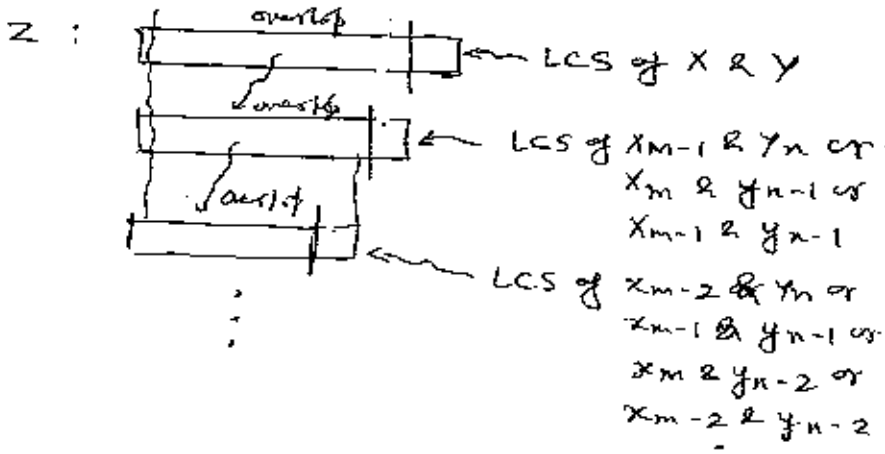
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$
$Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences.
& $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be LCS of X, Y.

1. if $x_m = y_n$, then $z_k = x_m = y_n$ & Z_{k-1} is LCS of x_{m-1} & y_{n-1}.



2. if $x_m \neq y_n$, then $z_k \neq x_m \Rightarrow Z$ is LCS of X_{m-1} & Y
3. if $x_m \neq y_n$, then $z_k \neq y_n \Rightarrow Z$ is LCS of X & Y_{n-1}.

(2) LCS has overlapping subproblems property.
LCS of two sequences contains within it the LCS prefixes of the two sequences.



length of X

length of Y.

(3) LCS has only $O(mn)$ distinct subproblems. Hence we can efficiently 2 dimensional bottom up DP to solve the problem efficiently.

Dynamic programming algorithm for LCS

```
LCS-LENGTH(X, Y)
1.  m = X.length
2.  n = Y.length.
3.  let b[1..m, 1..n] & c[0..m, 0..n] be new tables
4.  for i = 1 to m
5.      c[i, 0] = 0
6.  for j = 0 to n
7.      c[0, j] = 0
8.  for i = 1 to m                                   = O(m)
9.      for j = 1 to n                               = m O(n)
10.         if xi == yj
11.             c[i, j] = c[i-1, j-1] + 1
12.             b[i, j] = "↘"
13.         else if c[i-1, j] ≥ c[i, j-1]
14.             c[i, j] = c[i-1, j].
15.             b[i, j] = "↓"
16.         else c[i, j] = c[i, j-1].
17.             b[i, j] = "→".
18. return c, b.
```

$T(n) = O(m \times n)$

$S(n) = \Omega(m n)$

To print LCS

```
PRINT-LCS(b, X, i, j).
1.  if i == 0 or j == 0
2.      return.
3.  if b[i, j] = "↘"
4.      PRINT-LCS(b, X, i-1, j-1).
5.      print xi.
6.  else if b[i, j] == "↓"
7.      PRINT-LCS(b, X, i-1, j)
8.  else PRINT-LCS(b, X, i, j-1).
```

Initial call to print LCS is PRINT-LCS(b, X, X.length, Y.length)

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS084
FoE&T Semester: 6 Session: 2021-22
Jamia Millia University, New Delhi - 110025

X = NOWADAYS

Y = CONGRATS.

i \ j		0	1 (C)	2 (O)	3 (N)	4 (G)	5 (R)	6 (A)	7 (T)	8 (S)
0	x_i	0	0	0	0	0	0	0	0	0
1	N	0	N≠C 0	N≠O 0	N=N 1	N≠G 1	N≠R 1	N≠A 1	N≠T 1	N≠S 1
2	O	0	O≠C 0	O=O 1	O≠N 1	O≠G 1	O≠R 1	O≠A 1	O≠T 1	O≠S 1
3	W	0	W≠C 0	W≠O 1	W≠N 1	W≠G 1	W≠R 1	W≠A 1	W≠T 1	W≠S 1
4	A	0	A≠C 0	A≠O 1	A≠N 1	A≠G 1	A≠R 1	A=A 2	A≠T 2	A≠S 2
5	D	0	D≠C 0	D≠O 1	D≠N 1	D≠G 1	D≠R 1	D≠A 2	D≠T 2	D≠S 2
6	A	0	A≠C 0	A≠O 1	A≠N 1	A≠G 1	A≠R 1	A=A 2	A≠T 2	A≠S 2
7	Y	0	Y≠C 0	Y≠O 1	Y≠N 1	Y≠G 1	Y≠R 1	Y≠A 2	Y≠T 2	Y≠S 2
8	S	0	S≠C 1	S≠O 1	S≠N 1	S≠G 1	S≠R 1	S≠A 1	S≠T 1	S=S 3

