

Algorithm Analysis and Efficiency

Algorithm: A sequence of computational steps which solves a well-specified computational problem.

(steps) A function solves a well-specified computational problem.

Correct Algorithm: for every input instance, it halts with a correct output.

Incorrect Algorithm: might not halt at all on some input instances, or it might halt on wrong output.

P.T.O.

Algorithmic Analysis of Big O

i) Correctness

ii) Amount of work done (Time complexity)

iii) Amount of space used (Memory)

iv) Simplicity, clarity

v) Optimality

Running time:

$$T(n) = \text{constant} + O(n^{\log_2 2})$$

constant term is constant

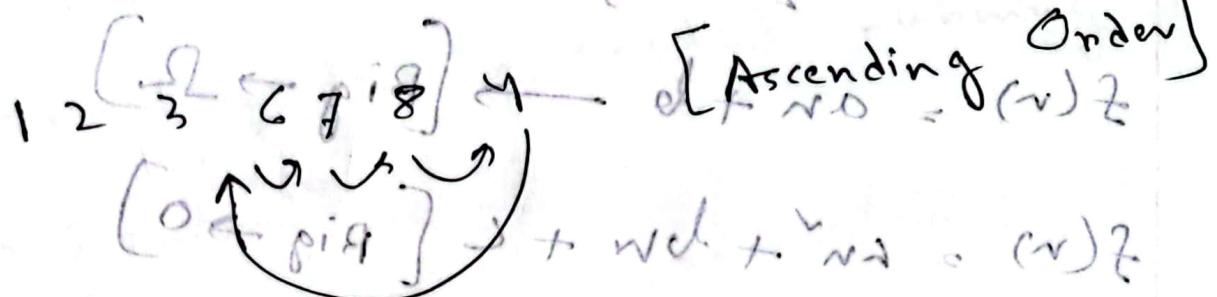
$$C = \frac{5}{3} \times (2^{32})$$

$$Z = f(x) + O(2^x)$$

P.T.O.

Insertion Sort

$i, b, t, l, r \in T$



$\Rightarrow 1, 2, 3, 4, 6, 7, 8$ after sorting sifted up to A *

Insertion sort (A, n) { with num ←

know how to operate n { from n ←

for i = 2 to n { until $n-1$ ←

key = $A[i]$

$j = i-1$;

while $(j > 0)$ do $A[j+1] = A[j]$ { $\rightarrow n-1$

$\frac{n(n+1)}{2}$

~~key~~ $= A[j]$;

$A[j+1] = \text{key}$ with num shiforg ←

$\Rightarrow j = j-1$; until $j=0$

\Rightarrow know how to operate $n-1$ ←

$A[j+1] = \text{key}$ for i = 2 to $n-1$ ←

\Rightarrow

$$T = t_1 + t_2 + t_3 \xrightarrow{\text{total wait time}}$$

$f(n) = an + b \rightarrow [Big O]_S$

$$f(n) = cn^2 + dn + e \xrightarrow{\text{total wait time}} [Big O]$$

* Asymptotic Performance

→ Run time $\xrightarrow{\text{total wait time}}$

→ Memory / storage requirement

→ Bandwidth / Power requirements /

$\left\{ \begin{array}{l} \text{logic gates etc.} \\ \text{and } [i]_A = i \\ \text{etc. } i-i = i \end{array} \right.$

$$\frac{(mn)^n}{5}$$

Worst case:

→ provide upper bound $[i]_A$ of running time.

→ An absolute guarantee of requirement resources $[i]_A$

L

Average Case:

- \rightarrow provides the expected time.
- \rightarrow Random inputs
- \rightarrow Real life inputs.

Upper bound notation

A function $f(n)$ is $O(g(n))$ if

there exists positive constants C .

and w_0 such that $f(n) \leq Cg(n)$ for all $n \geq w_0$.

$\Rightarrow g(n) \geq f(n)$ for all $n \geq w_0$

$O(\cdot) \rightarrow$ "Big-O"

$$f(n) = 5n^2 + 3n + 2 \quad \{ 10^{w_0} \}$$

$$\{ 5n^2 + 3n^2 + 2n^2 \leq 10n^2 \}$$

$$(C=10, g(n)=n^2 \quad \{ n \geq 1 \})$$

$$g(f(r) = 10^{50} r^7 + \dots)$$

~~unit~~ ~~g(f(r))~~ ~~10^{50} r^7 + \dots~~ ~~approx~~

~~approximate value of r~~ ~~approximate value of n~~

~~approximate value of n~~ ~~approximate value of n~~

$$(10^{50} \cdot 2^7 + \dots) \cdot 2^7 \approx 10^{57}$$

$$(2^w (10^{57})) \cdot 2^w \approx 10^{57+2w}$$

~~approximate value of w~~ ~~approximate value of w~~ ~~approximate value of w~~

$$g(n) \approx 2^{10} \cdot 10^{57} \approx 10^{57+10} = 10^{67}$$

$$n > 10^{67} \approx 10^{67}$$

$$(10^{57}) \rightarrow n = 10^{57}$$

$$n_0 = 10^{57} \approx 10^{57}$$

$$\{f_k\} \approx n = 10^{57}, 10^{57}$$

6.11
30/7/23

Lecture-3

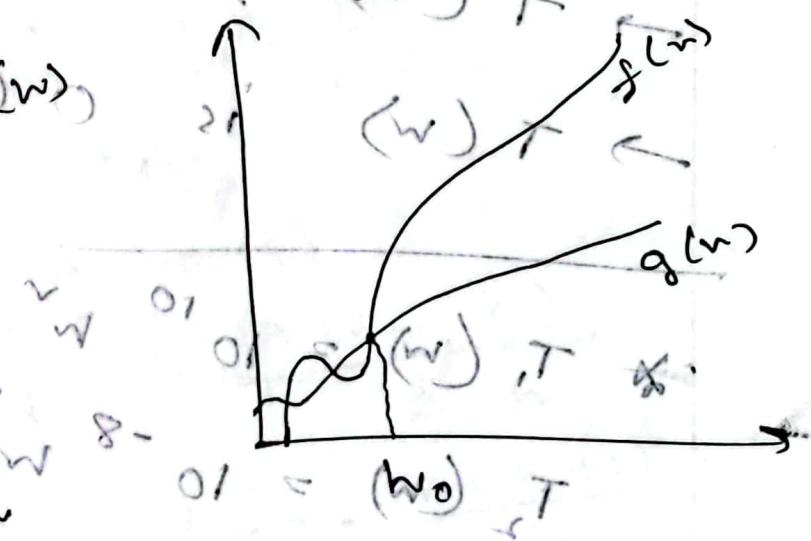
DSA-2

* Lower bound for $f(n) \in \Theta(g(n))$

if $c_1 g(n) \leq f(n) \leq c_2 g(n)$ where $c_1, c_2 > 0$
and $c_1, c_2 > 0, n \in \mathbb{N}$. Then we say $g(n)$ is asymptotic lower bound for $f(n)$

$\Omega(g(n))$

works Big Omega

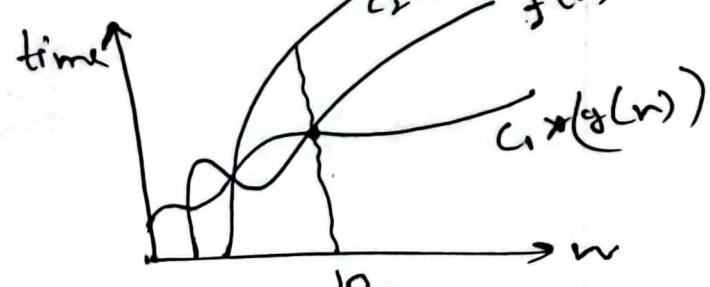


* Tight Bound (Big Theta)

$\Theta(g(n))$, if for any positive

constants c_1, c_2 & w_0

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall n > w_0$$



* $T(w) = 5w^3 + \text{constant}$, $g(w) \propto w^3$

$\rightarrow T(w)$ is $O(w^3)$

$\rightarrow T_1(w) = 10^{10} w^3$

$T_2(w) = 10^{-8} w^3$ ~~for where~~

$T_1 = T_2$

if $w \leq 10^{18}$, $T_1(w) > T_2(w)$

if $w \rightarrow \infty$, $T_2(w) \gg T_1(w)$

& $w \rightarrow \infty$, $T_1(w) \propto (w)^3$

$w \rightarrow \infty$, $T_2(w) \propto (w)^3$

$w \rightarrow \infty$, $T_1(w) \propto (w)^3$

$w \rightarrow \infty$, $T_2(w) \propto (w)^3$

$w \rightarrow \infty$, $T_1(w) \propto (w)^3$

$w \rightarrow \infty$, $T_2(w) \propto (w)^3$

* Expected cost Analysis: rot cost

1. for i , $\text{if } \text{to}(w[i]) \rightarrow w+1$
 $t_i \leftarrow \epsilon$

2. if $\text{arr}[i] \cdot \epsilon \cdot 3 \rightarrow \epsilon$; $\rightarrow w$
 $\leftarrow \epsilon$

3. print ($\text{array}_v[i]$);
 $\text{rot} \leftarrow n$

$\leftarrow [n] \rightarrow \text{Best}$
 $\rightarrow w \rightarrow \text{worst}$
bad $\rightarrow \text{final} \rightarrow \frac{(1+2+\dots+w)}{w}$
~~bad~~ $\rightarrow \text{Avg} \rightarrow \frac{w}{2}$
 $\rightarrow \frac{w}{2}$ $\frac{w(w+1)}{2}$

$O(w)$
 $\Omega(w)$
 $\Theta(w)$
full run time

$$(1+n_{\text{pool}}) \frac{w}{2}$$

$$w_{\text{pool}} \frac{w}{2}$$

$$\underline{\frac{w}{2}}$$

$$(w_{\text{pool}}) \underline{0}$$

$$[f] \underline{a}$$

* for ($i \leq w$, $i > 0$, $i \leftarrow i-5$) {
 } }

- 1 → if ($A[i] < 0$) $i \leftarrow i + 1$
- 2 → ~~if ($A[i] < 0$) $i \leftarrow i + 1$~~
 we → ~~i~~ Breaks $\downarrow [i]$ from $\overline{z_i}$
- 3 → ~~i~~ (i) \leftarrow ~~for ($k=1$ to w) $k \leftarrow k + 2$~~
- 4 → ~~for ($k=1$ to w) $k \leftarrow k + 2$~~ trying
- 5 → ~~print ($A[k]$)~~

line ($i \leq w$)	Worst	Best
1	$\frac{n}{5} + 1$	1
2	$\frac{n}{5}$	1
3	0 worst	$\frac{1}{w}$ (w) 0
4	$\frac{n}{5} (\log_2 n + 1)$	0 (w) 52
5	$\frac{n}{5} \log_2 n$	0 (w) 0
$O(n \log_2 n)$	0.5 n	
$\Omega(1)$		

* if ($k = 1$; $k \leq n$; $k \neq 2$) $\rightarrow \log_2 n$

+ 5⁶⁰¹

* $\rightarrow 2$ for ($i = n$; $i > 0$; $i = i - 3$)

$\rightarrow 2$ if ($A[i] < i \cdot 60$)

$\rightarrow 3$ ~~order vs~~

$\rightarrow 4$ for ($k = n$; $k \geq 1$; $k = \frac{n}{3}$) $\rightarrow 4$

$\rightarrow 5$ print $A[k]$ (f) ~~return~~

	worst	best
1	$\frac{n}{3} + 1$	1
2	$\frac{n}{3}$	1
3	0	1
4	$\frac{n}{3} (\log_u n + 1)$	0
5	$\frac{n}{3} \log_u n$	0

$$\begin{aligned}
 & \text{let } \log_4 w = \frac{\log_2 w}{\log_2 4} \quad (i=4) \\
 & \text{so } i = i \left(\log_2 w_i \right) \text{ note } \\
 & \quad (\text{as } i \in [1] \text{ and } \log_2 w_i) \\
 & \quad = \frac{1}{2} N \log_2 w \\
 & \text{# } \log_2 (w \log_2 w) \text{ is } n \times n \text{ of } n \\
 & \# S_2(2) \quad [n]_A \text{ twigs}
 \end{aligned}$$

fact	error	
0	$\epsilon + \frac{N}{\epsilon}$	0
ϵ	$\frac{N}{\epsilon}$	ϵ
0	0	ϵ
0	$(1 + n_{\text{pol}}) \frac{N}{\epsilon}$	n
?	$n_{\text{pol}} \frac{N}{\epsilon}$?

~~6W
3.10.23~~

Exact Cost Analysis

```
for (i=0; i<n; i++) {  
    for (j=2; j<=i; j++) {  
        print(j);  
    }  
}
```

$$\begin{aligned} & \text{Time complexity: } O(n^2) \\ & \text{Cost analysis: } \\ & \quad 1 \rightarrow n+2 \\ & \quad 2 \rightarrow \frac{n(n+1)(2n+1)}{6} - 1 \end{aligned}$$

Big O (n^3)

P.20

```

*  $c = 0$ ; for (j=0; j < w; i++) {
    for (i =  $\frac{w}{2}$ ; i < w; i++) {
        for (j = 2; j < w; j = j + 2) {
            c += k +  $\frac{w}{2}$ 
        }
    }
}

for (i = 0; i < m; i++) {
    for (j = 2; j < w; j = j + 2) {
        c += k +  $\frac{w}{2}$ 
    }
}

```

$$1 \rightarrow \frac{n}{2} + 2 \rightarrow \frac{n}{2} + C$$

$$2 \rightarrow \log_2 w + 2 \rightarrow \log_2 w + C$$

$$3 \rightarrow \log_2 w \rightarrow \log_2 w + C$$

$$4 \rightarrow m + 2 \rightarrow m + C$$

$$5 \rightarrow \frac{n}{2} + 2 + 1 \rightarrow \frac{n}{2} + C$$

$$6 \rightarrow \frac{n}{2} \rightarrow \frac{n}{2}$$

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

$$O(\log w) \quad \text{if } \log w > m$$

$$O(m) \quad \text{if } m > \log w$$

$$\frac{1}{\epsilon^{\frac{1}{n}}} + \frac{m}{w} + C$$

P.T.O.

59

- * Devise the best & worst case running times equations & express in big-O notation.
- * Prove that running time ~~$\Theta(n^3)$~~

$$T(w) = w^3 + 20w + 1 \text{ is } O(w^3)$$

→ By the Big-O definition, $T(w) \leq C \cdot w^3$ for some $w > w_0$.

Let's check this condition.

$$\text{if } w^3 + 20w + 1 \leq C \cdot w^3$$

$$\Rightarrow 1 + \frac{20}{w^2} + \frac{1}{w^3} \leq C$$

∴

Therefore, the big-Oh notation holds for $n > n_0 = 1$ & $c \geq 1.22$

$$\begin{aligned} &= (1+20n) \\ &= 22 \end{aligned}$$

larger values of n_0 result in smaller factors (ex: if we let $n_0 = 10$, then $c \geq 1.20n$ & so on). But in any case the above statement is valid.

Prove that $T(n) = n^3 + 20n$ is

$$T(n) \leq cn^3$$
$$n^3 + 20n \leq cn^3$$
$$n + \frac{20}{n} \leq c$$

P.T.O

$f(n) \leq cn^3$
 $f(n) \leq cn$

$$r = \sqrt{20} = 4.47 \text{ when } C=18.94$$

$$\frac{d}{dw} \left(w + \frac{20}{w} \right) = 0$$

$$\text{when } w \text{ is greater than } \frac{20}{w} = 0$$

then it has to be less than $\frac{20}{w}$ to have a root

$$(w - \frac{20}{w}) > 0 \rightarrow w^2 > 20 \rightarrow w > \sqrt{20} \rightarrow 4.47$$

$$\Rightarrow w^2 > 20 \Rightarrow w > \sqrt{20} \Rightarrow w > \cancel{\sqrt{20}}$$

$$w > 4.47 \text{ and } (l \leq 2 \text{ times of } \sqrt{20})$$

$$(i=0; i \leq w; i++) (n+2) \rightarrow n+9$$

$$(i=1; i \leq w; i++) (n+1)$$

$$(i=2; i \leq w; i++) (n)$$

$$(n+9) \rightarrow n+9 + w$$

6.3
10.23

DSA-2

Divide & conquer

* Divide & conquer is a general algorithm design paradigm.

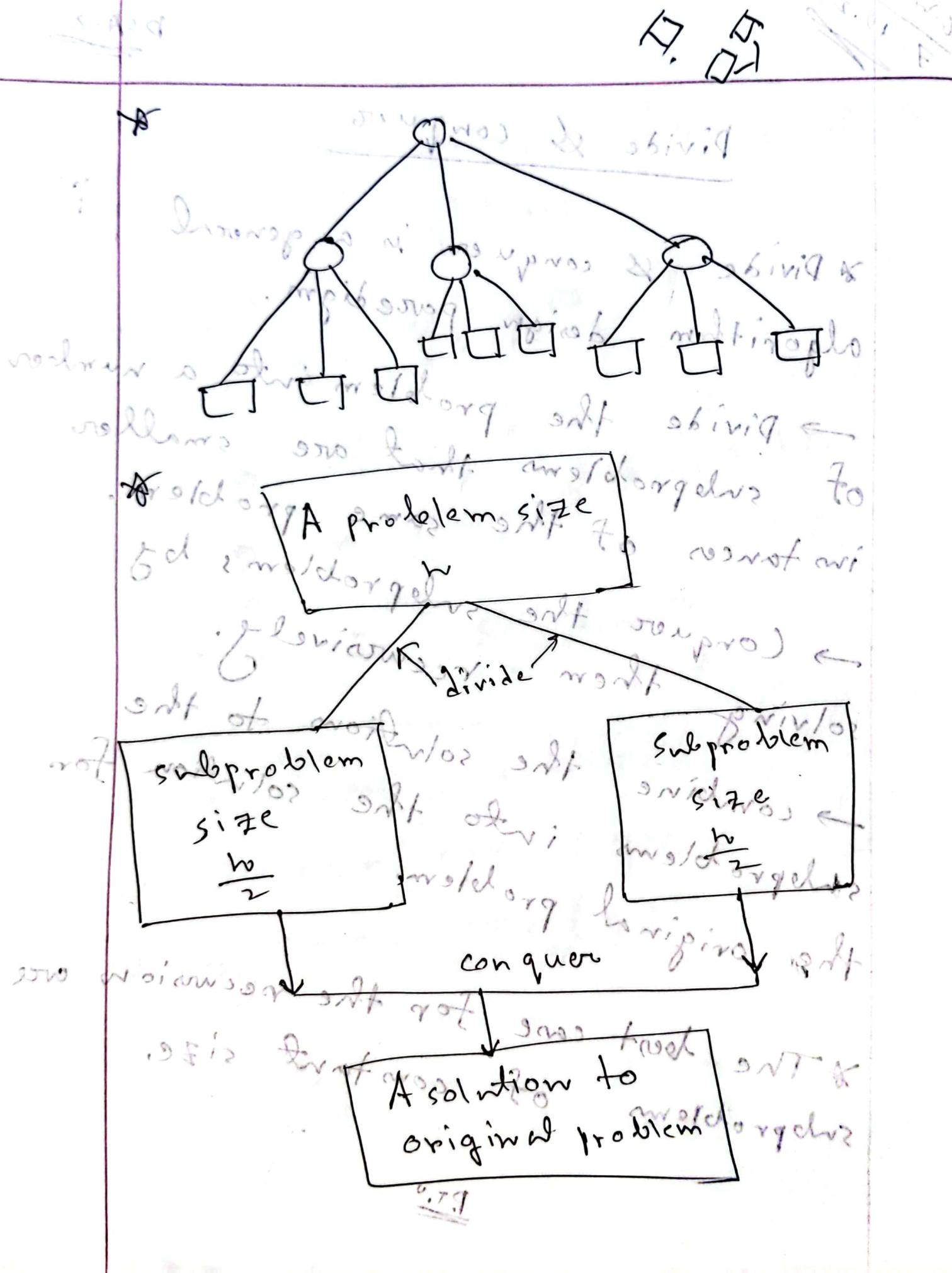
→ Divide the problem into a number of subproblems that are smaller instances of the same problem.

→ Conquer the subproblems by solving them recursively.

→ combine the solutions to the subproblems into the solution for the original problem.

* The best case for the recursion tree of width w of constant size.

P.T.O



Finding Maximum & Minimum

Input: An array $\{1 \dots n\}$ to n numbers.

Output: The max & min value.

A =	13	-12	-25	20	5	6	7	8	10	11
	Max									Min

List \rightarrow List 1 (min) with $\frac{n}{2}$ elements
List \rightarrow List 2 (max) with $\frac{n}{2}$ elements

min \rightarrow Min (min1, min2)

max \rightarrow Max (max1, max2)

Algorithm Minmax (A , start, end)

{ if ($start == end$) return ($A[start]$, $A[end]$)

else if ($start == end - 1$) {

 if ($A[start] < A[end]$) return ($A[start]$, $A[end]$);

 else return ($A[end]$, $A[start]$);

}

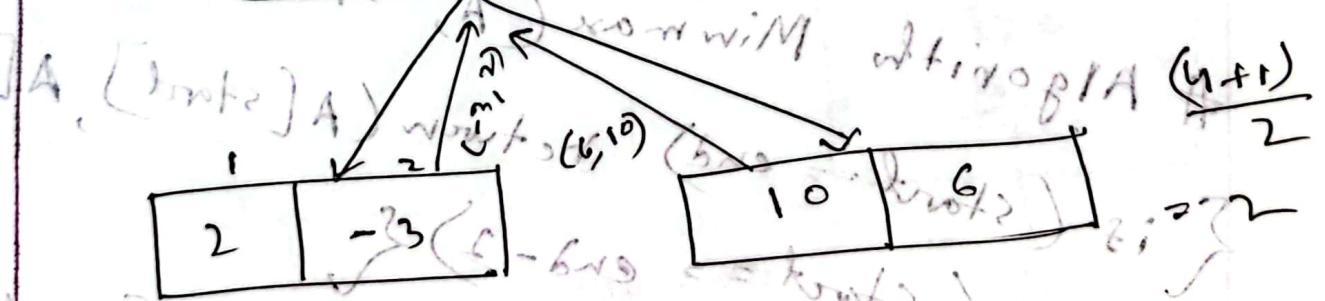
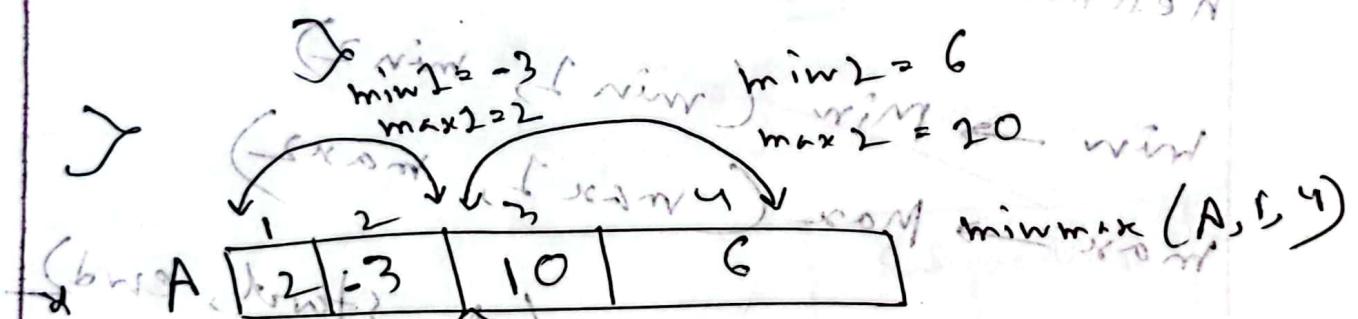
else { // if no movement possible
// calculate mid

$$\text{mid} = \lfloor (\text{start} + \text{end}) / 2 \rfloor; \text{left} = \text{right}$$

if (min1, max1) = Minmax(A, start, mid);
(min2, max2) = Minmax(A, mid+1, end);

(min2, max2) = Minmax(A, mid+1, end);

return (min(min1, min2), Max
of max1, max2));



if (left > right) return (A[left], A[right]);

if (left == right) return (A[left], A[right]);

if (left < right) {

L R

0	8	5	2	3	12	5	1	6
-1	3	2	-20	25	31			

$(n-1)A \rightarrow (n-1)A (n-1)A (n-1)A$

~~C.W
10. 10. 23. 37. A~~

Maximum Sum Subarray

divide &

1	2	3	4
2	-4	3	2

$\{1, -4, 3, 2\} \rightarrow 3 - 4 \text{ tr}$

1	-4
2	3

-3

-3 tr

3	2
4	5

-1

4+3+2+1

= 10

$\frac{n(n+1)}{2}$

$= O(n^2)$

1	-4	3
2	3	2

0

-4	3	2
3	2	1

1

1	-4	3	2	1
2	3	2	1	2

-1 tr

1	2	3	4	5	6	7	8	9	10
13	-3	-25	20	-3	-16	-23	18	20	-7

* $A[1-1], A[1-2], A[1-3], \dots, A[1-n]$

* $n-1$ $A[2-2], A[2-3], \dots, A[2-n]$

* $n-2$ $A[3-3], A[3-4], A[3-5], \dots, A[3-n]$

1	2	3	...	$n-1$	n
---	---	---	-----	-------	-----

Top { $\rightarrow s$ } \rightarrow s s \rightarrow $A[n-n]$

1

$\rightarrow s$

$\rightarrow s$

\rightarrow

$\max = -\infty$
for ($i=1$ to n) {

 sum = 0;

 for ($j=i$ to n) {

 sum = sum + $A[j]$;

 if ($sum > \max$) {

$\max = sum$;