c

i \ j		1 (C)	2 (O)	3 (N)	4 (G)	5 (R)	6 (A)	7 (T)	8 (S)
1	(N)	↓	↓	↘	←	←	→	→	→
2	O	↓	↘	↓	↓	←	↓	↓	↓
3	W	↓	↓	↓	↓	←	↓	↓	↓
4	(A)	↓	↓	↓	↓	↓	↖	←	→
5	D	↓	↓	↓	↓	↓	↓	↖	↓
6	A	↓	↓	↓	↓	↓	↘	↖	↓
7	Y	↓	↓	↓	↓	↓	↓	↖	↓
8	(S)	↓	↓	↓	↓	↓	↓	↓	↖

b

X : N O W A D A Y S

Y : C O N G R A T S

LCS : N A S

There can be other LCSs also if other paths are followed.

PL(b, X, 8, 8)
↓
PL(b, X, 7, 7) → X_8 = 'S'
↓
PL(b, X, 6, 7)
↓
PL(b, X, 5, 7)
↓
PL(b, X, 4, 7)
↓
PL(b, X, 4, 6)
↓ → X_4 = 'A'
PL(b, X, 3, 5)
↓
PL(b, X, 2, 5)
↓
PL(b, X, 1, 5)
PL(b, X, 1, 4)
↓ → X_1 = 'N'
PL(b, X, 0, 3) → ...

Analysis of Algorithm.

There are nested loops of m within n & each inner loop costing O(1) therefore,

$$T(n) = O(m \times n)$$

length of X, length of Y.

Since we're allocating two tables of size $m \times n$ & $(m+1) \times (n+1)$, the space complexity is.

$$S(n) = O(mn).$$

We can use space optimization to store only 3 entries to calculate current entry, but we cannot avoid building

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS 081
FoE&T Semester: 6 Session: 2021-22
Jamia Millia University, New Delhi - 110025

Q5 a) String Matching Algorithm with finite Automata

→ we preprocess the pattern to build
→ we run the text through this finite automata character by character, whenever few reaches final state, we detect the pattern P at location $i - m$ (m = pattern length)

current character pos"

string : pattern set : Alphabet of text

```
COMPUTE-TRANSITION-FUNC (P, Σ)
1. m = P.length.
2. for q = 0 to m
3.     for each character a ∈ Σ.
4.         k = min(m+1, q+2).
5.         repeat k = k-1 until Pk ⊐ Pq a
6.         δ(q, a) = k.
7. return δ.
```

$T(n) = O(m^3|\Sigma|)$
$S(n) = O(m^2)$
pattern length

```
FINITE-AUTOMATON-MATCHER
1. n = T.length.
2. q = 0
3. for i = 1 to n
4.     q = δ(q, T[i])
5.     if (q == m)
6.         print "pattern found at shift i - m.
```

text len

$T(n) = O(n)$

Transition function δ for FSM of given pattern P = "a c b"
No of states = pattern size + 1 = 3 + 1 = 4 [0...3]

$\Sigma = \{a, b, c\}$

state	a	c	b	pattern
→ 0	1	0	0	a
1	1	2	0	c
2	2	3	0	c
(3)	1	0	0	

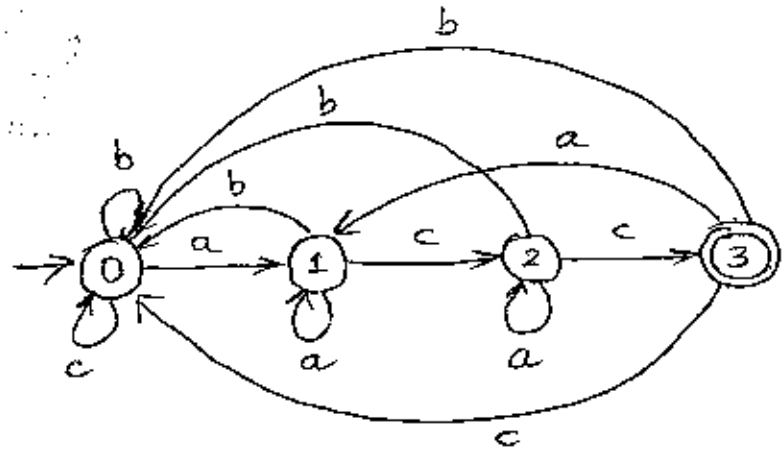
a|a
match pat len = 1
a c a
match pat len = 2