(Brute force
Algorithm)

s	s
-----	-----

3

1	2	3	s	s	s
---	---	---	-----	-----	-----

3

* Divide & conquer suggests that we divides the array into two sub arrays as equal as possible. That is, we find the mid point (mid), and consider the subarrays $A[\text{low} \dots \text{mid}]$ and $A[\text{mid}+1 \dots \text{high}]$. Therefore, any contiguous subarray $A[i \dots j]$ of $A[\text{low} \dots \text{high}]$ must lie in exactly one of the following patterns:

① entirely in the left subarray

$A[\text{low} \dots \text{mid}]$

$$\text{so } \text{low} \leq i \leq j \leq \text{mid}$$

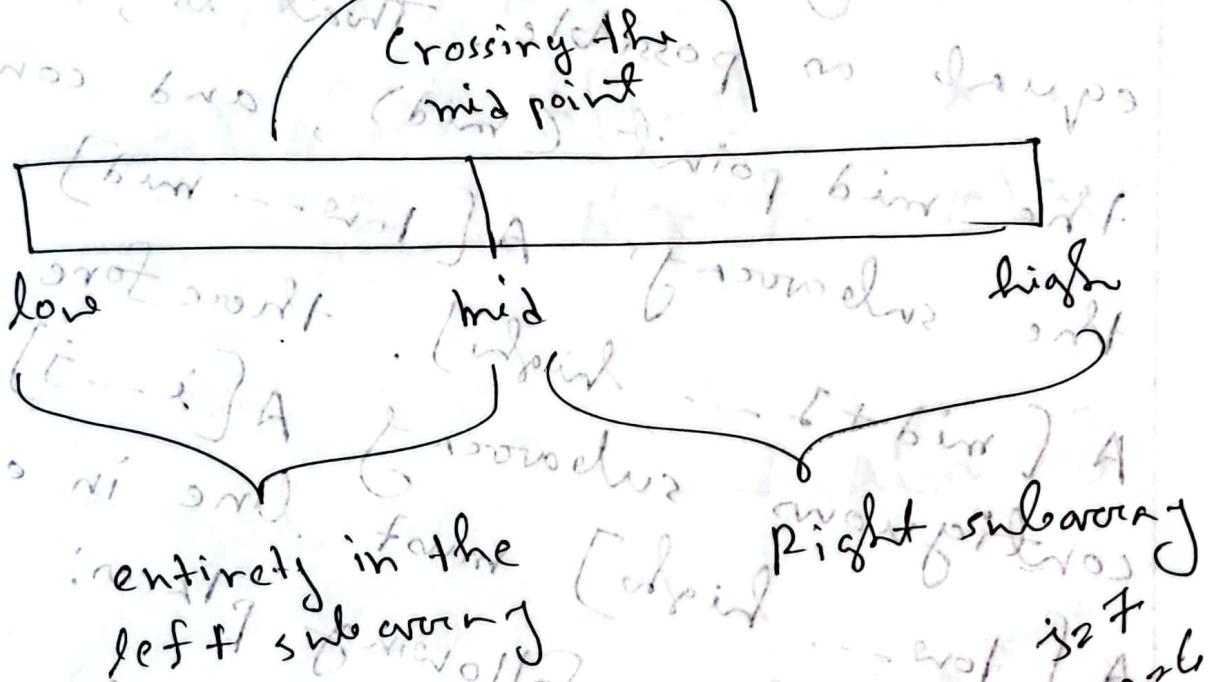
② entirely in the right subarray

$[\text{mid}+1 \dots \text{high}]$

$$\text{so that } \text{mid} \leq i \leq j \leq \text{high}$$

④ Crossing the mid point

so $low \leq i \leq mid \leq high$



2	3	2	3	6	4	1	5	8	7	8
-3	10	-5	2	3	-1	6	4	7	9	8

low bin \rightarrow mid \rightarrow high

now due to $i < mid$

\leftarrow 2 \rightarrow 3 4 5 6 7 8

\leftarrow -3 \rightarrow 2

high \leftarrow 7 (max) \rightarrow 6 (max)

(2, 7, 7+6)

* Find max crossing subarray (left, mid, right)

$$\text{left_sum} = -\infty;$$

$$\text{sum} \rightarrow 0,$$

```
for( i = mid; down to low ) {
```

sum = sum + A[i] - sum_if

if ($A^{sum} > left_sum$) {
 $sum = A[i]$
 $i++$
 $right_sum = right_sum + A[i]$
}

Left sum = semi) wnter

$\text{cost} = \text{left} + \text{right}$

$\max_{\{w_i\}} \text{efficiency}$

Feb 1st 1961 at 10:00 A.M.

~~Devon~~ (A) ~~1900~~ ~~1900~~

Sign $\sum > -\infty \rightarrow \text{no m - Swift}$

$m = 0$; $\text{there is no } \zeta$

for (i = mid + 1 to high) {
 if (arr[i] < arr[low]) {
 temp++;
 }
}

sum = sum + A[i][j];

if (sum > right - sum)

right_sum = sum;

2 max-right = 7

~~def~~ return (max-left, max-right,
left-sum + right-sum);

$\Rightarrow O(n)$

~~Find-max-subarray(A, low, high)~~
if ($low \geq high$)
return (low, high, A[low]);
 $mid = (low + high) / 2$

(left-low, left-high, left-sum)

\Rightarrow find-max-subarray(A, low, mid);

(right-low, right-high, right-sum)

\Rightarrow find-max-subarray(A, mid+1, high);

$max = max(left, right)$
 $C = left.sum + right.sum$

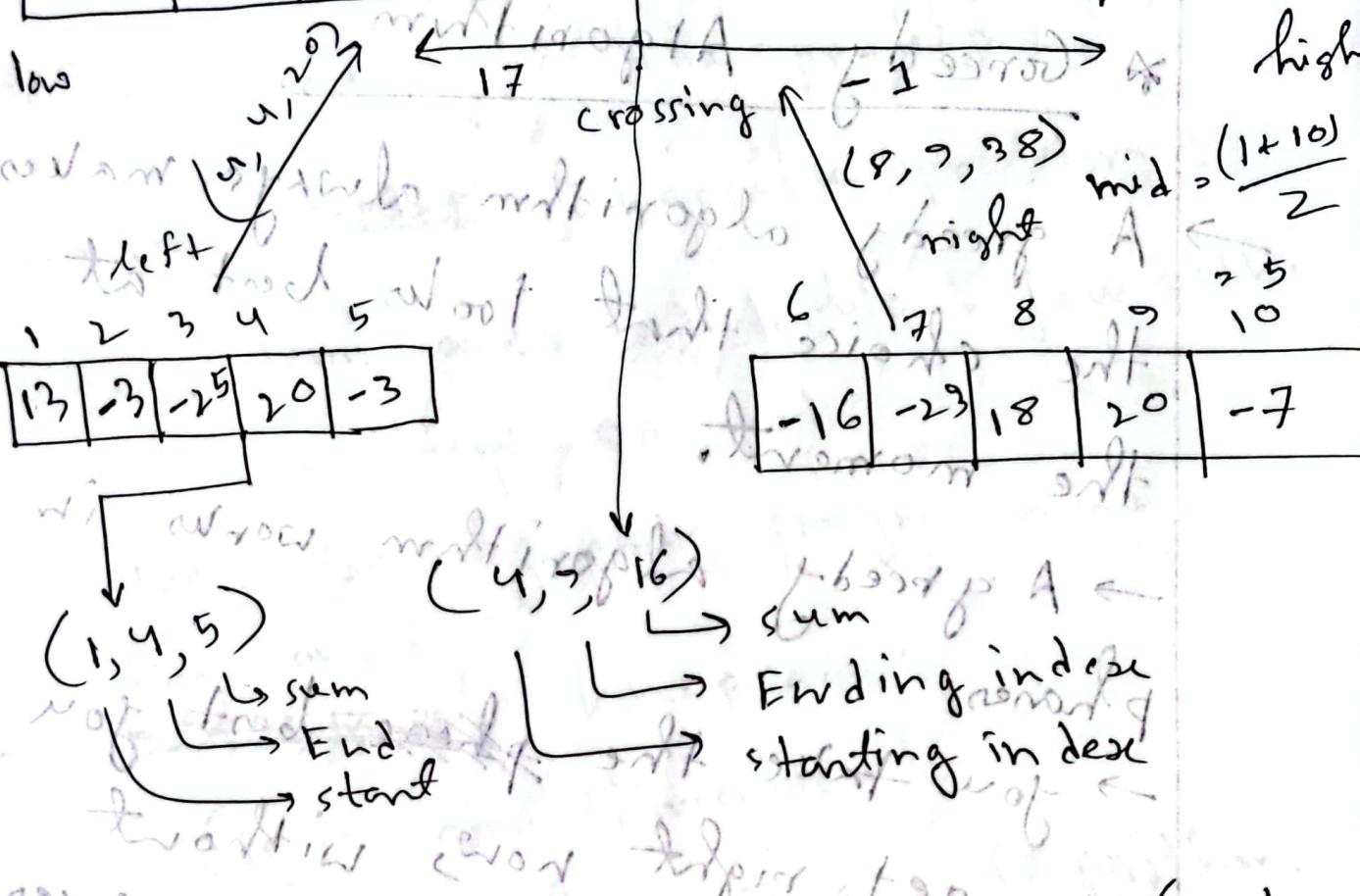
6W
14.10.23

Divide & Conquer

DSA-2

A

1	2	3	4	5	6	7	8	9	10
13	-3	-25	20	-3	-16	-23	18	20	-7



* Recurrence tree

* Greedy Algorithm

→ A greedy algorithm always makes the choice that looks best at the moment.

→ A greedy algorithm works in phases of ~~time~~ steps.

→ you take the ~~best~~ best you can get right now, without

regard for future consequences.

→ you hope that by choosing

a local optimum at each step,

you will end up at a global optimum.

→ for some problems, it works.

The knapsack problem

* Two versions of the problem.

0-1 knapsack problem:

→ The items are indivisible. You either take an item or not.

solved by dynamic programming.

Fractional

knapsack:

→ Items are divisible

solved with a greedy algorithm.

Thief must choose among n items, where the i -th item worth V_i dollars and weights w_i pounds.

→ Thief can carry W pounds.

maximize value

subject to

constraints

* Fractional Knapsack

→ knapsack capacity: W

→ there are n items, the i th item

has value v_i and weight w_i

→ goal: find $x_{i,0}$ such that

for all $0 \leq i \leq n$, $\sum x_{i,0} w_i \leq W$

$i = 1, 2, 3, 4, \dots, n$ maximum

$\sum v_{i,0} \geq \sum x_{i,0} v_i$

(cost constraint)

($n \log n$) (cost constraint)

($n \log n$) (item constraint)

constraint from cost constraint

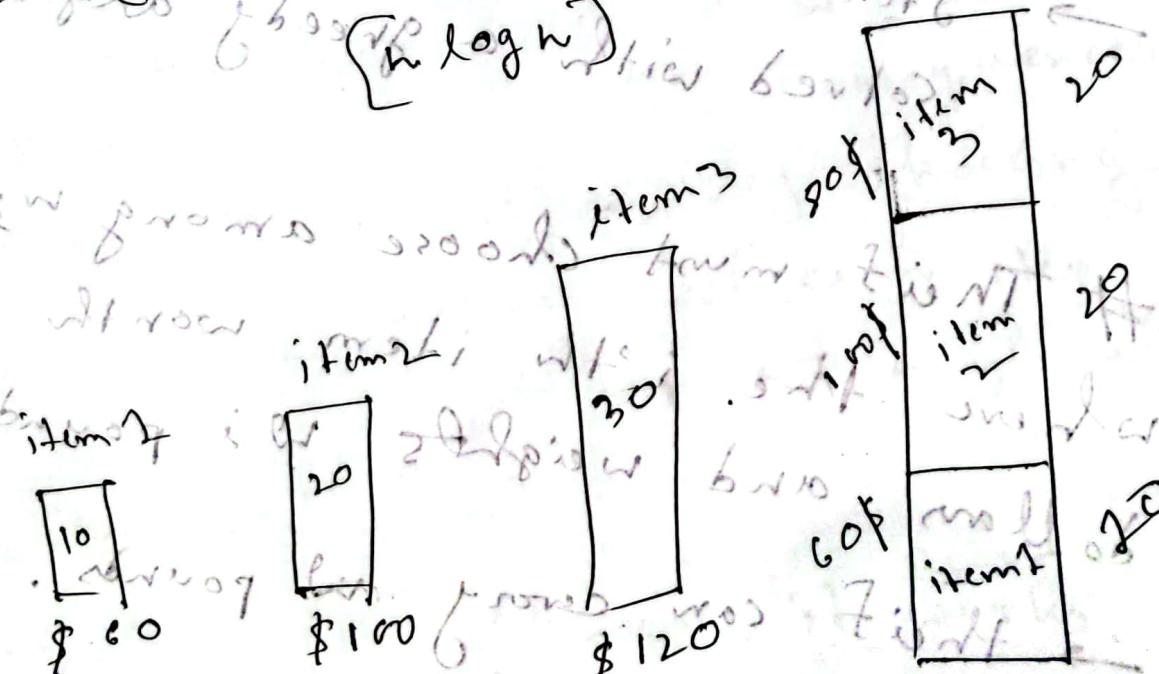
item constraint

item constraint

item constraint

item constraint

item constraint



\$6 per pound \$5 per pound \$4 per pound capacity 50

~~GW~~
16.10.23

28-10-23

Divide & Conquer
greedy

Fractional Knapsack

Anti-greedy strategy & Greedy strategy

Greedy strategy with the maximum

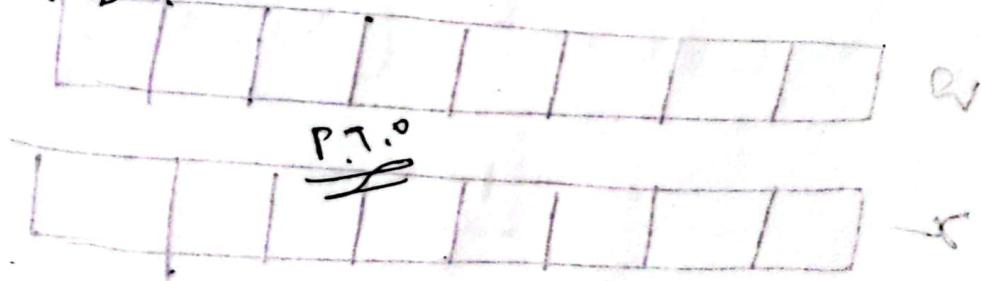
* Pick the item with the maximum value per pound $\frac{v_i}{w_i}$

* If the supply of the element is exhausted and the thief can

carry more, take as much as possible from the item with the greatest value per pound.

* It is good to order item based on their value per pound.

$$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \frac{v_3}{w_3} \geq \dots \geq \frac{v_n}{w_n}$$



v = value
 w = weight
 x =

* Fractional - Knapsack ($w, v[n], p[n]$)

{ which $w > 0$ & there are items
with v_i & w_i & we have to pick item with $\frac{v_i}{w_i}$

$$x \leftarrow \min(1, \frac{w}{w_i}) \cdot [0 \leq i \leq n]$$

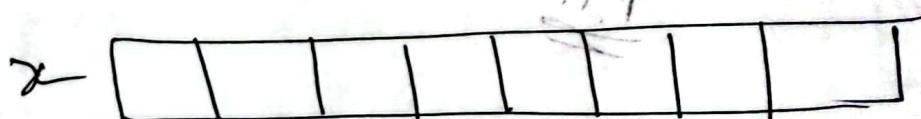
remove item i from list &
remove w_i & v_i from w & v

$w \leftarrow w - x_i w_i$; $v \leftarrow v - x_i v_i$
do this step until $w = 0$

$w \rightarrow$ the maximum of space remaining
in the knapsack

no branch initially, $w = w$

* running time:



Mathematical simulation

$$w = 50 \quad \left[\begin{array}{l} \text{bag capacity} \\ \text{item capacity} \end{array} \right] \quad (\text{gold})$$

$$w_i = 20 \quad \left[\begin{array}{l} \text{bag capacity} \\ \text{item capacity} \end{array} \right] \quad (\text{gold})$$

$$\frac{w}{w_i} = \frac{50}{20} = 2.5$$

$$x_i = \min(1, 2.5)$$

$$w = 50 - 2 \times 20 = 10$$

$$x_i = \min(1, 5)$$

$$x_i = \min(1, 2)$$

fibonacci search

$$w_i = 60 \quad \left[\begin{array}{l} \text{bag capacity} \\ \text{item capacity} \end{array} \right] \quad (\text{silver})$$

$$\frac{w}{w_i} = \frac{60}{40} = 1.5$$

$$x_i = \min(1, 1.5)$$

$$= \frac{2}{3}$$

$$w = 40 - \frac{2}{3} \times 20 = 10$$

Running time: $O(n)$ if w_i ordered

$O(n \log n)$ otherwise

0.79

item-1	item-2	item-3	
80 kg	60 kg	40 kg	100 kg
250 TK	90 TK	80 TK	Capacity

* An activity Selection problem.

Input : A set of activities $S = \{a_1, a_2, a_3, \dots, a_n\}$

→ Each activity a_i has a start time s_i and a finish time f_i

where $0 \leq s_i \leq f_i < \infty$

→ if selected activity a_i takes a time interval $[s_i, f_i]$.

P.T.O

* two activities are compatible if & only if their intervals do not overlap.

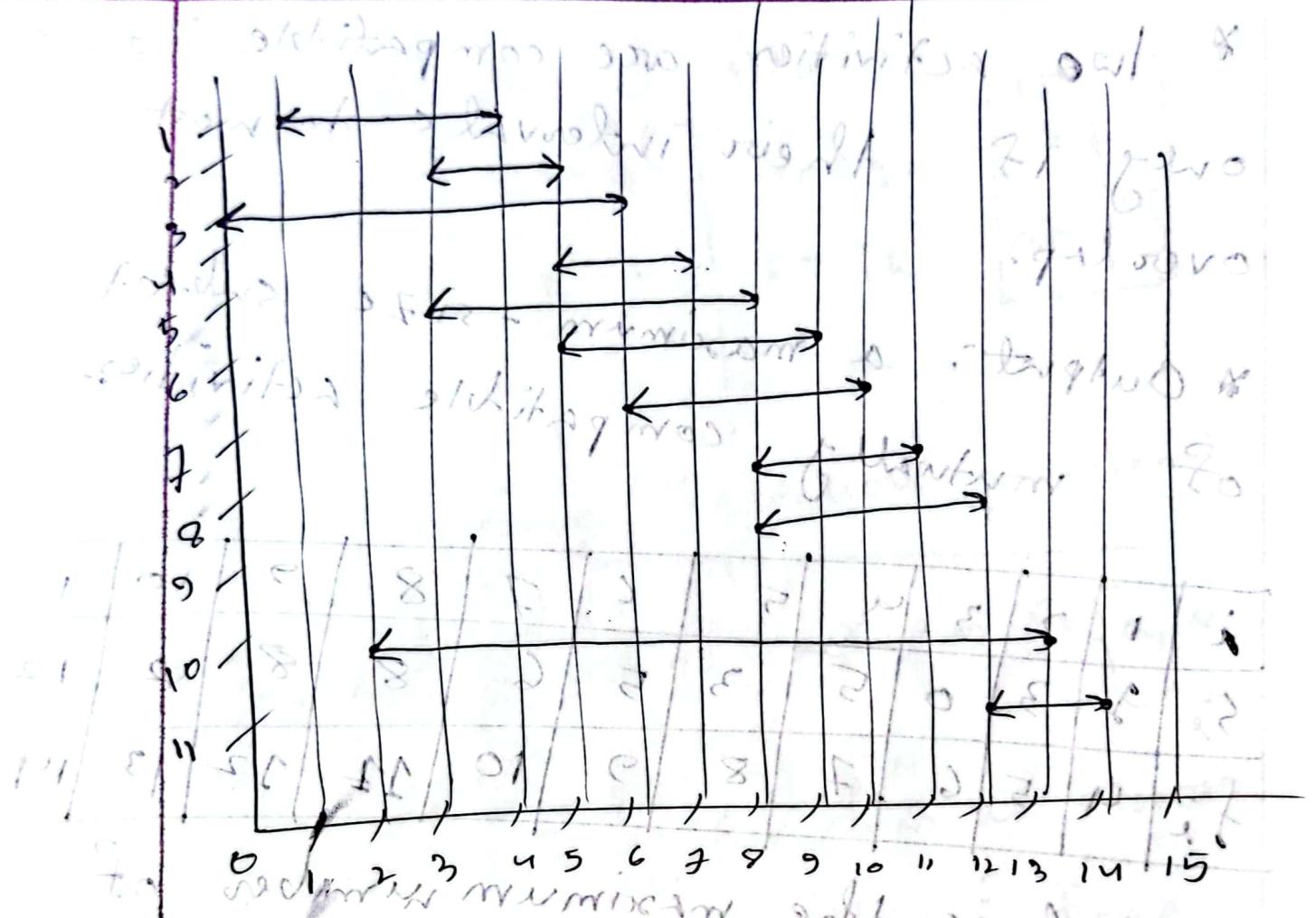
* Output: a maximum-size subset of mutually compatible activities.

i	1	2	3	4	5	6	7	8	9	10	11
s_i	2	3	0	5	3	5	6	8	8	12	12
f_i	4	5	6	7	8	9	10	12	12	13	14

What is the maximum number of activities that can be completed?

$a_1 \rightarrow a_4 \rightarrow a_8 \rightarrow a_{11}$ (max)

P.T.O



Activities and their initial times

(now) $a_1 \leftarrow a_2 \leftarrow a_3 \leftarrow a_4 \leftarrow a_5 \leftarrow a_6 \leftarrow a_7 \leftarrow a_8 \leftarrow a_9 \leftarrow a_{10} \leftarrow a_{11} \leftarrow a_{12} \leftarrow a_{13} \leftarrow a_{14} \leftarrow a_{15}$

Early finish greedy

→ select the activity with earliest finish.

→ Eliminate the activities that could not be rescheduled.

→ Repeat

Sudo code (Do it yourself)
and go to next iteration with
misleading descriptions of

Progressive simpf

is other matching & rework
and orgnization - preparation to write
report of activities behind the

err

~~219~~
~~17.10.23~~

Dynamic Programming

DSA 2

Greedy

Follows divide & conquers

Builds up a global solution

incrementally by optimising some local criterion.

Divide & Conquer

Break up a problem into disjoint subproblems, solve them recursively, and then combine their solutions to form of solution to the original problems.

Dynamic programming:

Break up a problem into a series of overlapping subproblems and build up solution to larger

P.T.O

and larger subproblems. Typically some subproblems are generated repeatedly when a recursive algorithm is running. Note that an algorithm that starts off using a

* $\text{Fib}(n)$ {
 if lot of numbers
 is ($n=0$ or 1) {
 return n ; }
 }
}

else {
 return $\text{Fib}(n-1) + \text{Fib}(n-2)$;
}
}

P.7.0

171

- ~~Q~~ ~~1~~ ~~0-1 knapsack problem~~, ~~but~~
- knapsack capacity $\leq w$
- n items to choose from.
- pack the knapsack to achieve maximum total value.

Goal:

Find x_i such that for all

$$x_i \in \{0, 1\}.$$

$$\& i = 1, 2, 3, \dots, n. \quad \{ \text{only} \}$$

$$\sum x_i \leq w$$

$$\sum x_i v_i \text{ is maximum.}$$

P.T.O

6.15 added on 20/11/2021

$p(i, w)$ → The maximum profit that can be obtained from items 1 to i if the knapsack has size w .

(knapsack of size w can have profit i item taken or not. If item i is taken then profit will increase by v_i . If item i is not taken then profit will remain same.)

Case 1:

i th item taken

$$p(i, w) = v_i + p(i-1, w - w_i)$$

\cancel{i} th item not taken

$$p(i, w) = \cancel{v_i + p(i-1, w)}$$

P.T.O

$$P(1,2) = \max(P(0,2), P(1,0) + v_2)$$

item	Weight	Value
1	2	12
2	1	10
3	3	20
4	2	15

$$W = 5$$

item	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	12	12	12	12	12	12
2	0	10	12	22	22	22	22	22
3	0	10	12	22	30	32	32	32
4	0	10	15	25	30	37	37	37

$$(w_{i-1})_q = (w_i)_q$$

CS

~~6.2
21.10.22~~

Dynamic Programming weight differs acc as 1 or 2

0-1 knapsack

if total items n , capacity $W = 5$

item	weight	value
1	2	12
2	1	10
3	3	20
4	2	15
5	5	90

$$P(i, w) = \max \{ P(i-1, w), P(i-1, w - w_i) + V_i \}$$

initially $P(0, w)$

Weight \rightarrow

	0	1	2	3	4
0	0	0	0	0	0
1	0	12	12	12	12
2	0	10	12	22	22
3	0	20	12	22	30
4	0	20	15	25	37

item {4, 2, 1}

35 for
first

* it takes ~~time~~ value for fact.

* given weight info, 2

item 1 value

item 2 value

item 3 etc.

* Runtime: $O(n^w)$

* Sudocode: do it yourself

Coin change Problem

* given unlimited amount of coins of values c_1, c_2, \dots, c_d , give change for amount M with the least number of

coins

P3.0

notes lesson 3 notes

Short algorithm

Ex: If we have coins $\{1, 2, 5\}$ and $M = 18$.
 C = {25, 10, 5, 1} with value 6.

Solution:

$$25 * 1 + 10 * 2 + 5 * 3 \rightarrow 6$$

$$= 1 * 2 + 2 * 2 + 4 \rightarrow 6$$

~~(abs) direct solution~~

Given the ~~work~~ ~~problem~~ ~~problem~~
 denominations: 1, 3 & 5, what
 is the minimum number of
 coins needed to make change
 for a given value?

$$M = 10$$

(T^v)	values	2	3	4	5	6	7	8	9	10
minimum number of coins	1	2	1	2	1	2	3	2	3	2
	$1+1$	$1+1$	$1+1$	$1+1$	$1+1$	$2+1$	$2+1$	$3+1$	$3+1$	$3+1$
	$(1+1)+2$	$(1+1)+2$	$(1+1)+2$	$(1+1)+2$	$(1+1)+2$	$2+2$	$2+2$	$1+2$	$2+1$	$3+1$
	$(1+1)+3$	$(1+1)+3$	$(1+1)+3$	$(1+1)+3$	$(1+1)+3$	$2+3$	$2+3$	$1+3$	$2+1$	$3+1$

(*) How many coins per previous coin + 1 always needed

Algorithm note

Assuming activities are sorted by finish time:

```
→ Greedy-Activity-Selector(s, f) {  
    n ← length[s];  
    A ← {a1};  
    for m ← 2 to n do  
        do if sm ≥ fi;  
        then A ← A ∪ {am};  
    return A;
```

* Merge-Sort (A, p, r)

if p < r

then q ← ⌊(p+r)/2⌋

MergeSort (A, p, q)

MergeSort (A, q+1, r)

Merge (A, p, q, r)

Merge sort running time

$$\begin{aligned}
 T(n) &= 2T\left(\frac{n}{2}\right) + bn \\
 &= 2\left(2T\left(\frac{n}{4}\right) + bn\right) + bn \\
 &= 2^2 T\left(\frac{n}{4}\right) + 2bn + bn \\
 &= 2^3 T\left(\frac{n}{8}\right) + 3bn + bn \\
 &= 2^4 T\left(\frac{n}{16}\right) + 4bn + bn \\
 &= 2^5 T\left(\frac{n}{32}\right) + 5bn + bn \\
 &= 2^i T\left(\frac{n}{2^i}\right) + ibn
 \end{aligned}$$

$$T(n) = 2^i T\left(\frac{n}{2^i}\right)$$

$$\begin{aligned}
 2^i &= n \\
 \Rightarrow i &= \log n
 \end{aligned}$$

$$\therefore T(n) = bn + b \log n$$

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

$$O(n \log n)$$

CW

0.25

30.

Binary tree approach

DSA 11

Asymptotic Analysis for T(n)

we have $T(n) = T(\frac{n}{2}) + \Theta(1)$ Recurrence

we have $T(n) = T(\frac{n}{2}) + \Theta(\log n)$

Three methods $(N) \rightarrow$

→ Substitution method $(N) \rightarrow$

→ Recursion Tree method $(N) \rightarrow$

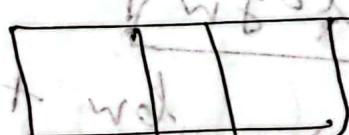
→ Master method $(N) \rightarrow$

we have $T(n) = T(\frac{n}{2}) + \Theta(1)$



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

$$n = i \cdot \frac{c}{2}$$



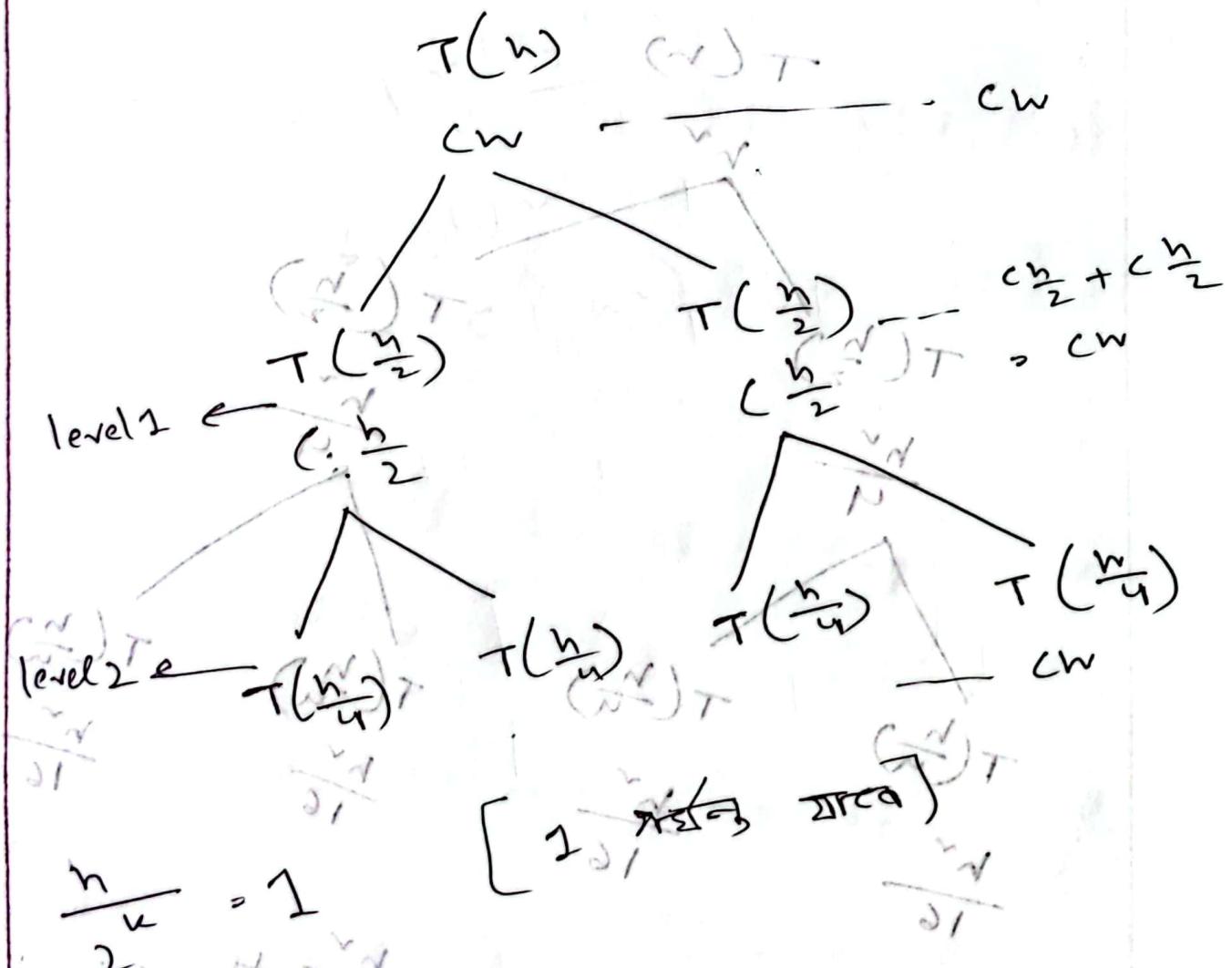
$$T\left(\frac{n}{2}\right) (w \text{ per level}) T\left(\frac{n}{2}\right)$$

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

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

$$n \rightarrow \frac{n}{2} \rightarrow \frac{n}{4} \rightarrow \frac{n}{8} \dots$$

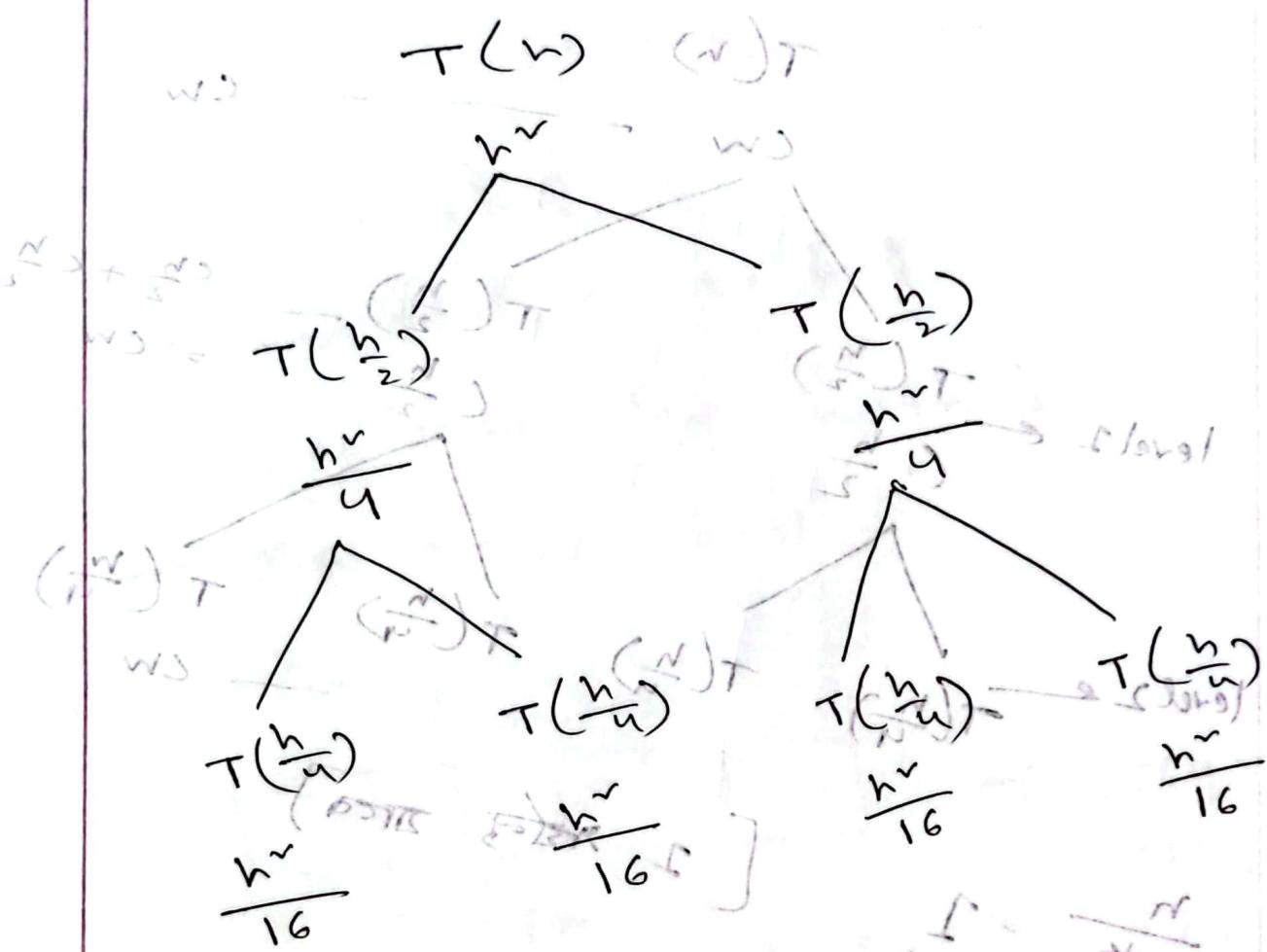
$$T(n) = 2T\left(\frac{n}{2}\right) + cn \quad \text{for } n > 0$$



$$\begin{aligned} k &= \log_2 n + \frac{n}{n} \quad \text{if } T(1) = c \cdot 1 \\ T(n) &= cn + cn + cn + \dots \\ &= cn \log n + cn \\ &\Rightarrow O(n \log n) \end{aligned}$$

129

$$T(n) = 2T\left(\frac{n}{2}\right) + h^v \approx n^{\alpha} T$$



$$\begin{aligned}
 T(n) &= h^v + 2 \cdot \frac{h^v}{4} + 4 \cdot \frac{h^v}{16} + \dots \\
 &= h^v \left(1 + \frac{2}{4} + \frac{4}{16} + \dots \right) \\
 &\approx h^v \left(1 + \frac{16}{2} + \frac{1}{4} + \dots \right)
 \end{aligned}$$

P.T.O.

E-A2d

$$\text{noten } \Rightarrow n \cdot \frac{1}{1 - \frac{1}{2}} \text{ anzuwenden}$$

$$(e^{i\theta} \text{ pol. } \tilde{w}) \tilde{h}^{\tilde{w}} \left(\frac{2}{1 - \frac{1}{2}} \right) T_n = (w) T \#$$

$$\text{aus } O(n) = \frac{2^n}{O(n)} \text{ folgt weiter}$$

rechnen hier in 9

möglichkeit ist es die augen zu schließen

anzuschließen zu rechnen = 0

anzuschließen muss zu sparen

$$(e^{i\theta} \text{ pol. } \tilde{w}) \theta = (w) T \tilde{h}^{\tilde{w}} \text{ zu } z_i \quad ①$$

$$(e^{i\theta} \text{ pol. } \tilde{w}) \theta = (w) T \tilde{h}^{\tilde{w}} \text{ zu } z_i \quad ②$$

$$(e^{i\theta} \text{ pol. } \tilde{w}) \theta = (w) T \tilde{h}^{\tilde{w}} \text{ zu } z_i \quad ③$$

$$(w \text{ pol. } \tilde{w}) \theta = (w) T \tilde{h}^{\tilde{w}} \text{ zu } z_i \quad ④$$

$$(w \text{ pol. } \tilde{w}) \theta = (w) T \tilde{h}^{\tilde{w}} \text{ zu } z_i \quad ⑤$$

* Recurrence equation Master Formula

$$\# T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^k \log^p n)$$

where, $a > 1$, $b > 1$, $k \geq 0$ and p is real number.

n = input size of the problem.

a = number of subproblems

$\frac{n}{b}$ = size of each subproblem.

① if $a > b^k$, $T(n) = \Theta(n^{\log_b a})$

② if $a = b^k$

if $p > -1$, $T(n) = \Theta(n^{\log_b a} \cdot \log^{p+1} n)$

if $p < -1$, $T(n) = \Theta(n^{\log_b a})$

if $p = -1$, $T(n) = \Theta(n^{\log_b a} \cdot \log \log n)$

$$\log^0 k = 1$$

$$(11) \quad a < b^k + \left(\frac{N}{\delta}\right)T \leq C_1 T^{\alpha} \quad \forall T > 0$$

if $p > 10$, $T(n) = \Theta(n^{\nu} \log^{\rho} n)$

if $P \in O$, $T(n) = \Theta(n^k)$

$$T(w) = 2T\left(\frac{w}{2}\right) + h$$

$$\Rightarrow a = 2, \quad \lambda_0 = 2\lambda_2, \quad k = 2, \quad P = 0$$

$$b^{\vee} = 2^{\sim} = 4$$

$$a < b^u$$

$$T(w) = \theta(r^{\sim} \log^{\circ} w)$$

$$= \theta(n^v)$$

$$* T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n^2) \quad (1)$$

$a=2, b=2, c(n)=n^2, d(p)=9$ if

$$a < b^{1/p} \Rightarrow n^2 < n^{\log_2^2}, \log_2^2 n > 2n$$

$$\therefore T(n) = \Theta(n^{\log_2^2})$$

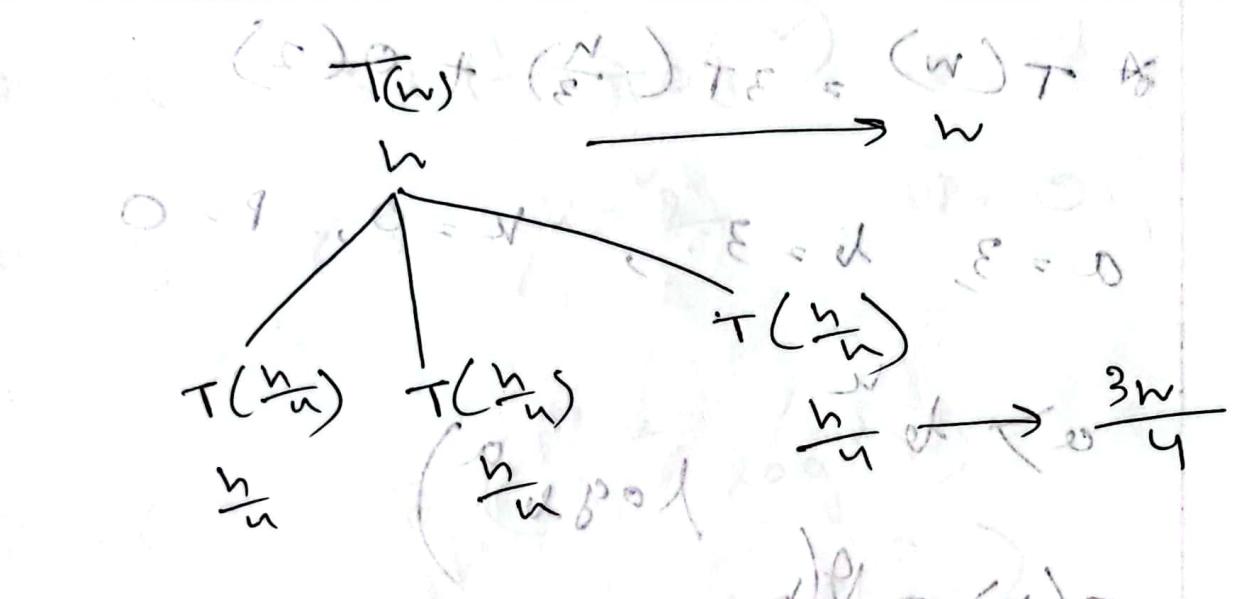
$$= \Theta(n^{\log_2^2})$$

$$= \Theta(n^{\log_2^2})$$

~~$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n^2)$$~~

~~$$(n^{\log_2^2}) \Theta(n^2) = n^{\log_2^2}$$~~

~~$$(n^2) \Theta(n^2)$$~~



$$T(w) = 3T\left(\frac{w}{4}\right) + w$$

$$a = 3, \quad b = 4, \quad k = 2, \quad p = 0$$

$$a < b^k \quad p > 0$$

$$\therefore T(w) = \Theta(n^2 \log^2 w)$$

~~$\Theta(n^2 \log^2 w)$~~

$$\Rightarrow \Theta(n)$$

$$* T(n) = 3T\left(\frac{n}{3}\right) + \Theta(1)$$

$$a = 3, \quad b = 3, \quad k = 0, \quad P = 0$$

$$\cancel{a > b} \quad \cancel{\frac{n}{n}} \quad \cancel{(n)}^k \rightarrow (n)^k \quad (n)^k$$

$$T(n) = \Theta(n^{\log_3 3})$$

$$= \Theta(n^{\log_3 3})$$

$$= \Theta(n^{\cancel{k}}) \quad \cancel{k} \quad n = \cancel{c} \cdot \cancel{s} = n$$

$$0 < 9$$

$$n \cancel{c} \downarrow s$$

$$\therefore (n^9 \cancel{c} \cdot \cancel{s}) \Theta(n^9)$$

$$\cancel{c} \cdot (n^9 \cancel{s}) \Theta(n^9)$$

$$(n^9) \Theta(n^9)$$

$$\log^0 w = 1$$

~~$$\log^2 w = \log n$$~~

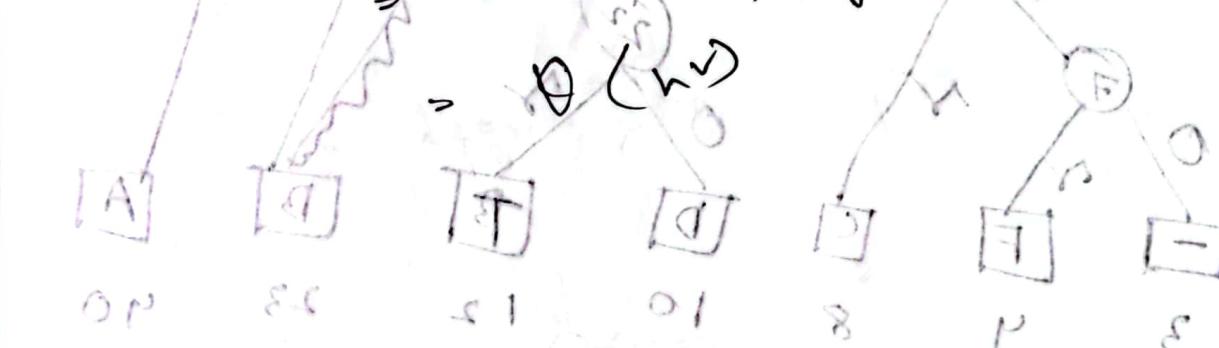
$$* T(n) = 3T\left(\frac{n}{3}\right) + O(n^{\sqrt{2}})$$

$$a = 3, b = 3, k = 2 \quad p = 0$$

$$a < b^k$$

$$\therefore T(n) = O(n^{\log^0 w})$$

$$= O(n^{\log^0 w})$$

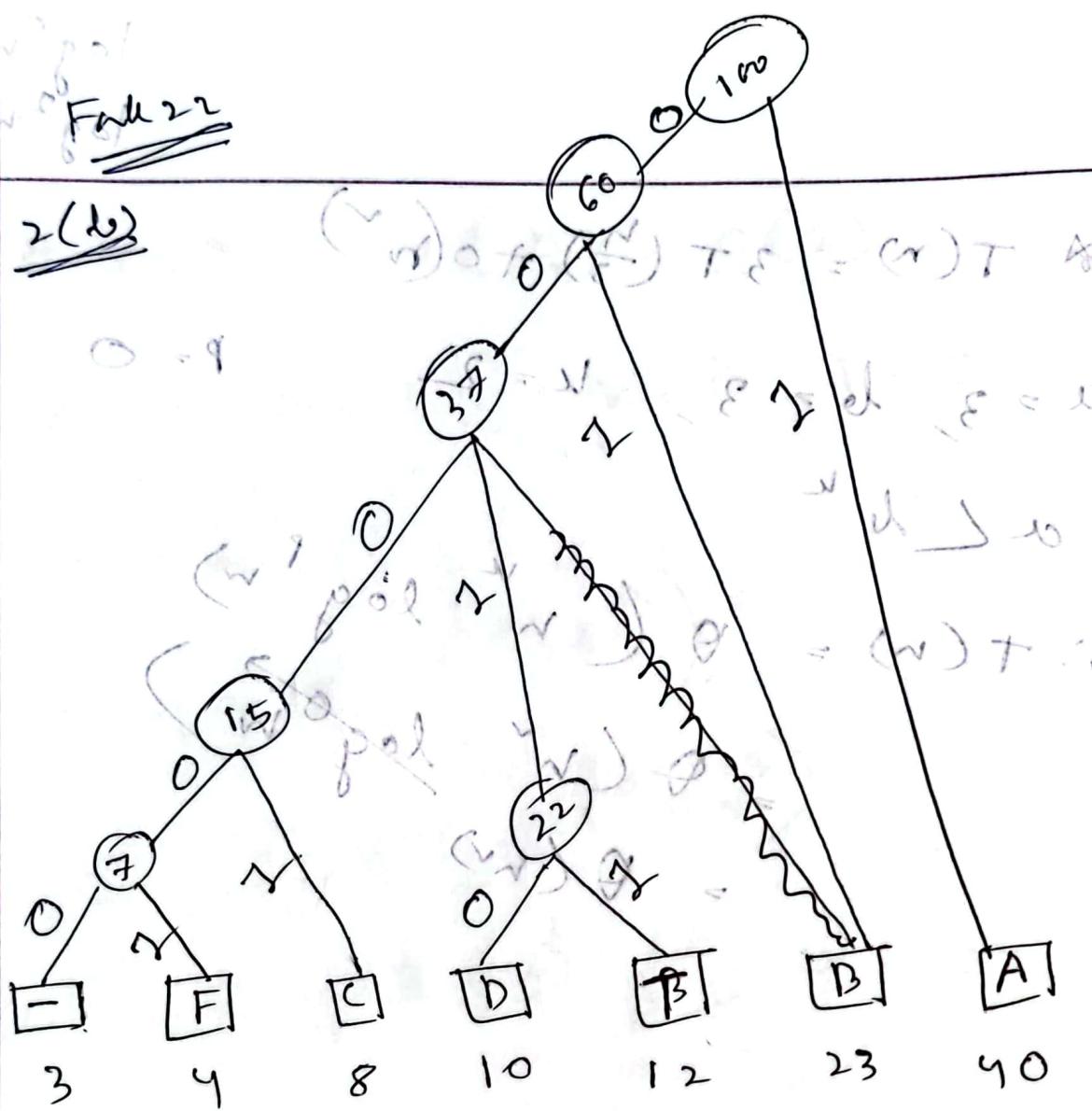


1111000110
AAA J A A A

AAA J A A A

~~Ques 2~~
~~Ques 2.2~~

~~2(b)~~



0111,0001,1111
B A ~~C~~ C AAA

B A ~~C~~ AAA

Graph representation

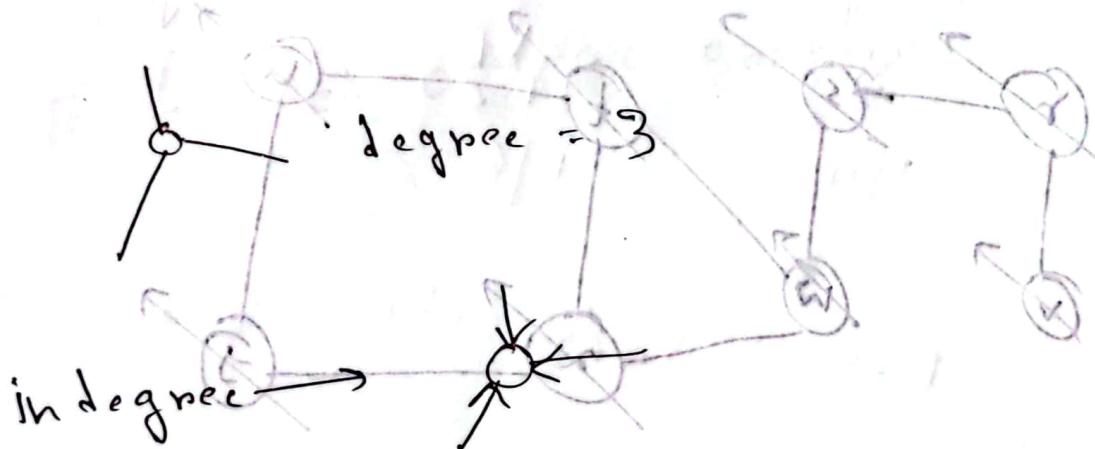
- Adjacency matrix → approach
- Adjacency lists → approach