a c c a
a c b

a c ∉
∉ c a
a ∉ ∉
∉ ∉ a
match pat len = 1

a c c c
a c c b

a c ∉
∉ c c
a ∉ ∉
∉ ∉ c



index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Text	a	c	a	b	b	a	a	b	c	b	a	c	c	a	b	b	a	c	c	b	a	b	c	c	a	b	a	b
state	0	1	2	2	0	0	1	1	0	0	0	1	2	3	1	0	0	1	2	3	0	1	0	0	0	1	0	1	0

found $14-3=11$ found $20-3=17$.

∴ acc is found at locations 11 & 17 in the given text.

Q 5 c)(i) **NP - class**

Set of all decision problems solved by a non deterministic machine in polynomial time

or.

It is a complexity class that represents set of all decision problems where the answer is 'yes' have proofs that can be verified in polynomial time.

This means, if someone gives us an instance of the problem and a certificate (sometimes called witness) to the answer being yes, we can check that it is correct in polynomial time.

eg. INTEGER FACTORIZATION.

pblm: given integers n & m, is there an integer f with $1 < f < m$ such that f divides n ?

This is a decision pblm bec answers are yes or no. If someone hands us an instance of the pblm (ie n & m) and an integer f s.t $1 < f < m$, and claims that f is a factor of n (the certificate), we can check the answer in polynomial time by performing the division n/f

NP - complete

NPC is a complexity class which represents the set of all problems X in NP for which it is possible to reduce any other NP pblm Y to X in polynomial time.

This means that we can solve Y quickly if we know how to solve X quickly. Y is reducible to X, if there is a polynomial time algorithms f to transform instances y of Y to instances $x = f(y)$ of X in polynomial time, with the property that the answer to y is yes, iff the answer to f(y) is yes.

eg. 3SAT (a Boolean satisfiability problem)

Pblm: given a conjunction (ANDs) of 3-clause disjunctions (ORs), ie statements of form

$(v_{11}$ OR v_{21} OR $v_{31})$ AND
$(v_{12}$ OR v_{22} OR $v_{32})$ AND
...
AND
$(v_{1n}$ OR v_{2n} OR $v_{3n})$

where v_{ij} is a boolean variable or negation of a variable from a finite predefined list $(x_1, x_2, \ldots x_n)$

It can be shown that every NP problem can be reduced to 3SAT: Cook's theorem.

NPC pblms are important because if a deterministic polynomial time algorithm is found to solve any of them, then every NP pblm is solvable in polynomial time.