(W) & G (adjacency)



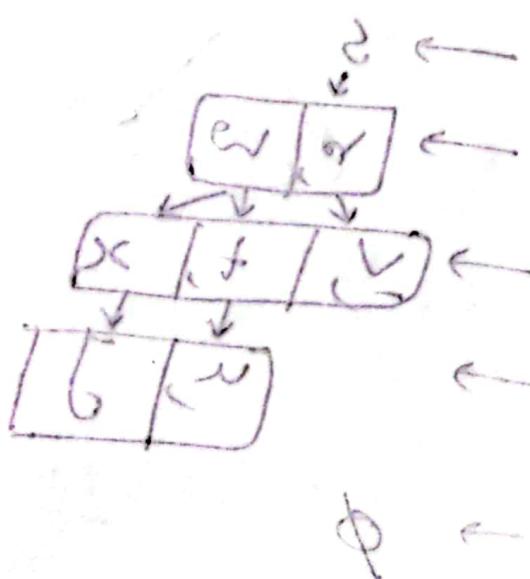
degree = 4

87B

outdegree \rightarrow

$$(E + V)^T = (W)^T$$

Needs of been on
smif bnd



check:

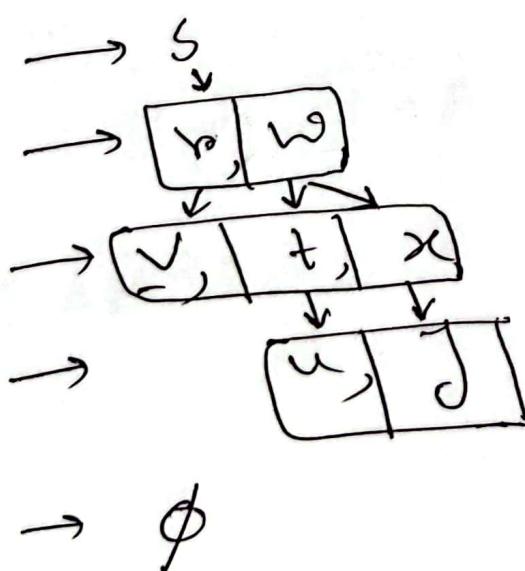
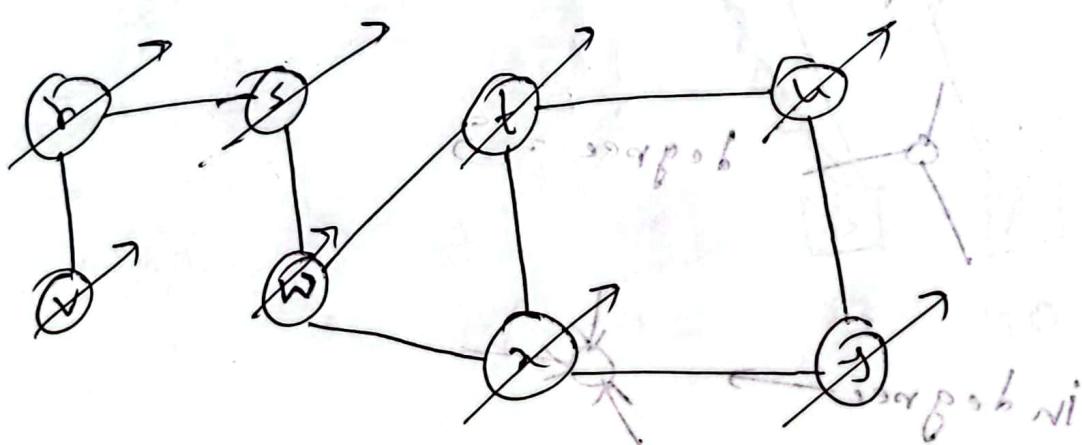
visit first row or column

indegree \rightarrow row check

outdegree \rightarrow column check

Complexity $\Rightarrow O(n)$

BFS



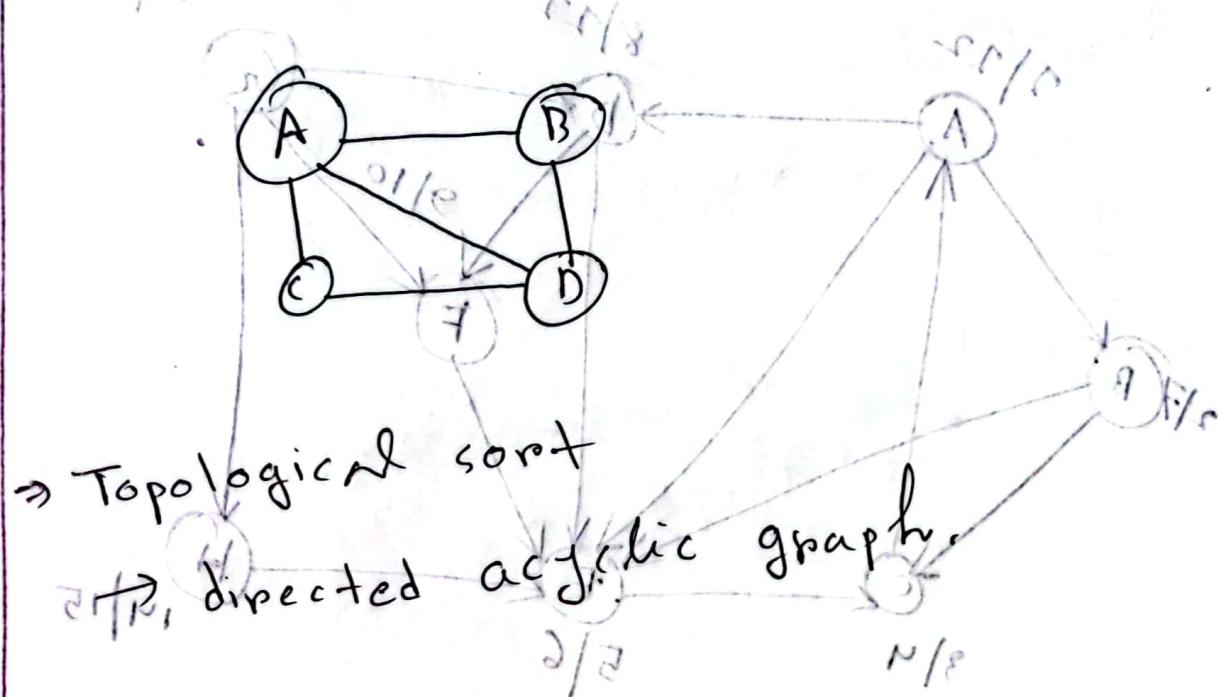
$\rightarrow \leftarrow$ \rightarrow \leftarrow \rightarrow \leftarrow \rightarrow \leftarrow \rightarrow \leftarrow \rightarrow

 $T(n) = O(n + E)$

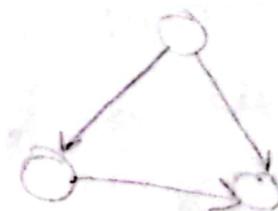
if visited,
no need to check
2nd time

A.21

BFS → Disconnected graph



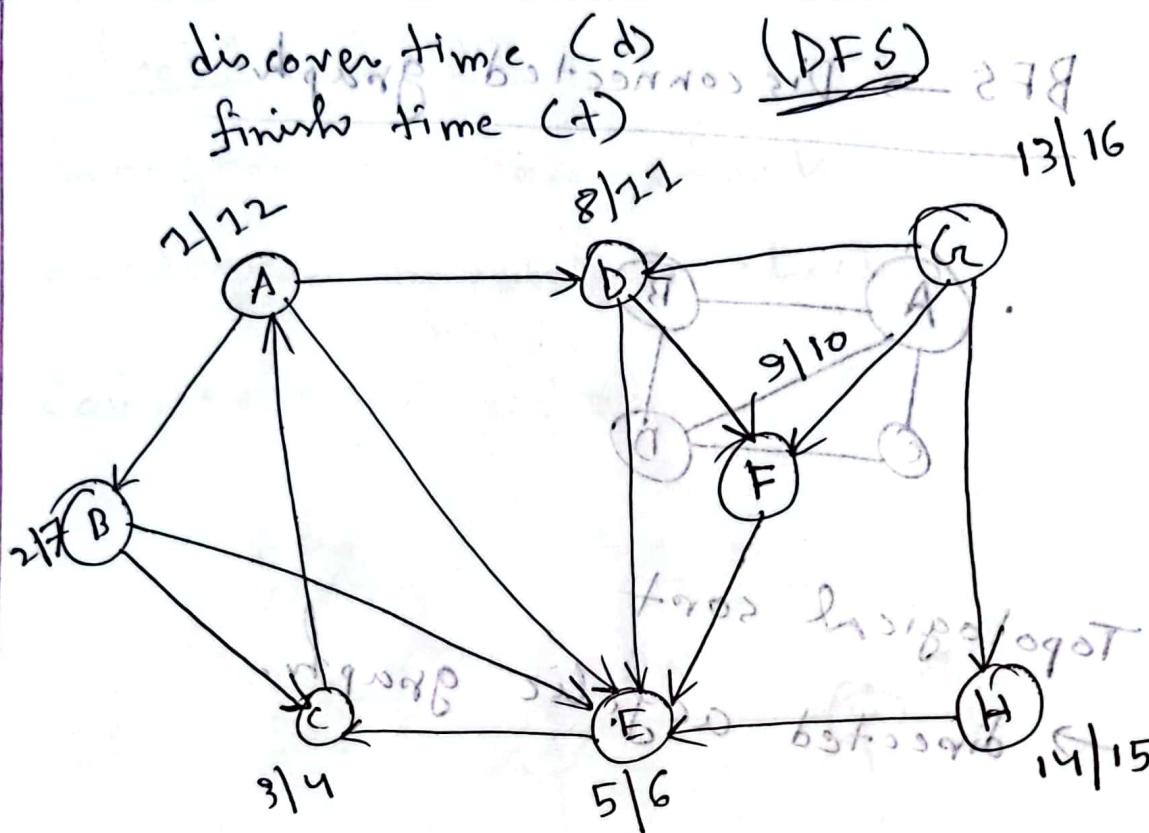
(iii) Adjacent sibling algorithm



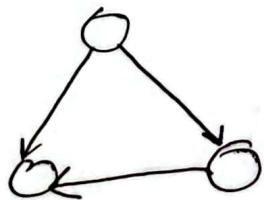
Q.29

11.11.23

DSA-2



Directed Acyclic Graph (DAG)



P.J.

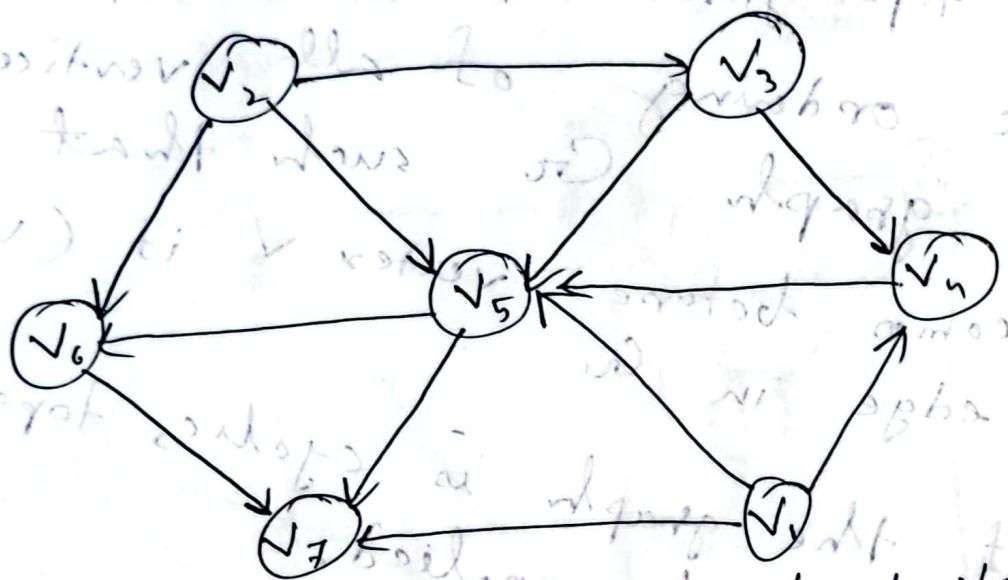
A topological sort of a DAG is a linear ordering of all vertices of the graph G such that vertex v comes before vertex u if (v, u) is an edge in G .

If the graph is cyclic, topological sort cannot be applied.

Topological sort (G)

1. call $\text{DFS}(G)$ to complete finish time $t[v]$ for each vertex v .
for each vertex v , insert it into the front of a linked list.
2. As each vertex is finished, insert it onto the front of a linked list of vertices.
3. return the linked list of vertices.

Time: $O(V+F)$.



Vertex

in degree

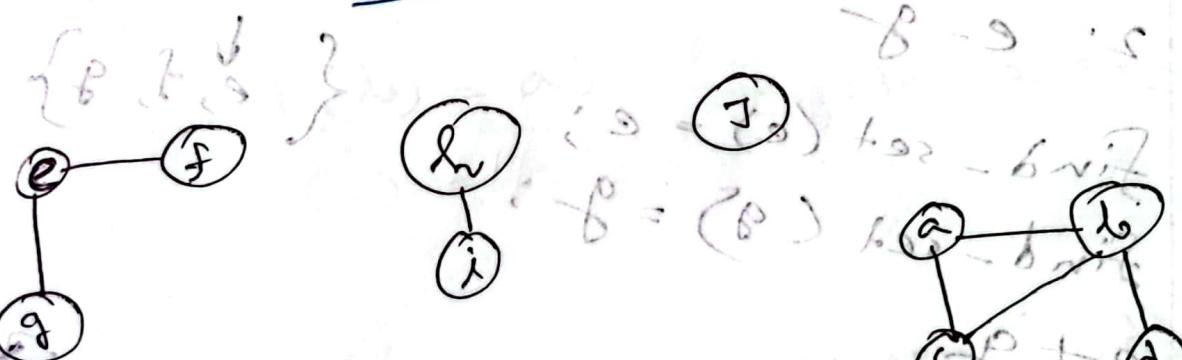
v_1	(0)
v_2	$\times \emptyset$
v_3	$\times \emptyset$
v_4	$\times \emptyset$
v_5	$\times \emptyset$
v_6	$\times \emptyset$
v_7	$\times \emptyset$

~~(2,3) min 2, 1, 2, 3~~ → topological sort

~~{2,3} f → {1,2}~~ Disjoint set

~~1,2,3,4,5,6,7,8~~ Disjoint set

~~1,2,3,4,5,6,7,8~~ (2,3) min 2, 1, 2, 3
Disjoint sets



make-set(x):

$\{x\}$, $\{x\}$

Algorithm:

for each edge (v, u) in $E(G)$

if $\text{find-set}(v) \neq \text{find-set}(u)$

$\text{Union}(v, u)$,

2. $e-f$ flugpilot und $\{e, f\}$
union (e, f)

find-set (e) = e ;

$\{e, f\}$

find-set (f) = f ;

$e \neq f$

2. $e-g$

find-set (e) = e ;

$\{e, f, g\}$

find-set (g) = g ;

$e \neq g$

3. $h-i$

find-set (h) = h ;

union (h, i)

find-set (i) = i ;

$\{h, i\}$

$h \neq i$

$\{i\}, \{hi\}, \{hi\}, \{i\}$

4. $a-b$

union (a, b)

find-set (a) = a ;

$\{a, b\}$

find-set (b) = b ;

(v, v) weis v

$a \neq b$

5. $a - c$

~~find-set(a) = a^i ; $|V| = \{a, b, c\}$~~

~~find-set(c) = c^j ; $|V| = \{a, b, c\}$~~

$a \neq c$

6. $b - d$

~~find-set(b) = b^i~~

~~find-set(d) = d^j~~

~~cycle exists~~

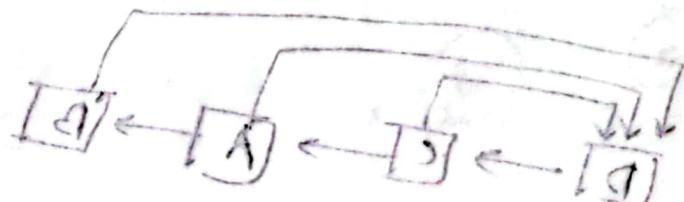
7. $b - d$

~~find-set(b) = b^i~~

~~find-set(d) = d^j~~

~~union(r, d)~~

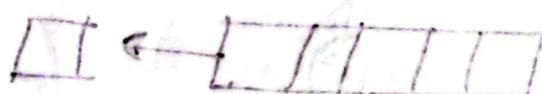
$a \neq d$



8. $\{i\} \rightarrow \{j\}$

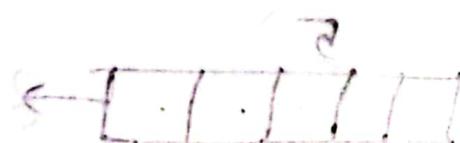
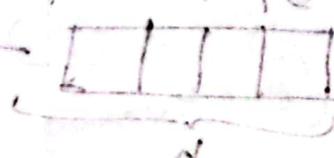
$(w) \circ = \text{insertion}$

$(w) \circ$



getro \(\bar{n}\) \(\bar{z}\)

(w, 0) \(\circ\)

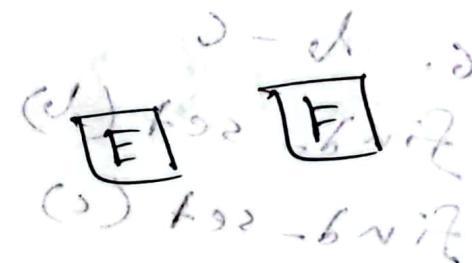
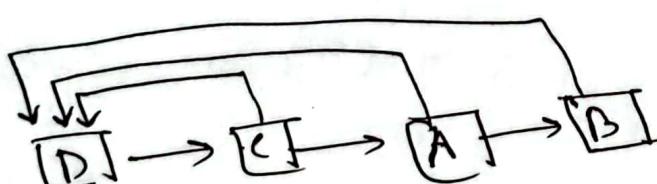
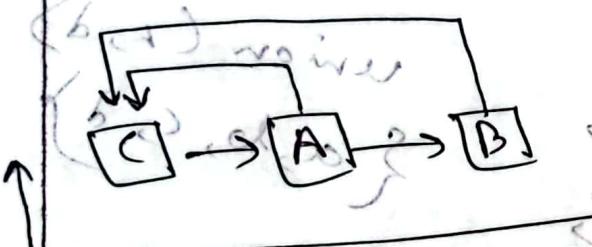
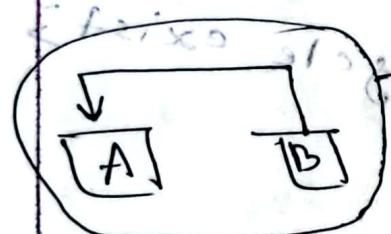


Cost analysis

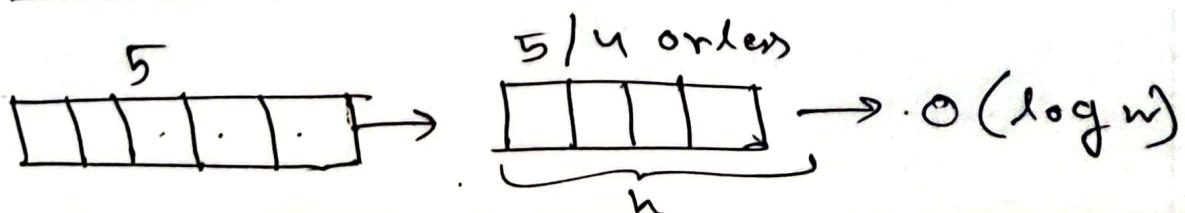
Union call = $V - k$ $\rightarrow O(n)$ for worst

Find-set = $2|E|$ $\rightarrow O(n)$ for worst

Connected components



Worst case = $O(n^2)$

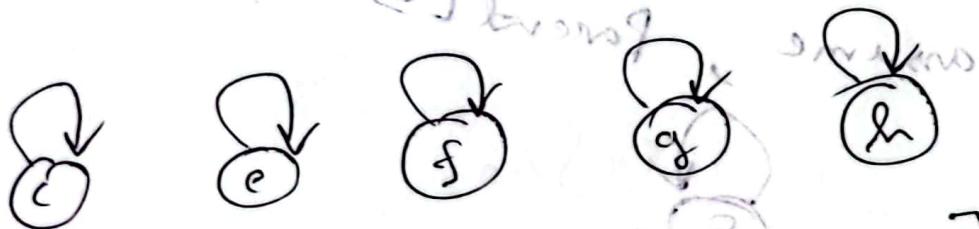


* $c \rightarrow h$ $\ell = (c, h)$ work

$c \rightarrow e$ $\ell = (c, e)$ not work

$f \rightarrow g$ $\ell = (f, g)$ not work

$f \in [5]$ break



[make-set]

$\text{find-set}(c) = c$

$\text{find-set}(h) = h$

$\text{parent}[h] = c$

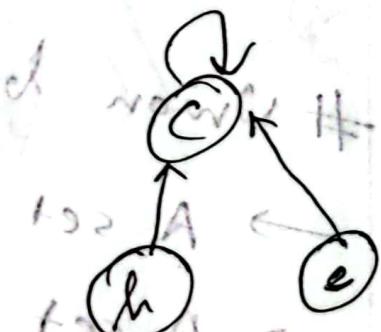
$\text{find-set}(c) = c$

$\text{find-set}(e) = e$ still not same

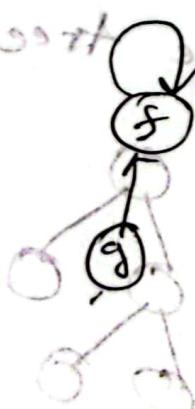
$\text{parent}[e] = c$

$\text{find-set}(f) = f$

$\text{find-set}(g) = g$
 $\text{parent}[g] = f$



sort point



Union (e , g)

find-set(e) = c

find-set(g) = f

assume Parent [c] = f



Union by rank \Rightarrow (nl) for - brif

→ A set of large tree

→ A set of small tree.

small tree will be the child of
large tree



G.W
2 1 2 2 3

Disjoint-set

Make-set (x)

$$P[x] \leftarrow x$$

Union (x, y)

$a = \text{Find-set}(x)$ work for x ; $*$

$b = \text{Find-set}(y)$ work

$P[a] = b$ it $\rightarrow \{x\} \cup \{y\}$ work

Find-set (x)

if $x \neq P[x]$ return $\text{Find-set}(P[x])$

return $P[x]$

path compression

$P[x] = \text{Find-set}(P[x])$

P.T.

Union ~~by rank~~

Make-set(x)

— $\xrightarrow{\text{rank}[x] \leftarrow 0}$ (x) has own
rank $[x] \leftarrow 0$

Link(x, y)

* if $\text{rank}[x] > \text{rank}[y]$ then

 then $P[y] \leftarrow x$, $bwif = 0$

 else $P[x] \leftarrow y$

* if $\text{rank}[x] = \text{rank}[y]$ then

$\text{rank}[x]++, bwif = 1$

 then $P[x] \leftarrow x$, $bwif = 1$

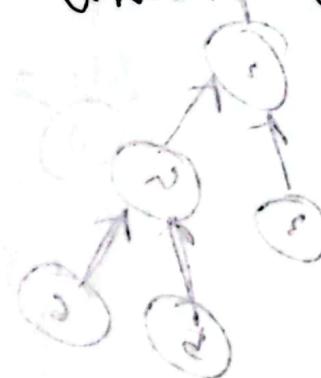
Union (x, y)

link($\text{find-set}(x), \text{find-set}(y)$)

$y \leftarrow \text{find-set}(y)$, $bwif = 1$

Find-set(x)

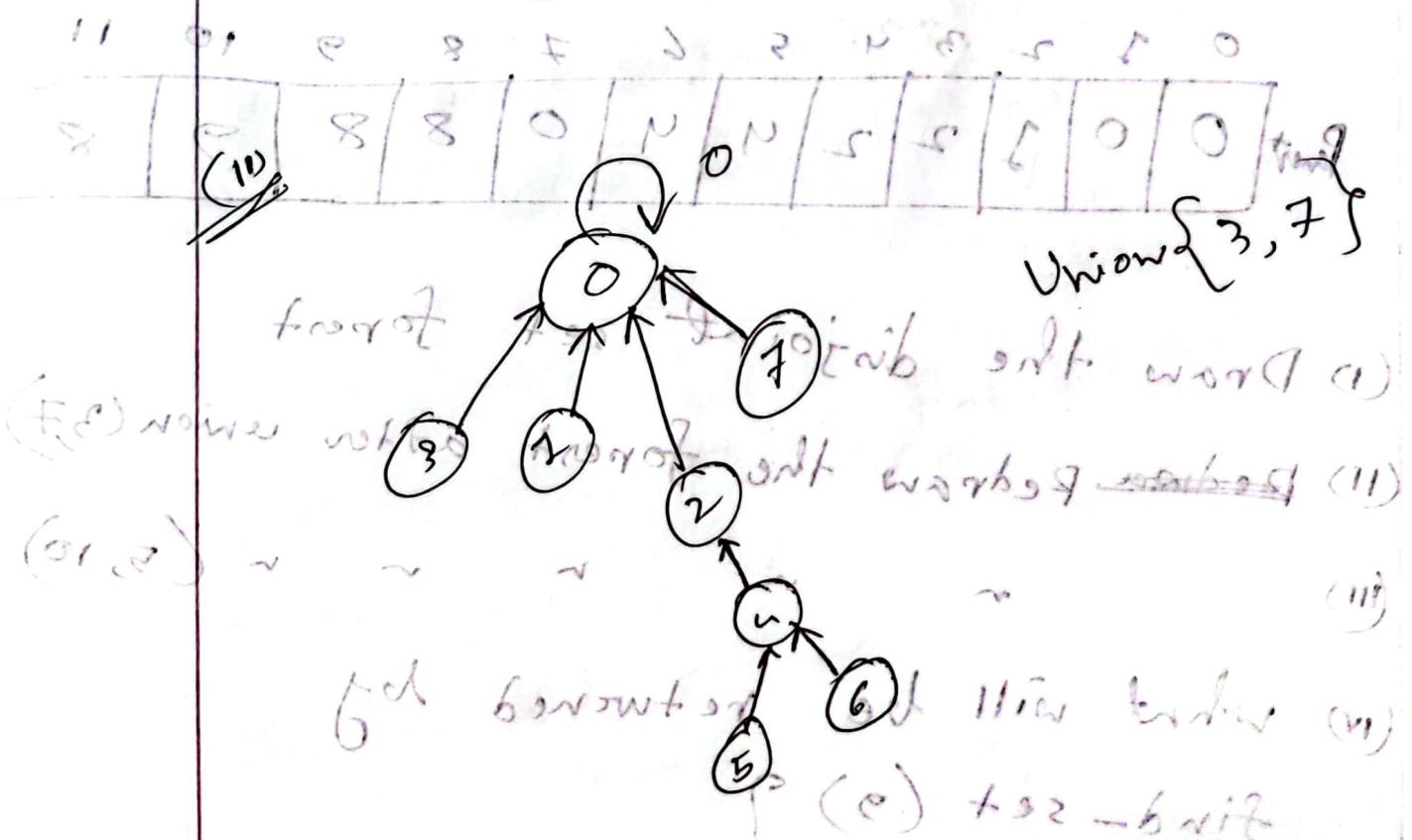
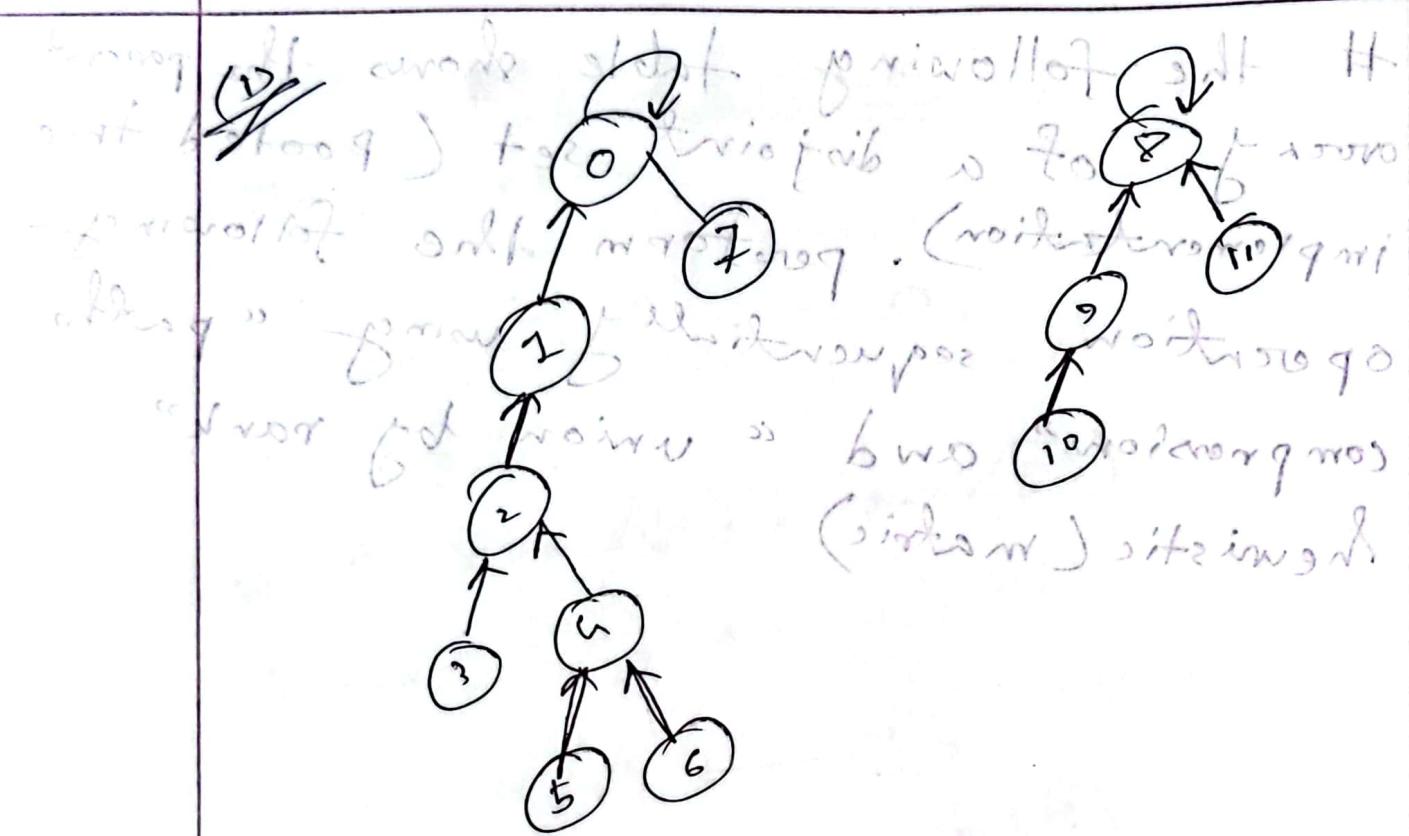
the following table shows the parent array of a disjoint set (rooted tree implementation). perform the following operation sequentially using "path compression" and "union by rank" heuristic (matrix)

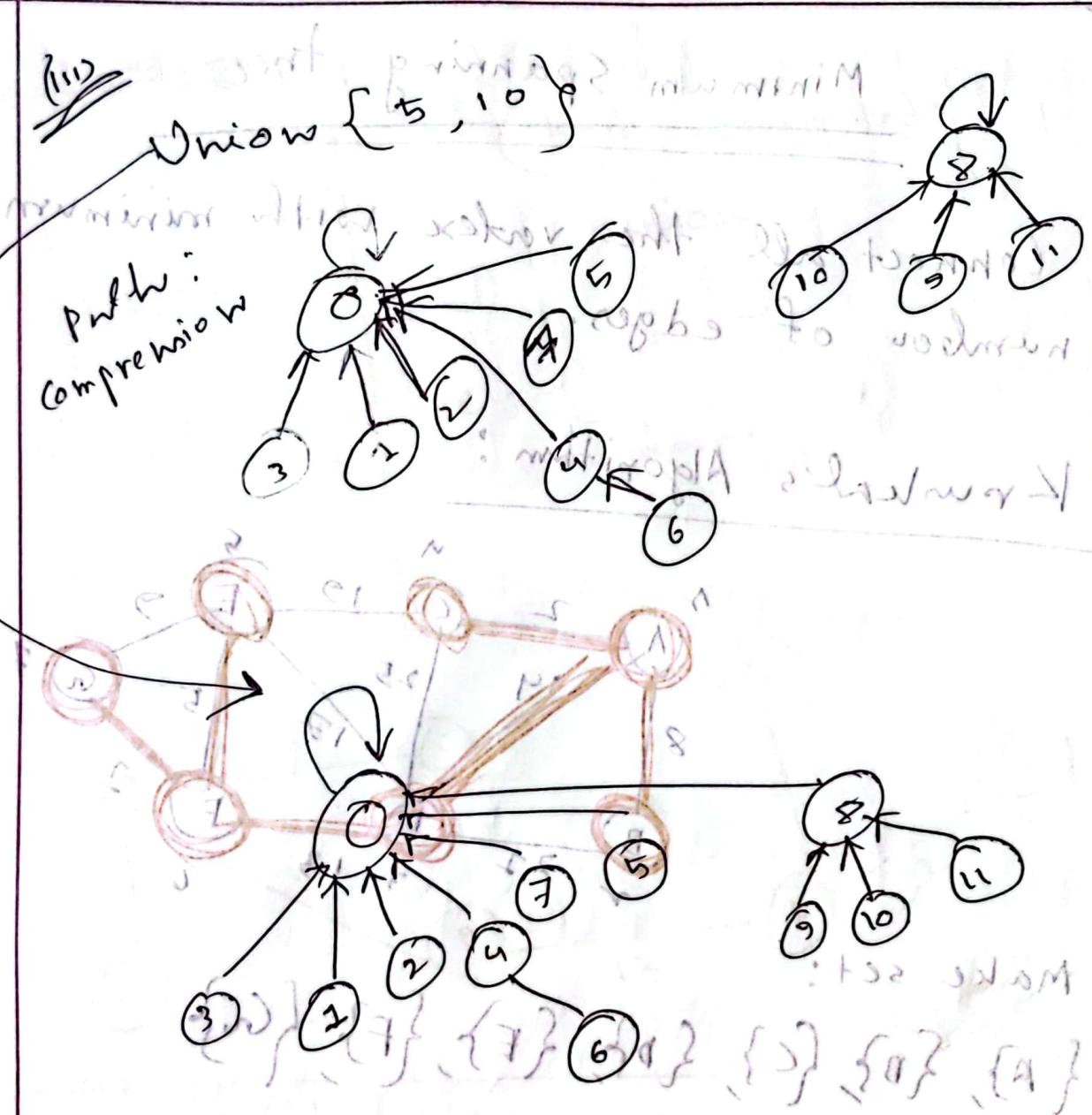


	0	1	2	3	4	5	6	7	8	9	10	11
Parent	0	0	1	2	2	4	4	0	8	8	9	8

Ques. Exercise

- (i) Draw the disjoint set forest
- (ii) ~~Redraw~~ Redraw the forest after union (3,7), (5,10)
- (iii)
- (iv) what will be returned by find-set (9)?





Find-set(5) = 0
 Find-set(10) = 0

0 <= 7 - A
 A - D = 0
 X + 1 = 0
 X + 1 = 0
 X + 1 = 0
 X + 1 = 0
 X + 1 = 0

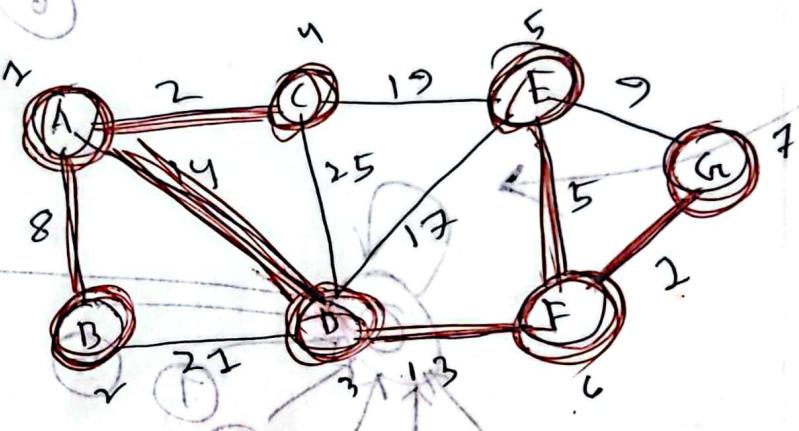
1.W
4.12.23

DSA 2

Minimum Spanning tree

Connect all the vertex with minimum number of edges.

Kruskal's Algorithm:



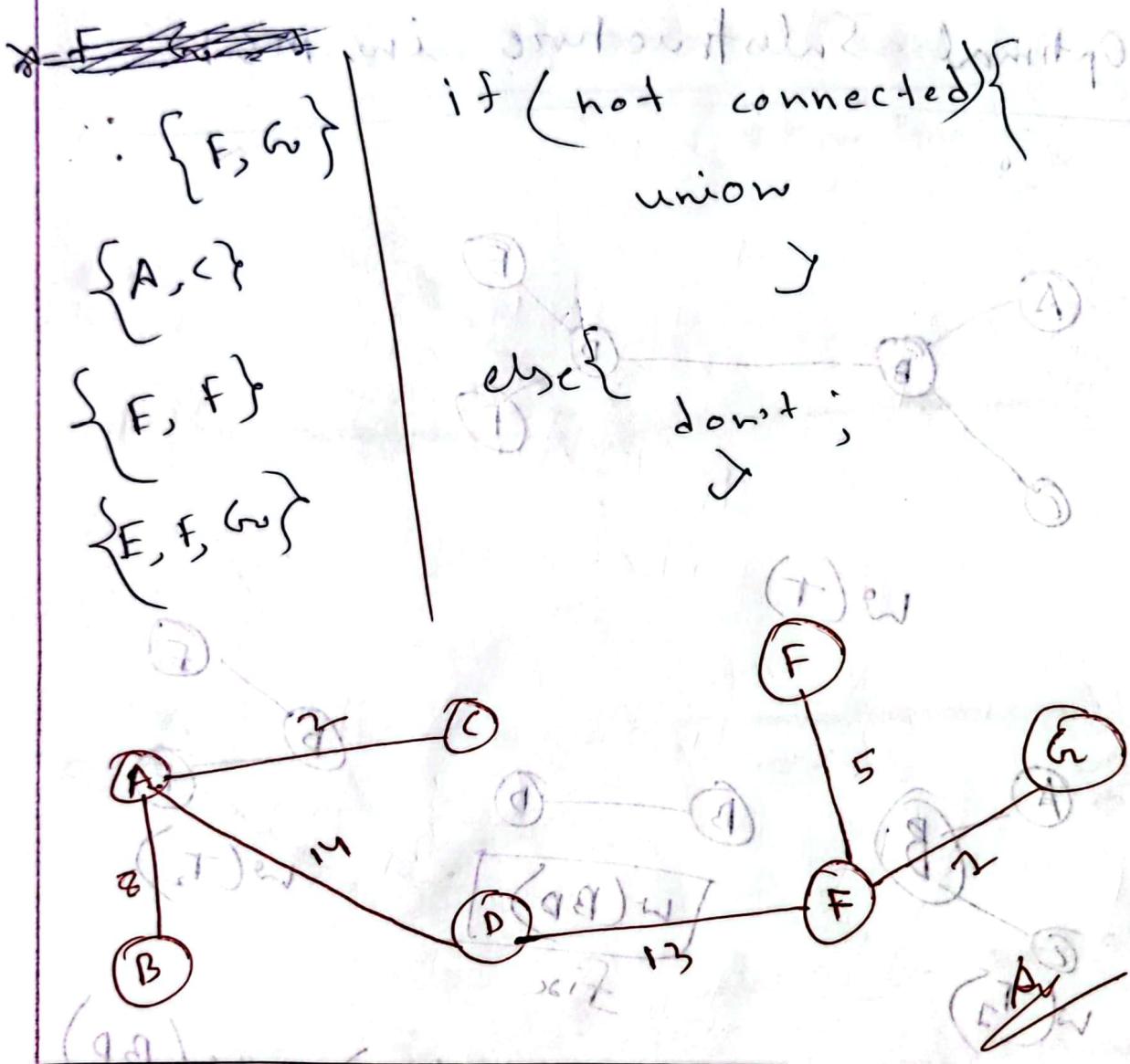
Make set:

{A}, {B}, {C}, {D}, {E}, {F}, {G}

Sort the edges in Ascending Order:

F-G \Rightarrow 1	D-F \Rightarrow 13 ✓
A-C \Rightarrow 2	A-D \Rightarrow 14 ✗
E-F \Rightarrow 5	D-E \Rightarrow 17 ✗
A-B \Rightarrow 8	C-E \Rightarrow 19 ✗
E-G \Rightarrow 9	B-D \Rightarrow 21 ✗
	C-D \Rightarrow 25 ✗

connection check



Next Tuesday

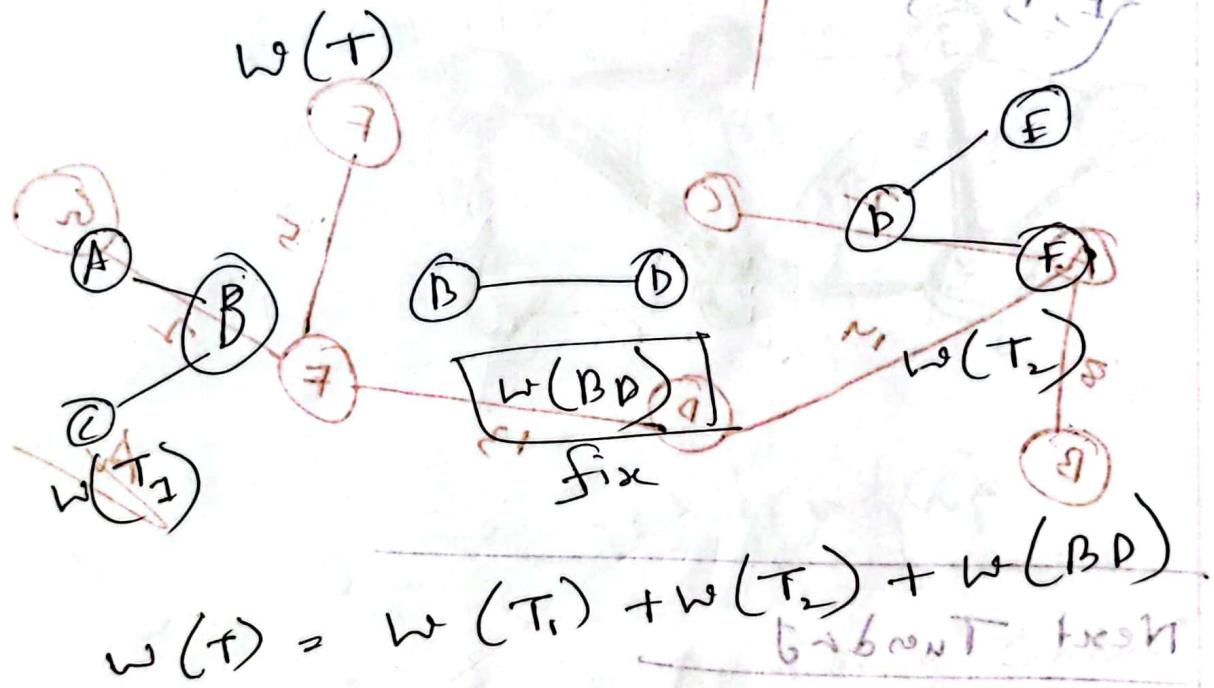
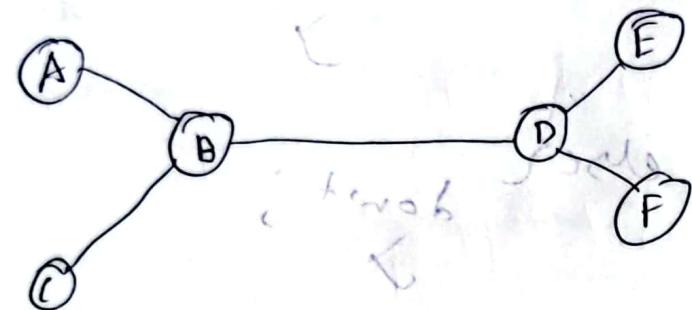
19.12.23

MST (Prims, Kruskal)

shortest path (Dijkstra, Bellmanford)

Wiederholung

Optimal Substructure in MST



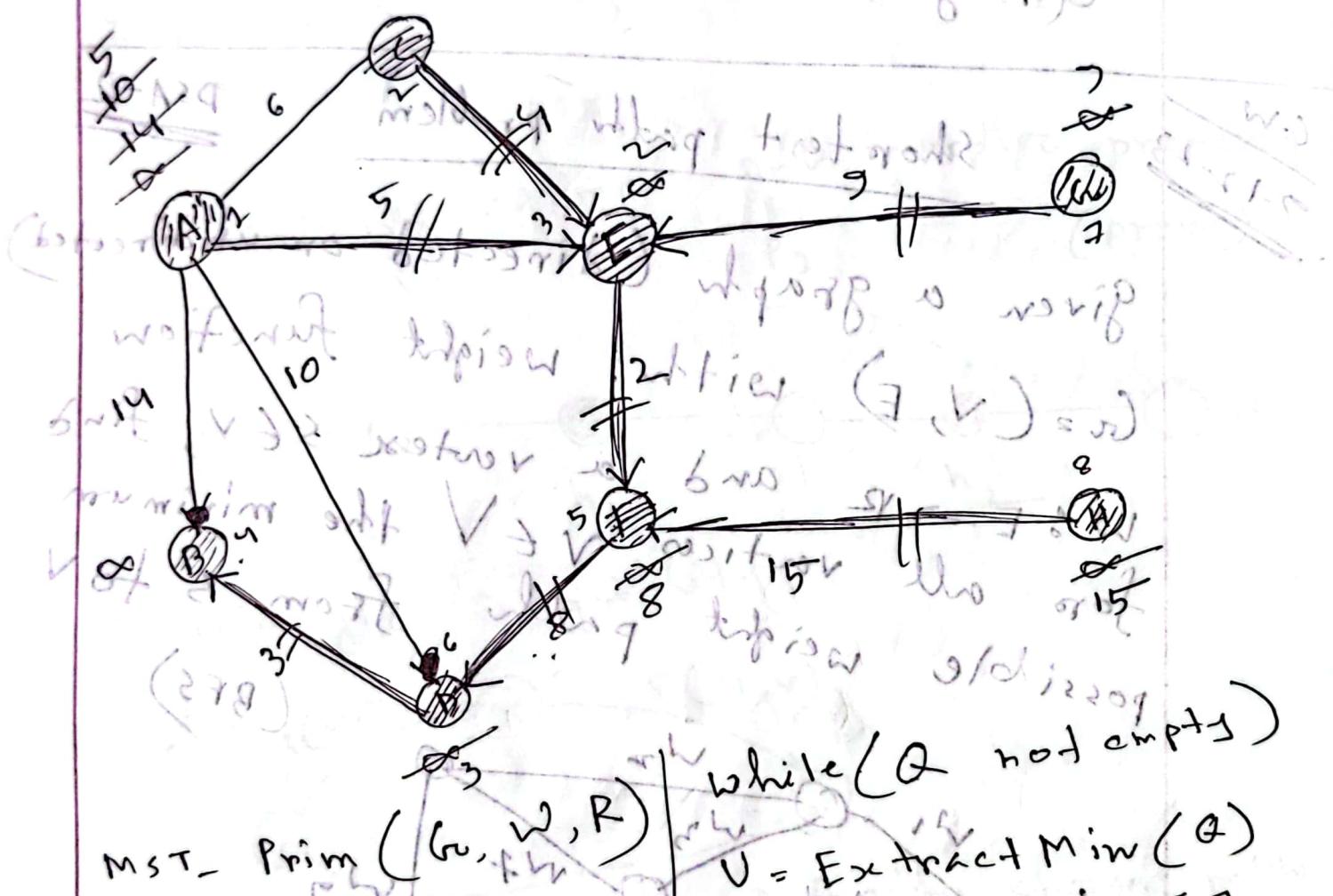
$$w(T) = w(T_1) + w(T_2) + w(BD)$$

So they are optimal.