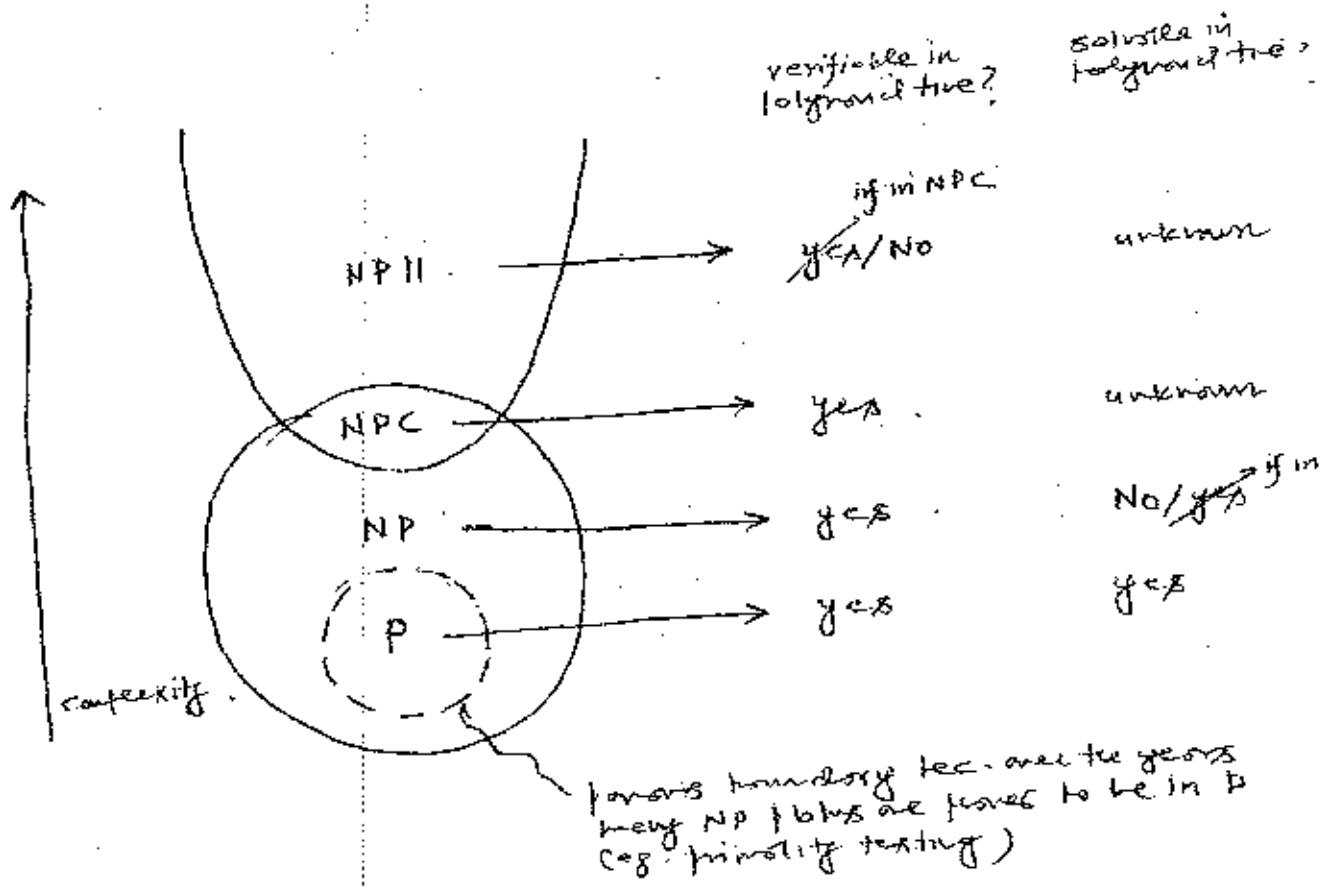
NP-Hard

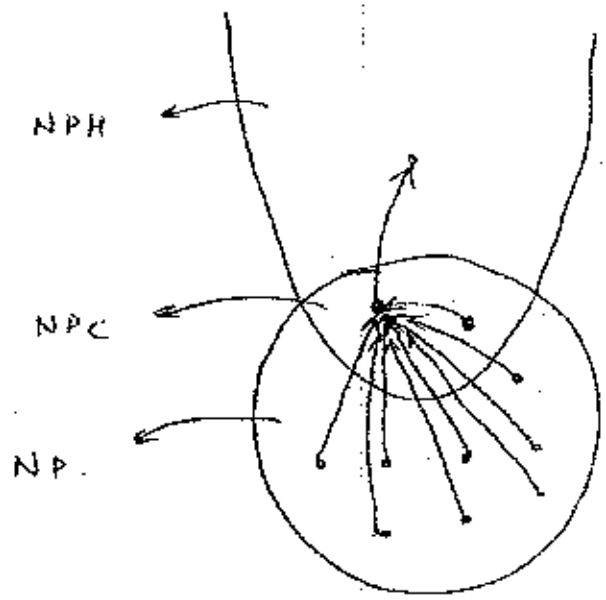
These are the problems which are at least as hard as the NP-complete problems. All NPH pblms do not have to be NP, and they do not have to be decision problems.

A problem X is NP-hard, if there is an NP-complete problem Y, such that Y is reducible to X in polynomial time.

But since any NPC pblm can be reduced to any other NPC in polynomial time, all NPC pblms can be reduced any NPH problem in polynomial time. Then, if there is a soln to one NPH pblm in polynomial time, there is a soln to all NP pblms in polynomial time.

eg. HALTING PROBLEM
Pblm: given a program P and input I, will it halt?





$i \longrightarrow j$ indicates i is reducible to j in polynomial time.

Q5c) (ii) Naive Algorithm for string matching.

```
NAIVE-STRING-MATCHER(T, P).
1.  n = T.length().
2.  m = P.length().
3.  for s = 0 to n - m
4.      if P[1..m] == T[s+1..s+m]
5.          print "pattern occurs at shift s".
```

explanation

The algorithm works by matching pattern with corresponding window of text by shifting window one character at a time.

Analysis.

$$T(n) = O((n-m+1)m).$$
$$S(n) = O(1).$$

This is an inefficient algorithm as it recomputes information which can be used to perform further operations.