(bottom up, min) $T \in M$

~~6.6.2~~ Minimum Spanning Tree with

(Min Priority Queue)



MST-Prim (G_0, w, R)

$$Q = V[G_0]$$

for each $v \in Q$

$$\text{key}[v] = \infty$$

$$\text{key}[R] = 0;$$

$$p[R] = \text{NULL};$$

while (Q not empty)

$v = \text{ExtractMin}(Q)$

for each $v \in \text{Adj}[v]$

if ($v \in Q$ & $w(v, v) < \text{key}[v]$)

$$\text{key}[v] = w(v, v);$$

$$p[v] = v$$

$$\text{key}[v] = w[v, v];$$

~~Min Priority Queue~~ ~~Priority Queue~~ key

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

CW
9.12.23

Shortest path problem

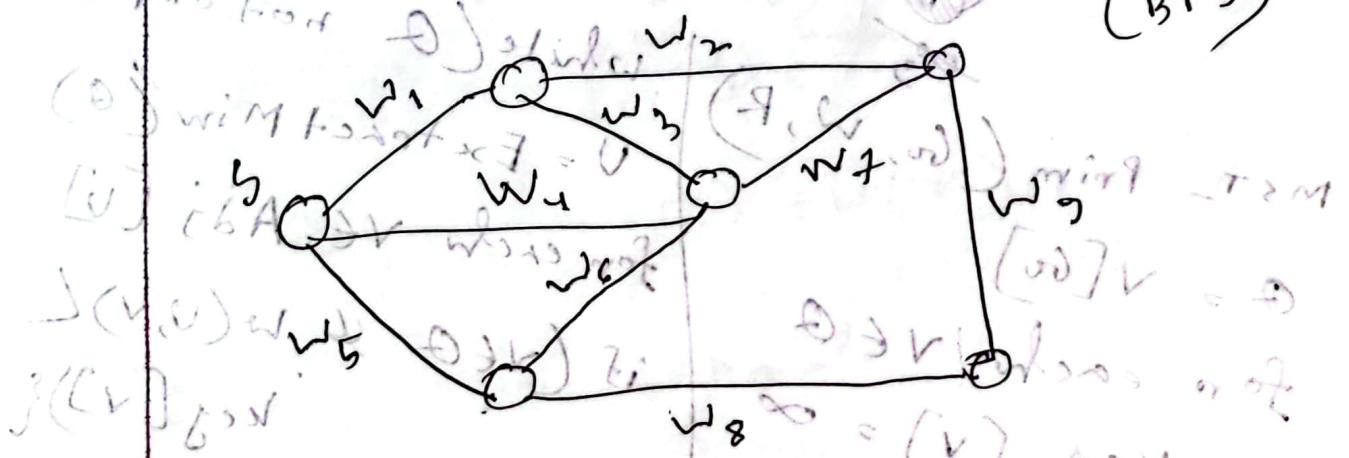
PSA-2

given a graph (directed or undirected)

$G = (V, E)$ with weight function

$w: E \rightarrow \mathbb{R}$ and a vertex $s \in V$, find for all vertices $v \in V$ the minimum possible weight path from s to v

(BFS)

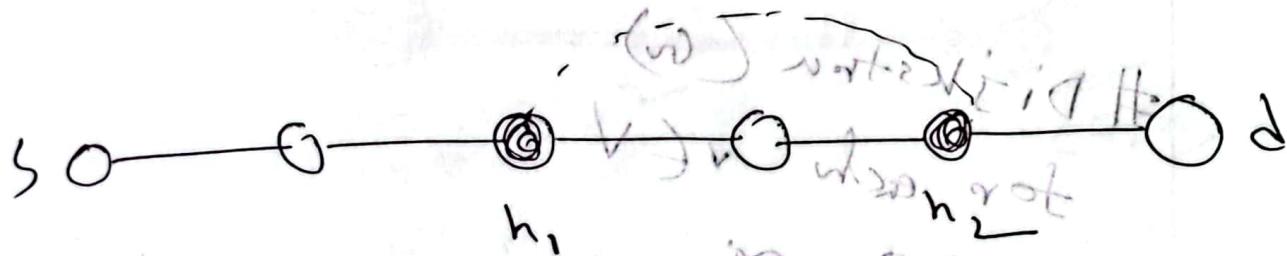


$$v = [v] \neq [v] \neq [v]$$

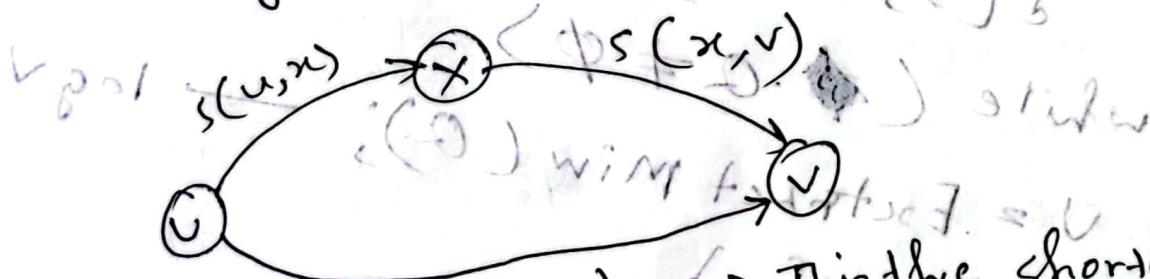
$$[v] \neq [s] \neq [v] \neq [v]$$

- * single source ✓
- * single pair ✓
- * All pairs → syllabus not X

→ Dijkstra follows optimal substructure property.
 P29 (prove)



→ Triangle inequality.

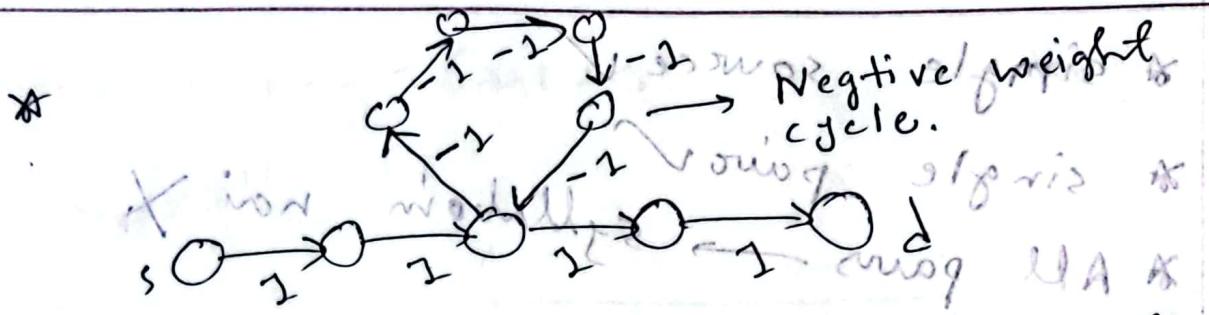


$s(u, v) \leq s(u, x) + s(x, v)$ This is the shortest path

$$s(u, v) \leq s(u, x) + s(x, v)$$

$$s(u, v) = s(u, x) + s(x, v)$$

$$(s(u, v))_0 \leq \underline{s(u, v)}_0$$



→ Every graph can not give a shortest path result. swallow IT

Dijkstra ($\mathcal{O}(V^2)$)

for each $v \in V$

$$\delta[v] = \infty$$

$$\delta[s] = 0$$

while ($Q \neq \emptyset$) $\log V$

$v = \text{Extract min}(Q)$

$S = S \cup \{v\}$

for each $u \in V \rightarrow \text{Adj}[u]$

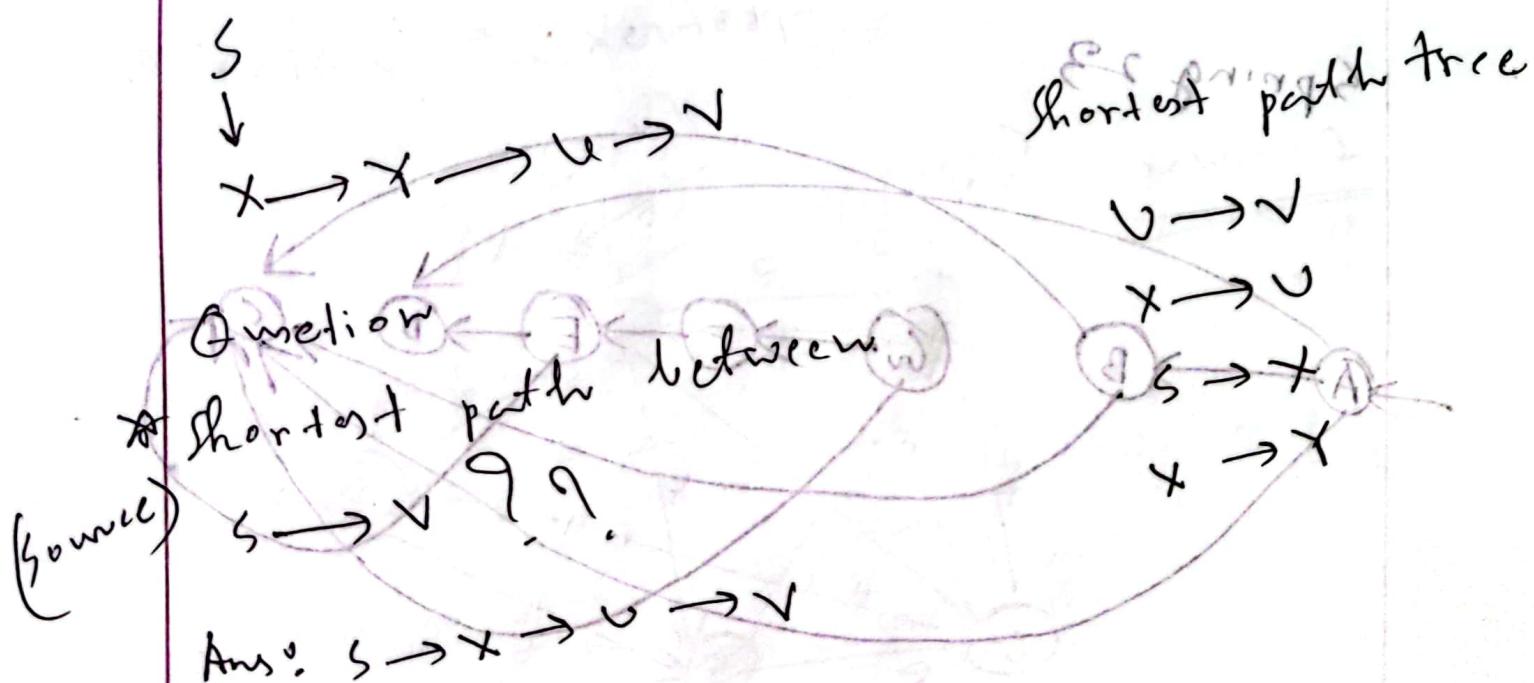
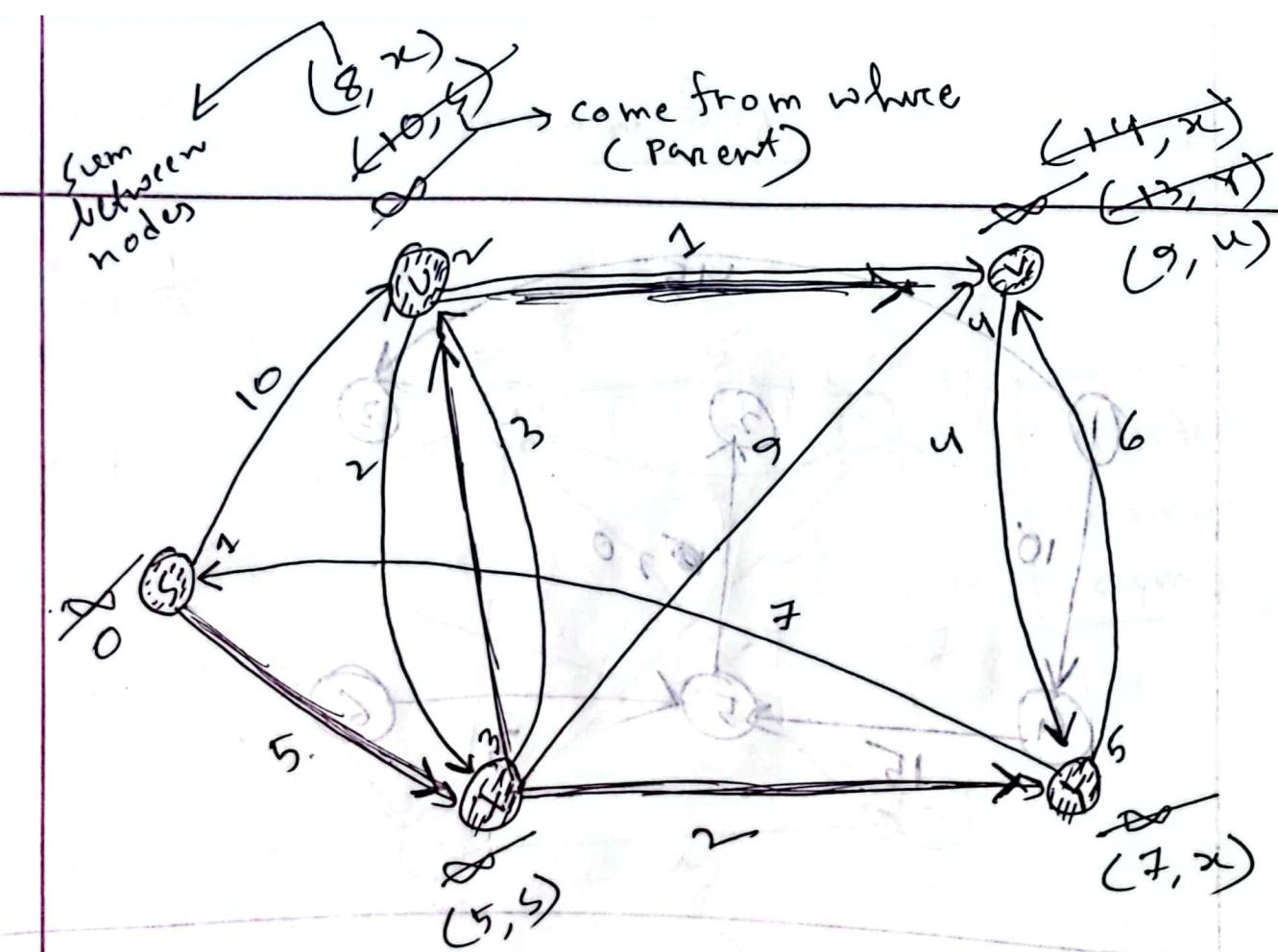
if ($\delta[v] > \delta[u] + w(u, v)$)

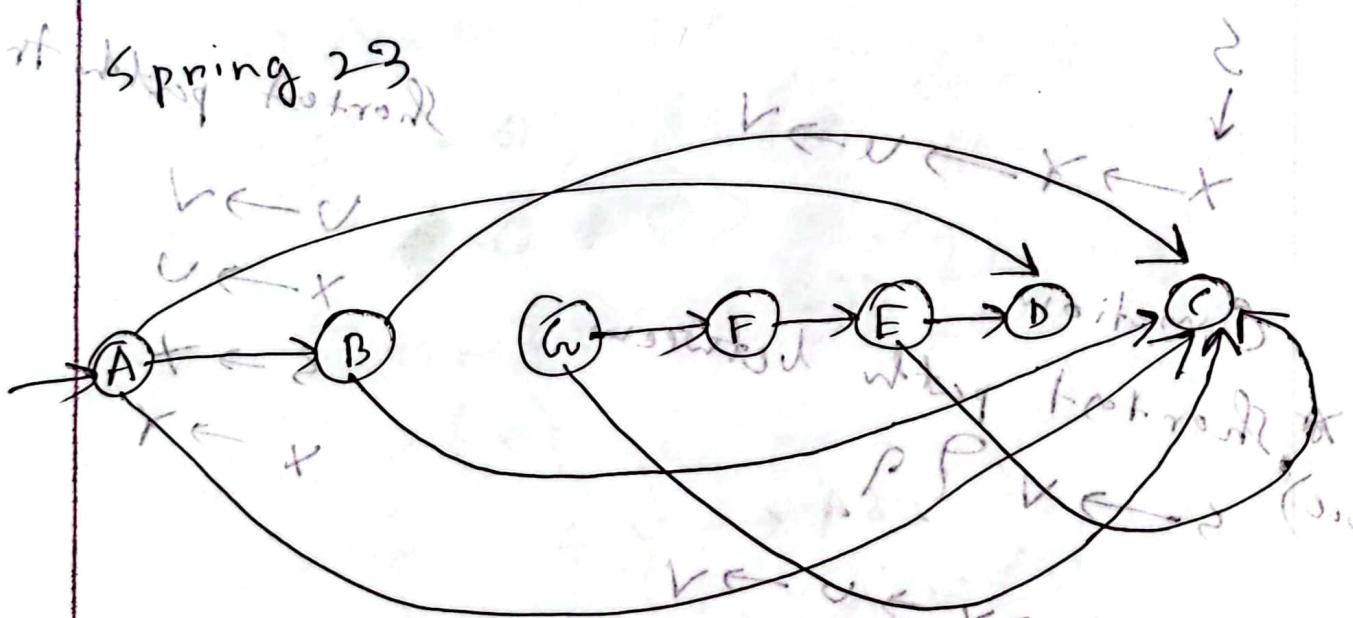
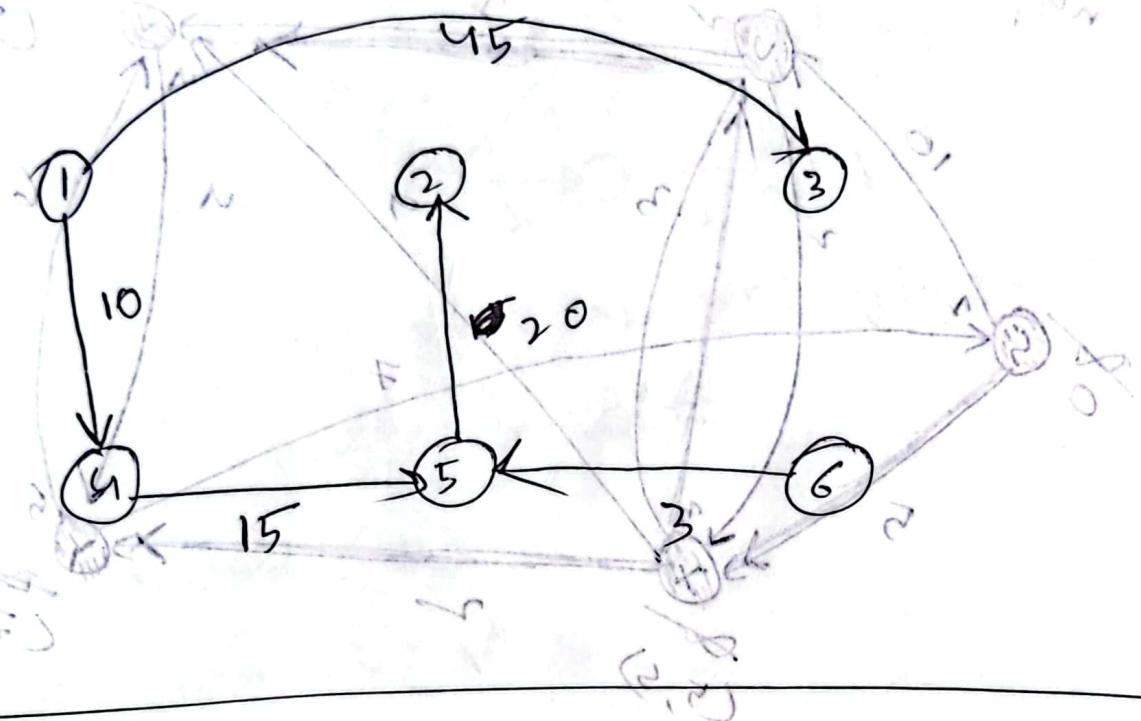
$$\delta[v] = \delta[u] + w(u, v)$$

time complexity $O(\log V)$

$$P[v] = u$$

$$O(\log V)$$





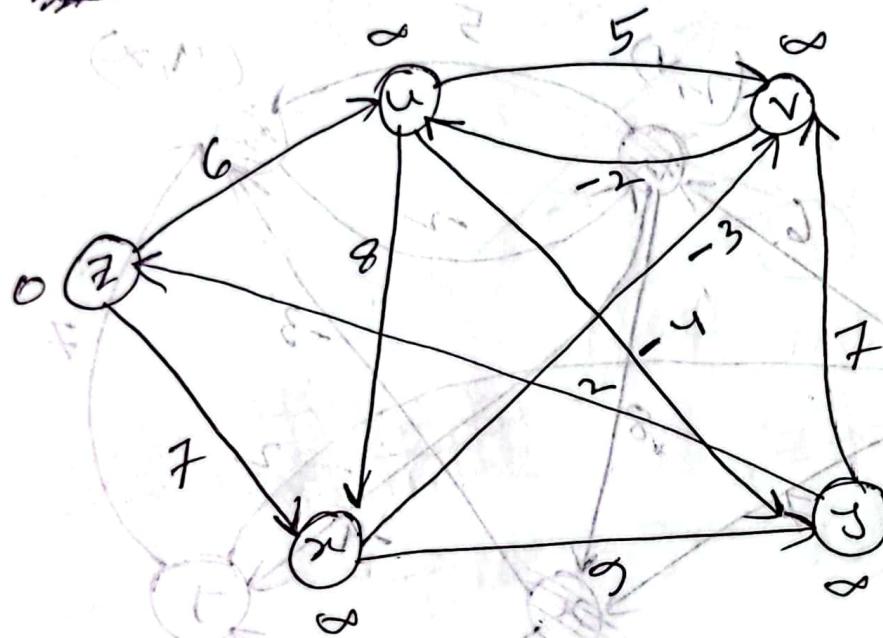
Topological sort
Dependency graph

~~6.2
11.12.23~~

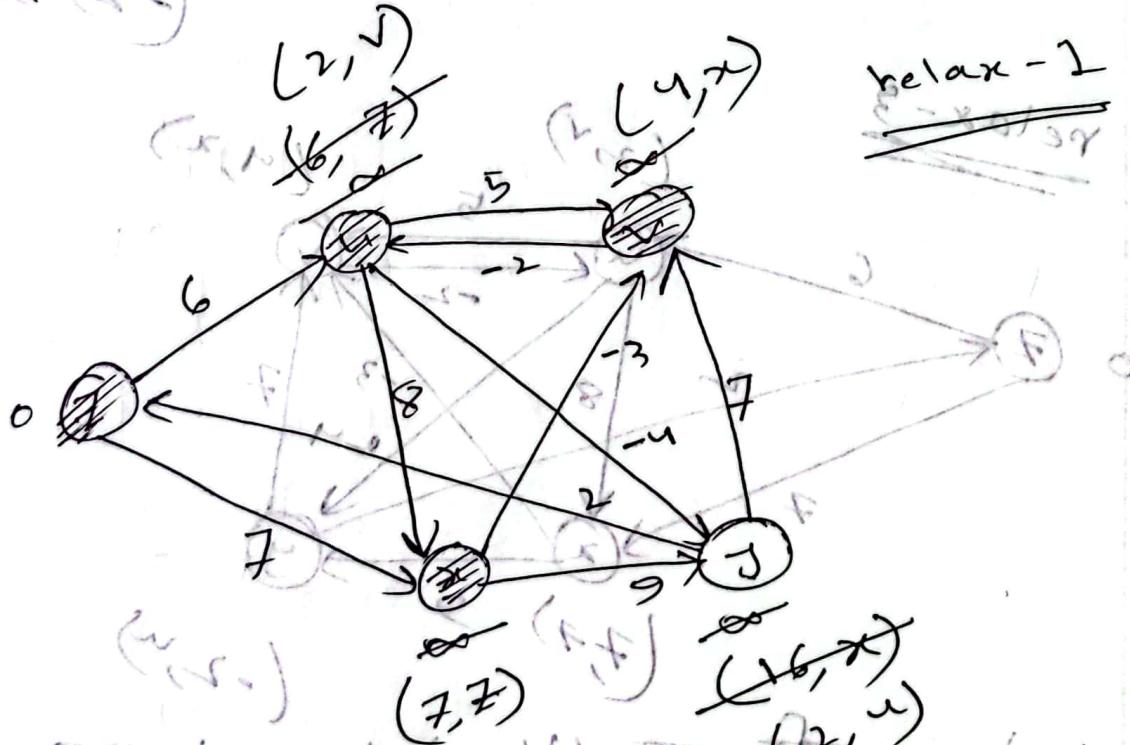
Bellman Ford

DSA-II

~~1~~



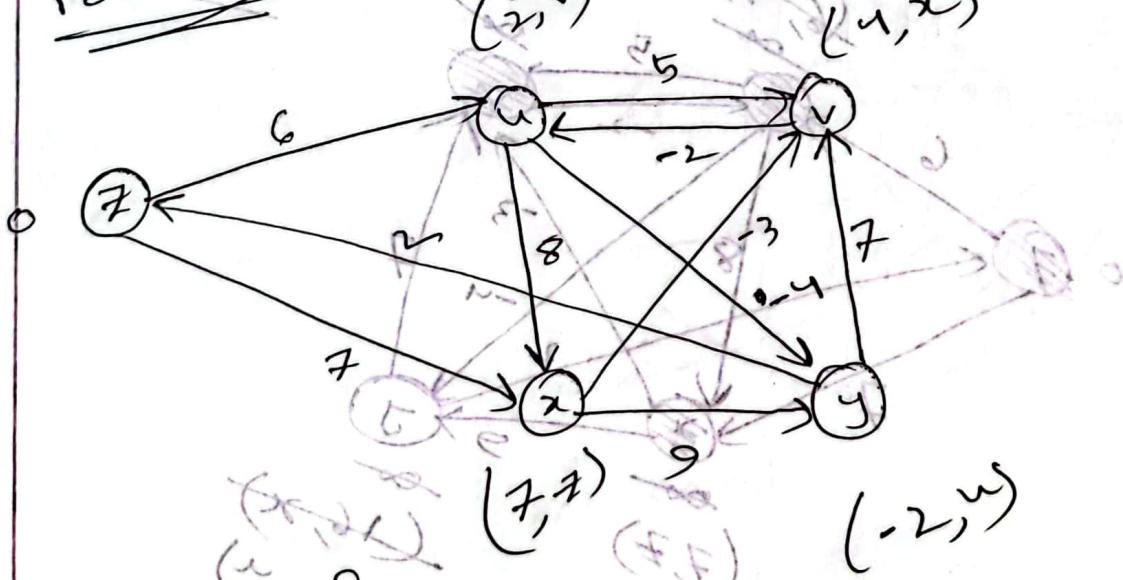
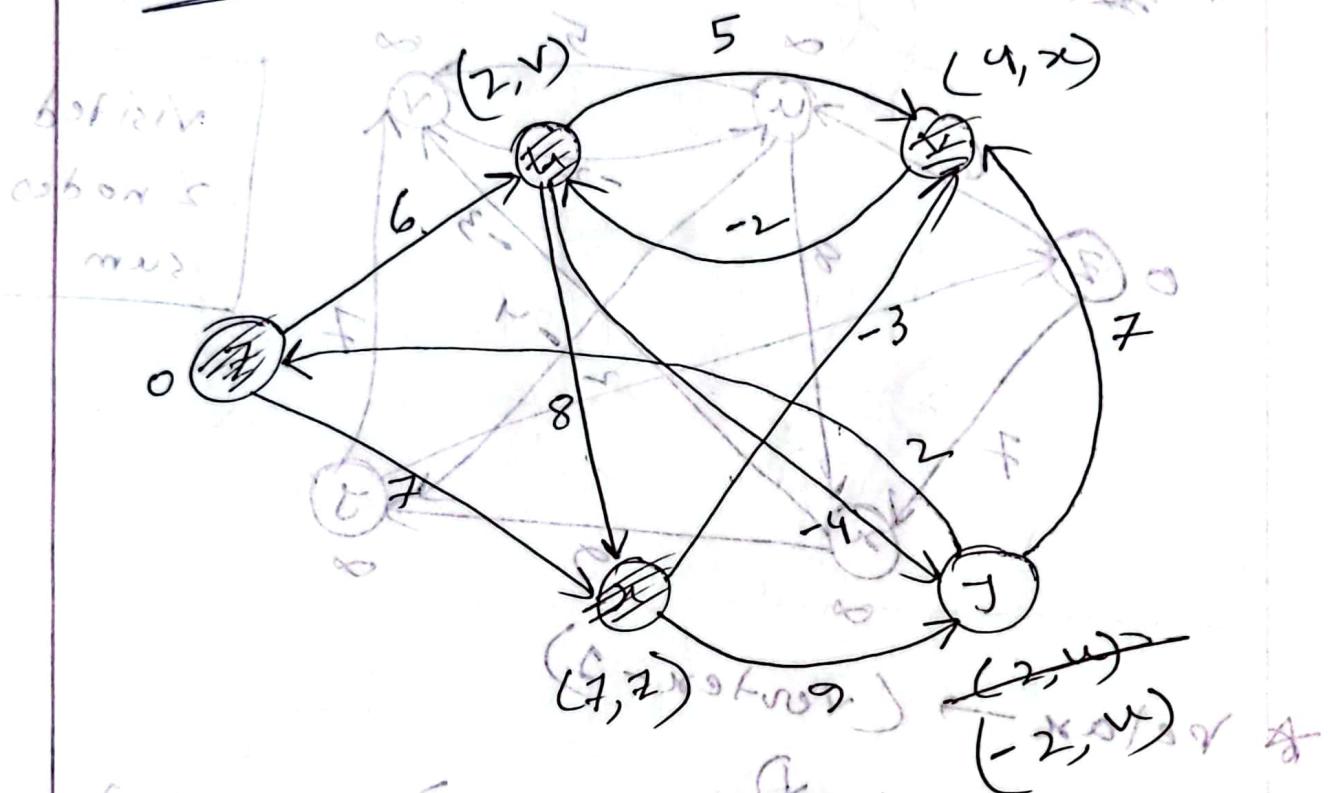
* relax \rightarrow (vertex 1)



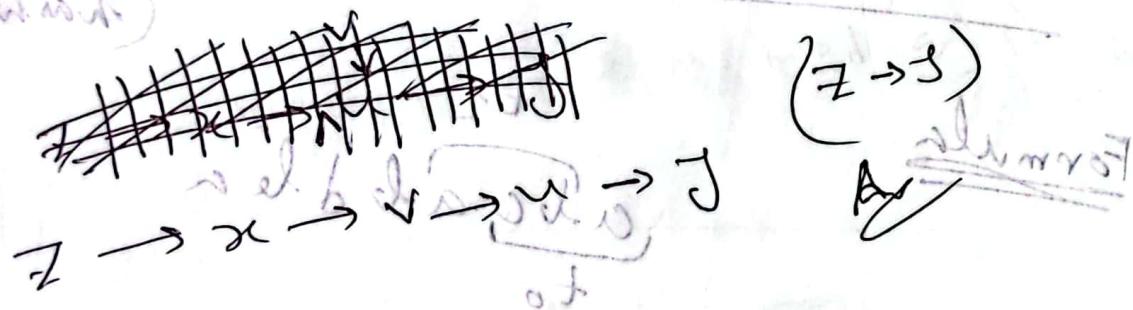
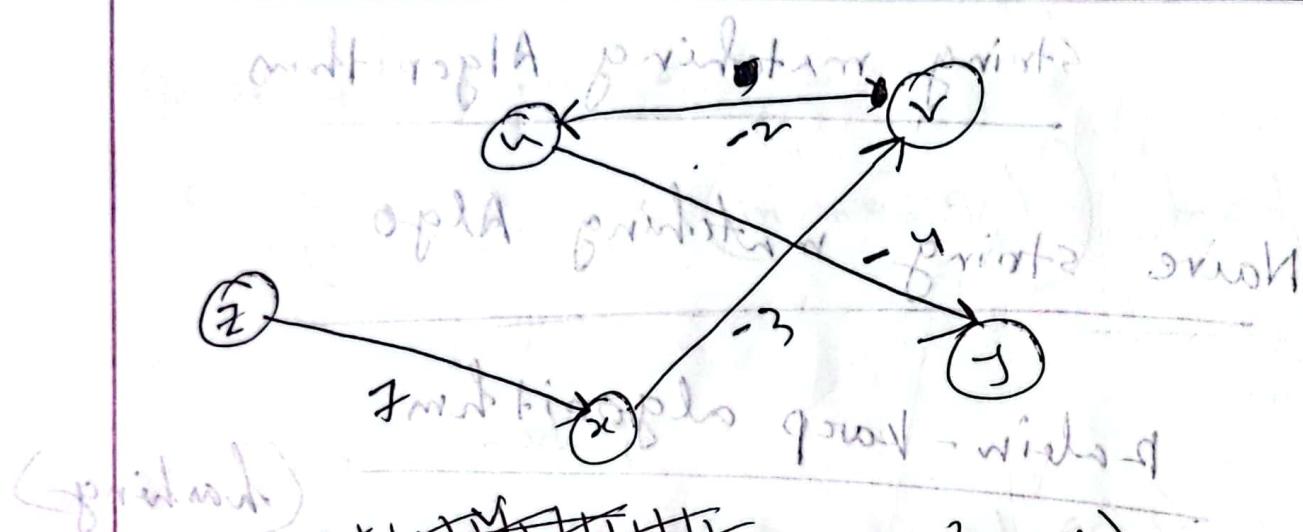
Now report (0-1) as the shortest
path via vertex 1 to vertex 0

TH DS

~~relax-2~~



relax with Δ $(V-1)$ target Δ Δ Δ
update Δ Δ Δ Δ Δ Δ Δ update
 Δ Δ Δ



multiple paths exist

multiple paths exist

multiple paths exist

prefix (comp - rep)

prefix

prefix

prefix

(shorter)

CW
14.12.23

DSA-II

String matching Algorithms

Naive string matching Algo

Rabin-Karp algorithm

(hashing)

Formula

$t \leftarrow \underbrace{a b c a b d}_{t_0} l e n$
 t_1, t_2, t_3, \dots

$T = \underbrace{43141}_{t_0} 592 \dots$
 t_1

$$\begin{aligned} & \textcircled{2} \{ (4314 - 400) \times 10 \} + 2 \\ &= 3140 + 2 \\ &\rightarrow 3141 \end{aligned}$$

$$t_1 = 3141$$

(rolling hash)

$$*(a+m) \bmod q$$

$$= ((a \bmod q) + (m \bmod q)) \bmod q$$

only one digit has been added

$$*(am) \bmod q$$

$$= ((a \bmod q) * ((m \bmod q))) \bmod q$$

~~$T = "daeiachreaf" \bmod 13$~~

$$a = 1$$

~~$P = "aei"$~~

$$b = 2$$

~~$h(P) = a \times 10^2 + e \times 10^1 + i \times 10^0$~~

$$d = 4$$

~~$= (100 + 50 + 9)$~~

$$e = 5$$

~~$\Rightarrow 159 \bmod 13$~~

$$f = 6$$

~~$g = 7$~~

~~$h = 8$~~

~~$i = 9$~~

~~$t_0 = dae = 415 \bmod 13$~~

~~$= 12$~~

$$\begin{array}{r} 13) 159(12 \\ - 13 \\ \hline 29 \\ - 26 \\ \hline 3 \end{array}$$

$$t_1 = ae \cdot i = 10(12 - 3 \times 10^0) + i$$

$$= -3872 \bmod 13 = 3$$

$$t_2 \left\{ f(t_1, g_2 \text{ result} - t_2 \text{ gain } \times 10^3) + b_2 \right\} +$$

t_2 Q? Last digit Q? Value

$$v_0 \text{ bond} = \left\{ \left[10 \left(3 - \frac{1}{100} \right) \right] + 2 \right\} \text{ v_bond}$$

$$-969 \mod 13$$

$$= \{ 969 + (75 \times 13) \} \mod 13$$

$$x \equiv 6 \pmod{13}$$

$$P = b \quad 01x_1 + 01x_2 + 01x_3 \quad \boxed{13) \text{ col } 4}$$

$$t_2 = 591 \text{ yrs}$$

16

6

2

\rightarrow) $\exists x \forall y$

100

$$i + (v_{01} \times b - s_1) \times i_{01} = i_{02} = i_2$$

80 Elbow 5430 -

$$x - 9 = 13, \quad d = 10 \quad P = 15 \quad \% 13$$

so d

$$\frac{d}{P} = \frac{10}{15}$$

$$1 + \{01 \times (01 \times 1 - 01)\} = 3$$

$$\begin{array}{r} 13) 159(12 \\ -13 \\ \hline 29 \\ -26 \\ \hline 3 \end{array}$$

String mismatch error = 6 or 2

Rabin-Karp

$S = T$

$$e = i \quad 8 + \{01 \times (01 \times 1 - 01)\} = 2$$

$$T = "daeiia hea" \quad a = 2$$

$$P = "hea" \quad b = 2$$

$$c = 3$$

$$h(P) = 8 \times 10^0 + 5 \times 10^1 + 1 \times 10^2 \quad d = 4 \quad (nD)$$

$$= 851 = 6 \quad e = 5$$

$$f = 6$$

$$g = 7$$

$$h = 8$$

$$h(T_0) = dae$$

$$= 1 + \{01 \times (1 \times 10^0 + 2 \times 10^1 + 5 \times 10^2)\} = 6$$

$$(b-a+1)^2 = 9 \times 5 = 45 \quad \% 13 = 12$$

$$h(T_1) = ae$$

$$= \{01 \times (4 \times 10^0 + 1 \times 10^1) + 1 \times 10^2\} = 159$$

$$\Rightarrow 159 \% 13 = 3$$

daeria hea

$$h_w(T_2) \Rightarrow eia \quad a=1$$

$$\Rightarrow \{(159 - 1 \times 10^5) \times 10\} + 1 \quad b=2$$

$$\Rightarrow 591 \% 13 = 16 \quad c=3$$

= (spurious
bit) $\rightarrow d=4$

$$h_w(T_1) \Rightarrow iah$$

$$\Rightarrow \{(591 - 5 \times 10^5) \times 10\} + 8 \quad e=5$$

$$\Rightarrow 1928 \% 13 = 8 \quad f=6$$

$$h_w(T_4) \Rightarrow ahe$$

$$\Rightarrow \{(928 - 9 \times 10^5) \times 10\} + 5 \quad g=7$$

$$\Rightarrow 185 \% 13 = 3 \quad h=8$$

$$h_w(T_5) \Rightarrow hea$$

$$\Rightarrow \{(185 - 1 \times 10^5) \times 10\} + 1 \quad i=9$$

$$\Rightarrow 852 \% 13 = 6 \quad [match found]$$

$$h_w(P) = h_w(T_5) \quad [pattern found]$$

$\epsilon = 51.2 \times 10^{-6}$

ECE-2018

$$13) 415 \quad (32)$$

$$\begin{array}{r} 39 \\ 25 \\ 13 \\ \hline 12 \end{array}$$

$$13) 851 \quad (65)$$

$$\begin{array}{r} 78 \\ 72 \\ \hline \end{array}$$

$$\begin{array}{r} 65 \\ 6 \\ \hline \end{array}$$

628

(pridivide)

Divide by 3

$$13) 159 \quad (12)$$

$$\begin{array}{r} 17 \\ 29 \\ 26 \\ \hline 3 \end{array}$$

$$13) 591 \quad (45)$$

$$\begin{array}{r} 52 \\ 72 \\ 65 \\ \hline 6 \end{array}$$

add of 30 : pridivide.

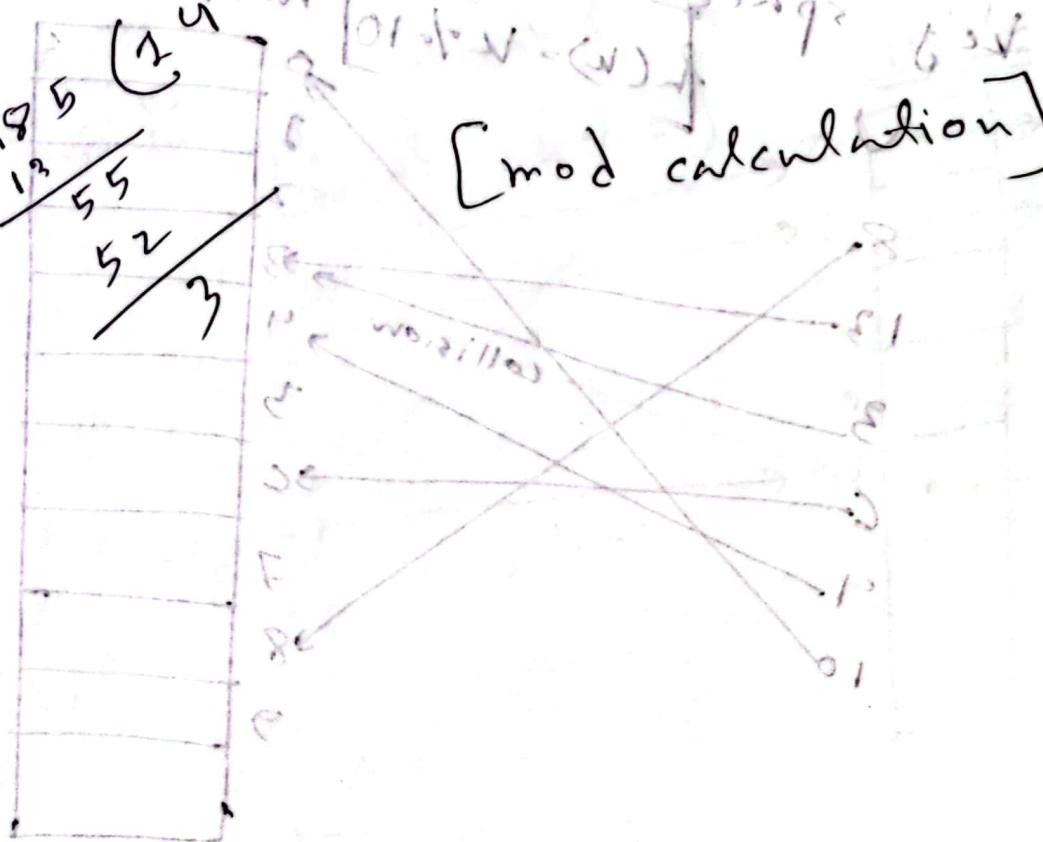
$$13) 918 \quad (10)$$

$$\begin{array}{r} 280 \\ 28 \\ 8 \\ \hline 0 \end{array}$$

[mod calculation]

$$13) 185 \quad (15)$$

$$\begin{array}{r} 12 \\ 55 \\ 52 \\ 3 \\ \hline \end{array}$$

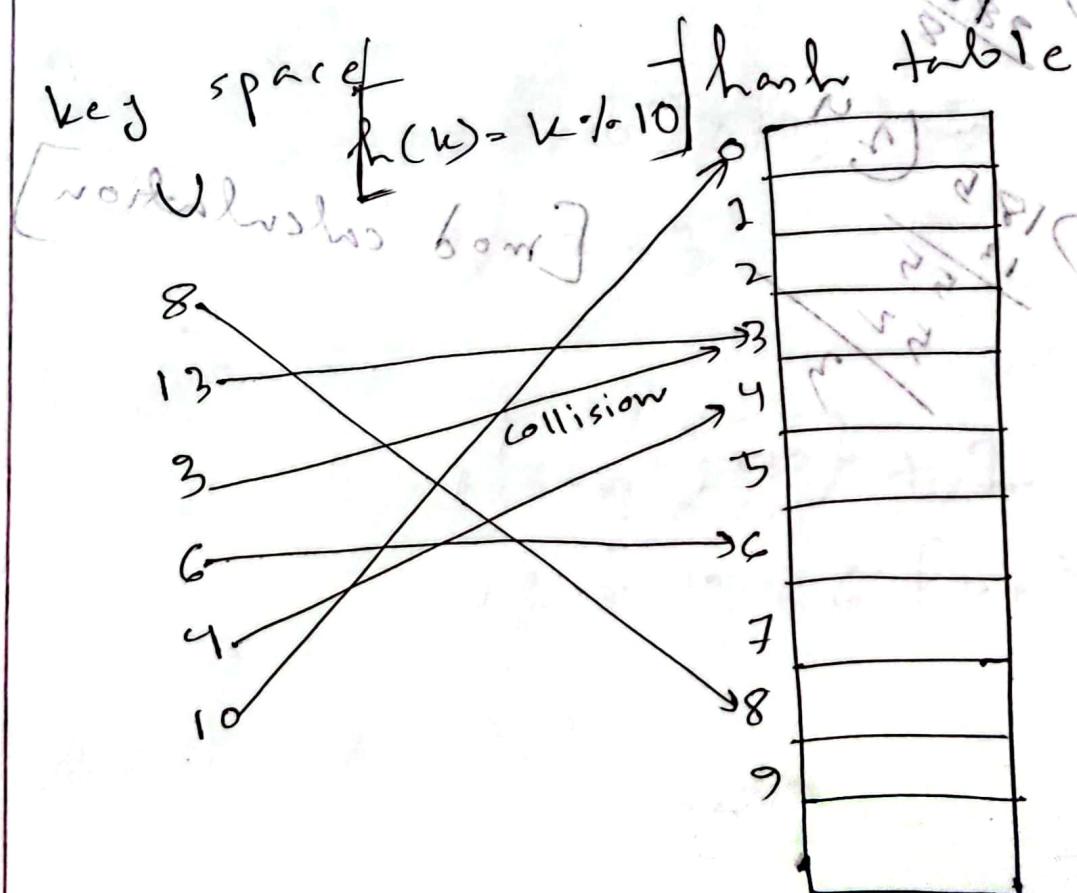


Hash tables

key → index

$$h(k) = k \mod m \quad [\text{ideal hashing}]$$

hashing: mapping from V to the slots of a hash table $T[0 \dots m-1]$



Avoiding collision: ~~using separate~~

1. Open ~~addressing~~ hashing \Rightarrow chaining \Rightarrow

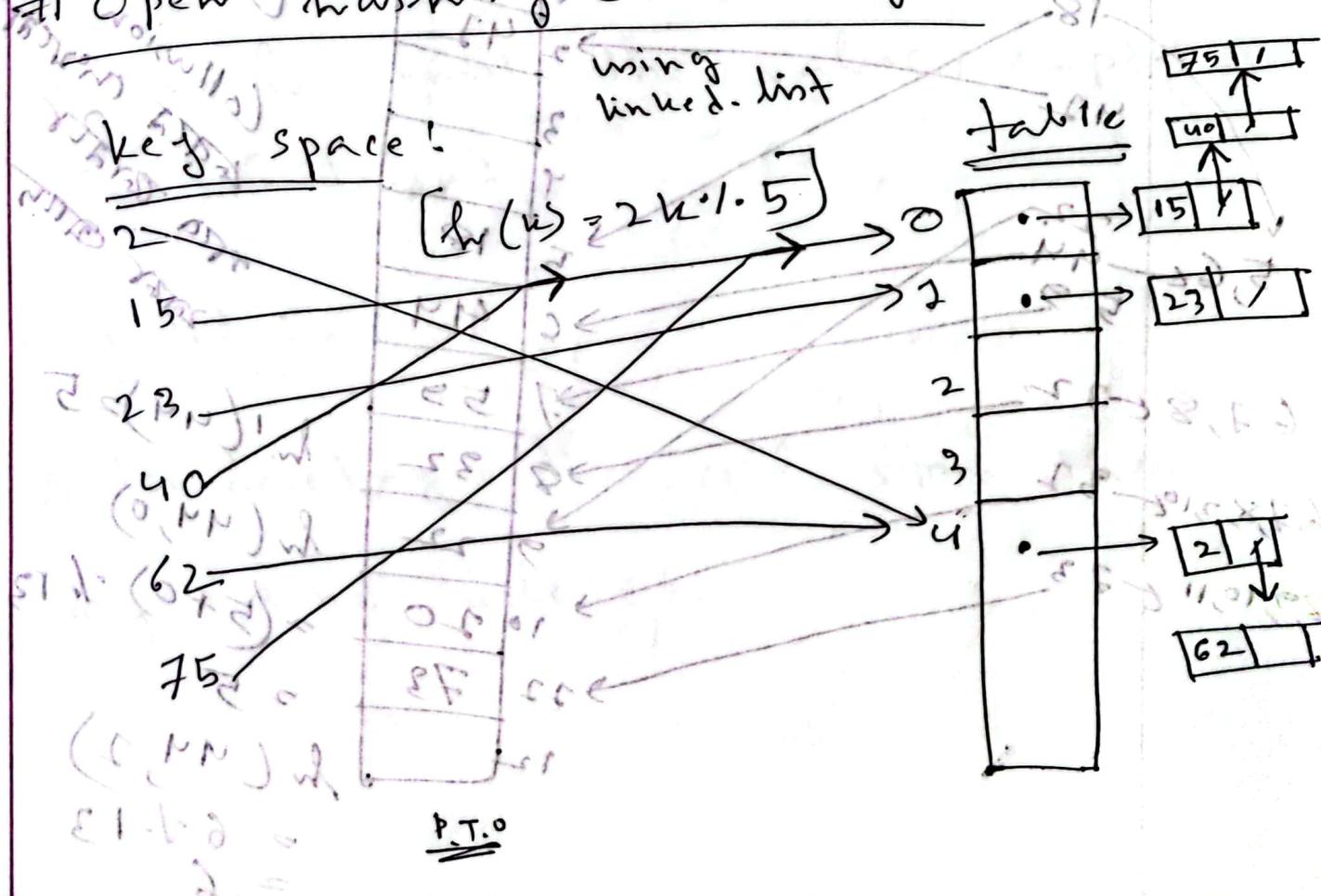
2. closed - hashing / open addressing

\rightarrow linear probing

3.1 (i) $(i \in \{0, 1\})$: \rightarrow Quadratic probing ①

\rightarrow double hashing

1) Open - hashing (chaining)



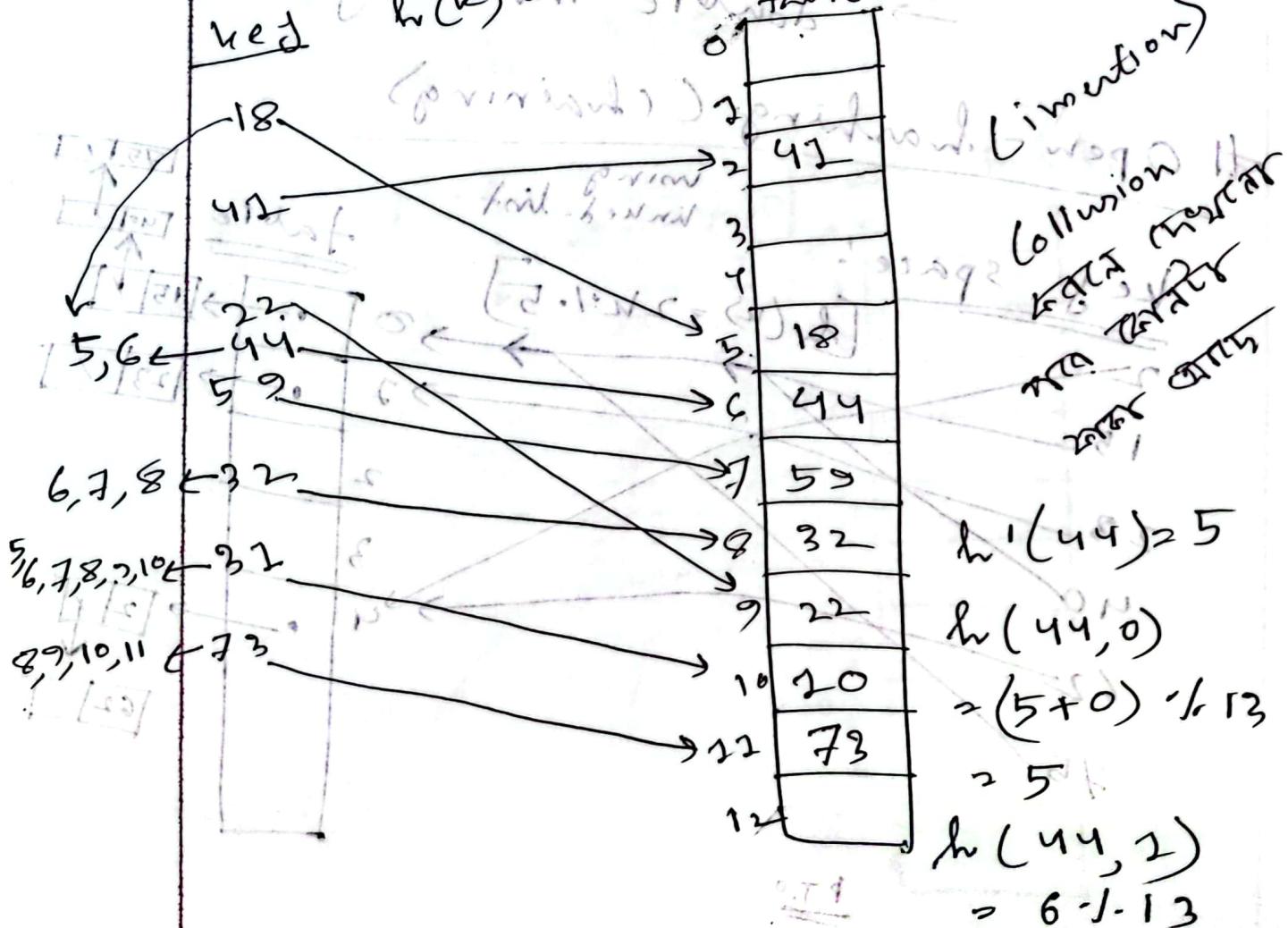
- * storing all elements when their dimensions are guaranteed.
- * waste of space

~~Guaranteed storage for initial blocks~~

~~# closed hashing~~

~~① Linear probing~~

$$h(k) = k \cdot 13$$



$$h^{-1}(44) = 5$$

$$h(44, 0)$$

$$= (5 + 0) \cdot 1 / 13$$

$$= 5$$

$$h(44, 1)$$

$$= 6 \cdot 1 / 13$$

$$= 6$$

$$h'(73) = 73 \bmod 13$$

→ 8 mod 13 = 8

$$h(73, 0) = (8+0) \bmod 13$$
$$= 8$$

$$h(73, 1) = 9 \bmod 13$$

$$h(73, 2) = 10 \bmod 13 = 10$$

$$h(73, 3) = 11 \bmod 13 = 11$$

$$h(73, 4) = 12 \bmod 13 = 12$$

→ search until the value / empty cell is found.

Delete: Prior to adding

After deleting the cell store their

tag name deleted.

~~start~~ ~~last~~ ~~catch~~

→ Primary clustering.

refer ~~start~~ ~~last~~ ~~last~~ ~~start~~ ~~last~~ ~~(0, EF)~~ ~~id~~

Quadratic probing

$$h(w)$$

$$h(w) = \text{start} + (w - 1) \cdot \text{size} \mod m$$

$$(h_1(w) + i) \mod m$$

if h_1, h_2, h_3, \dots all filled then choose
 $0, 1, 2, \dots$ if not available

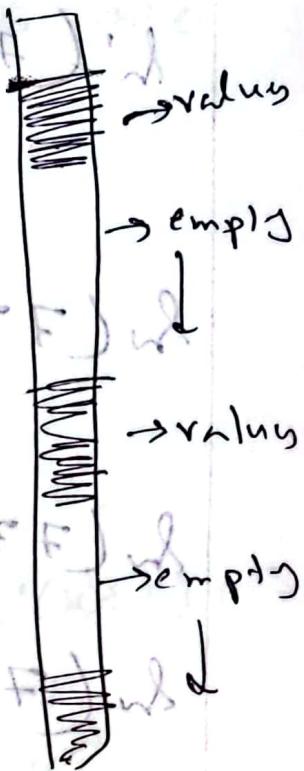
problem

→ secondary clustering:

note this will probably happen

666666 arrangement

P.T.O.



Double hashing

$$h(w) = (h'(w) + ih''(w)) \bmod m$$

hashing of word or string of length i

blowing up of word or string of length i

to triangles of width i and height i

Optimal choice of h' and h'' is constant

So $h''(x) = h''(y)$ implies that width

of x and y is same

length of string x is $\log n$

length of string y is $\log n$

so width of string x is $\log n$

so width of string y is $\log n$

so width of string x is $\log n$

so width of string y is $\log n$

so width of string x is $\log n$

so width of string y is $\log n$

so width of string x is $\log n$

so width of string y is $\log n$

so width of string x is $\log n$

so width of string y is $\log n$

so width of string x is $\log n$

so width of string y is $\log n$

0.7

NP Completeness

P (Polynomial) problems:

P problems refer to problems where an algorithm would take a polynomial amount of time to solve, where Big-O is a polynomial ($O(1), O(n), O(n^2)$, etc.). These are problems that would be considered "easy" to solve and thus do not generally have immense run times.

P.T.O

NP (Non-deterministic polynomial)

problems:

In terms of solving a NP problem, the run-times would not be polynomial if it would be exponential ($2^n, n^n, n!$ etc.). A class of problems can be solved in polynomial time and checking given a specific solution and the solution would have been given a polynomial run-time. Therefore, NP class problem does not have a polynomial run-time to solve, but having a polynomial run-time to verify a solution. In order to solve graphs we use a ~~greedy~~ algorithm.

Reduction:

Suppose, we have two problems, A & B and we know that problem B is NP class problem, then if problem A can be reduced, or converted to problem B, then we can say that A is also NP class problem. (A is reducible to B)

NP-hard problem

A problem is classified as NP-Hard when an algorithm for solving it can be translated to solve any NP problem.

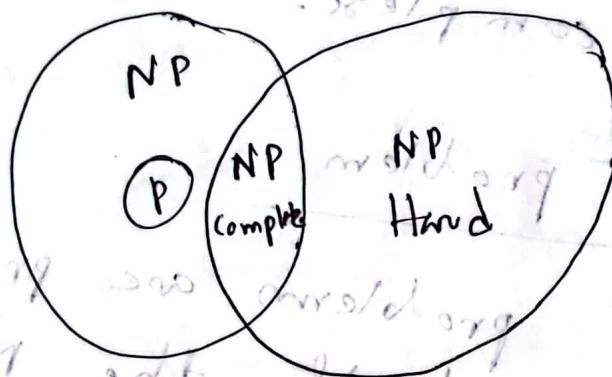
then we can say, this problem is at least as hard as any NP problem but it could be much harder or more complex.

NP-complete problem

NP-complete problems are problems that live in both the NP and NP-hard classes. This means that NP-hard problem can be solved in polynomial time, and any NP problem can be reduced to this problem in polynomial time.

$$\text{A class} \rightarrow \text{(say)} q_n = q$$

* Alexander Barvinok



P is the subset of NP. i.e.,
P is the subset of NP comple. Now
we prove that if a deterministic
algorithm converted into polynomial time deterministic
algorithm.

$$\text{P} \subseteq \text{NP}$$

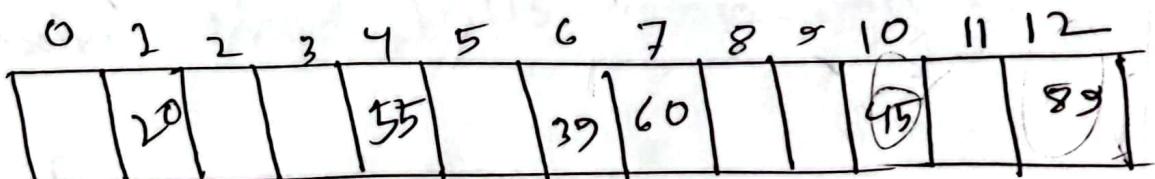
$$\text{P} = \text{NP} \quad (\text{Prove})$$

~~25~~
~~23~~
~~20~~

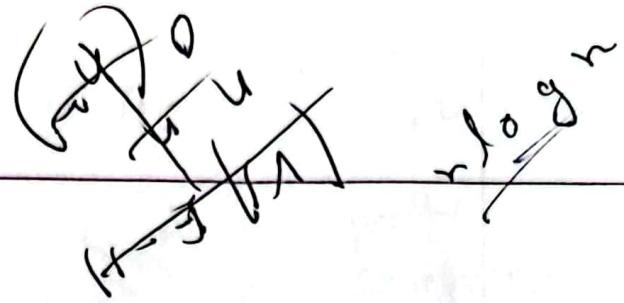
Year (for 1990) 0

(a) ~~Hebborg description~~ 2⁶³

κ	$h_1(w)$	$h_2(w)$	$h_3(w)$
Bottom of swim	0.15	0.15	0.15
89	0.12	0.12	(0, 12) w
55	1	3	(0, 2) x (2, 4) x
	ATTA	ATTA	ATTA



~~Spring-22~~



$$T_1 = 237395$$

$$P = 739 \approx 11$$

$$m \cdot d = 13$$

$$d = 10$$

$$T_0 = 237$$

$$= 2 \times 10 + 3 \times 10 + 7 \times 10$$

$$\therefore 237 = 3$$

$$T_1 = \{(237 - 2 \times 10) \times 10\} + 3$$

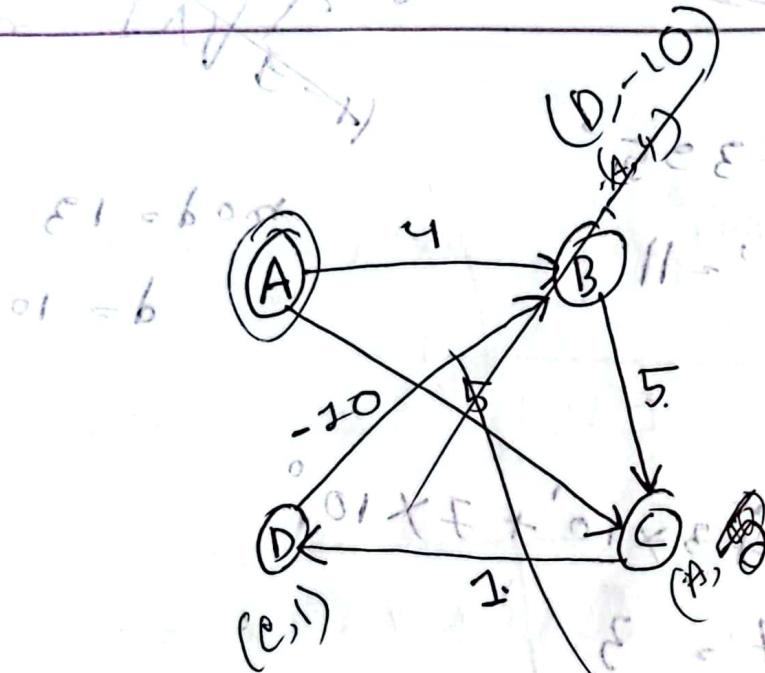
$$\text{why } 373 = 9 \rightarrow P - 1 \rightarrow$$

~~$$T_2 = \{(373 - 3 \times 10) \times 10\} + 9$$~~

$$\therefore 739 = 11 \quad [\text{match found}]$$

~~$$T_3 = \{(739 - 7 \times 10) \times 10\} + 5$$~~

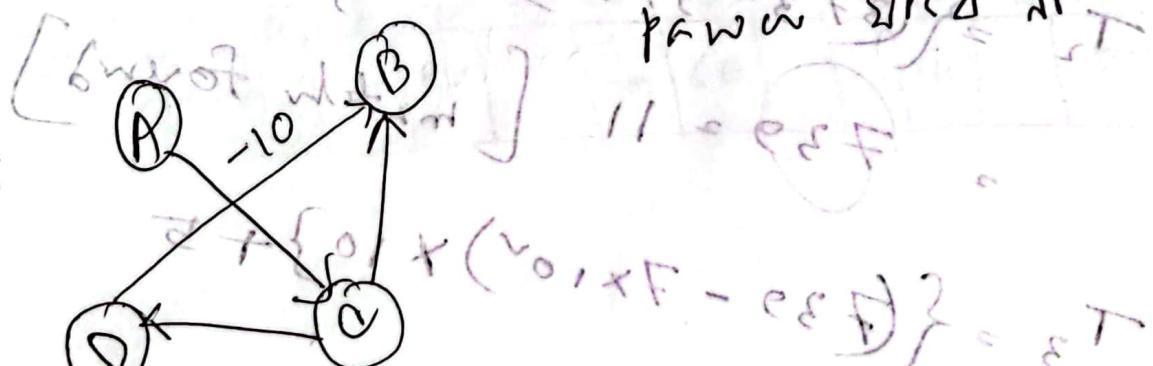
$$\therefore 395 = 5$$



$$(5 + 1 - 10) + (5 + 1 - 10) = 8 \text{ J.T}$$

$$\Rightarrow -4 - 9 = -8 \text{ J.T}$$

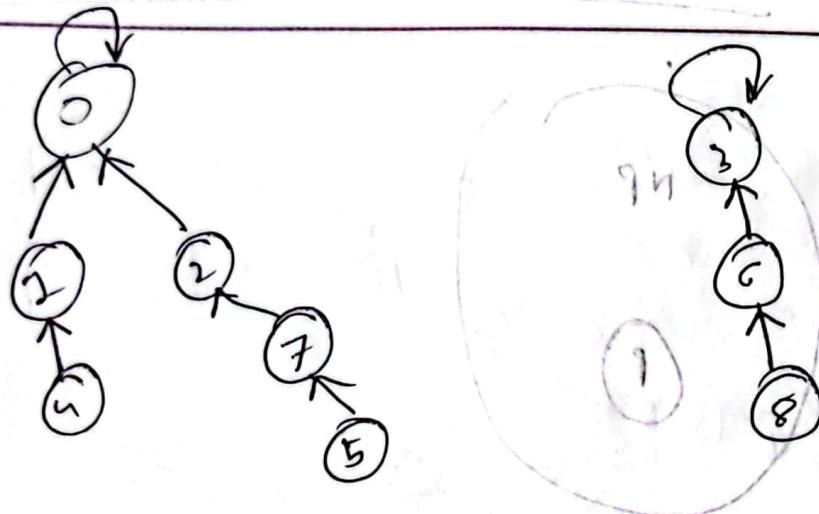
(Shortest path)



$$P = 8 \text{ J.T}$$

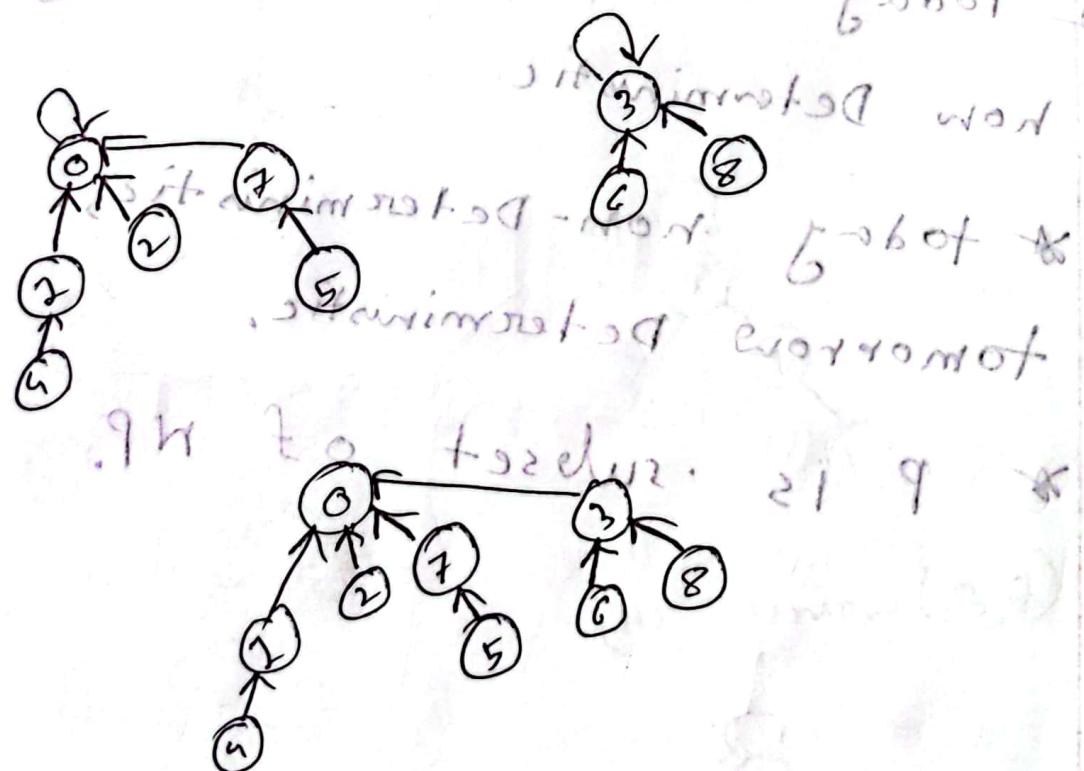
step 4: 9H 3: both - 9H

union by
range



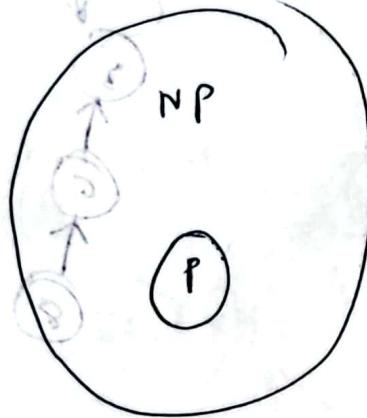
path
compression

now union(6,7,8)

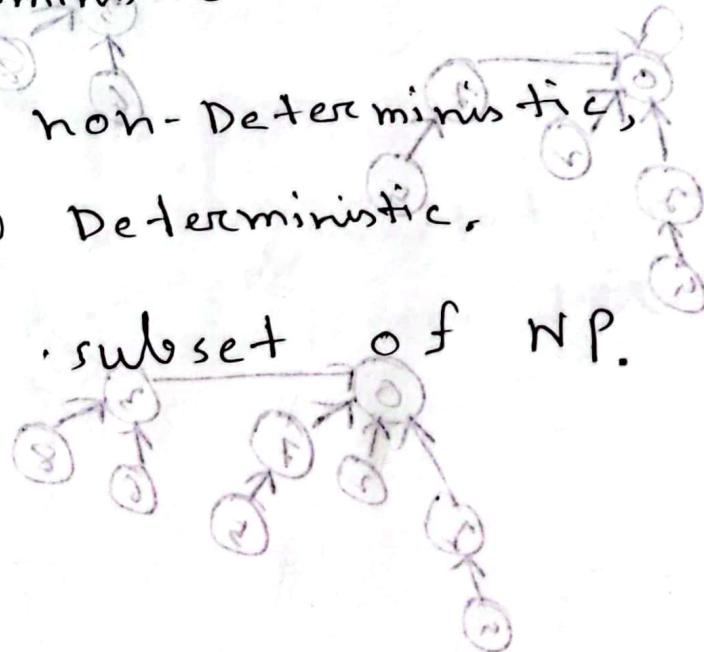


NP-Hard & NP complete

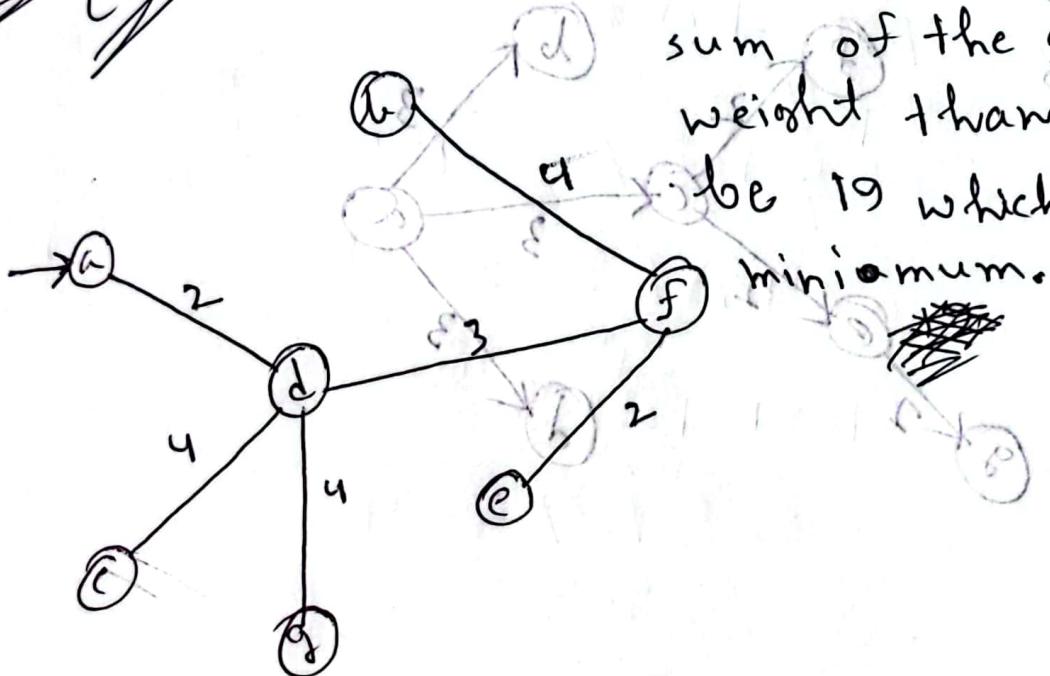
#



- * Today Deterministic, ~~yesterday~~ ~~ago~~ it was non Deterministic
- * today non-Deterministic, tomorrow Deterministic.
- * P is subset of NP.



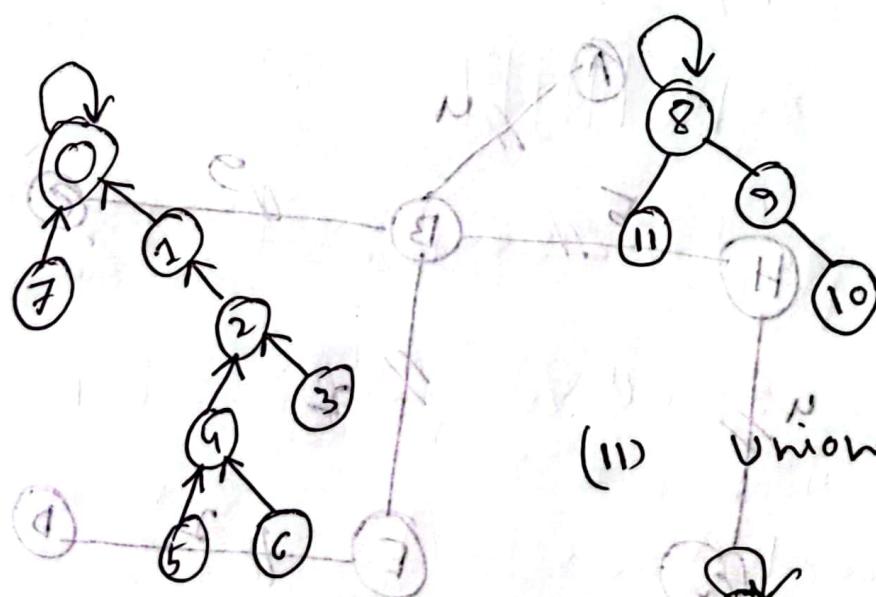
~~2(a) (ii)~~



if we check the sum of the all weight than it will be 19 which is minimum.

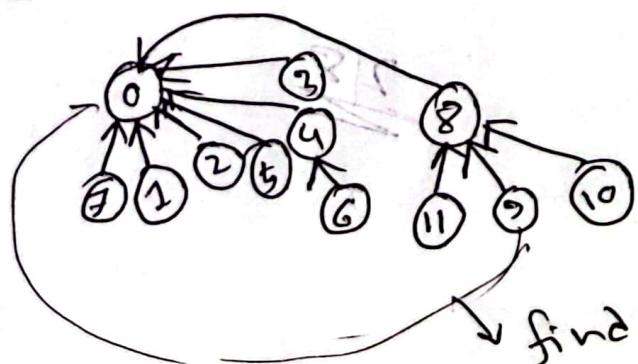
~~2(b)~~

(i)

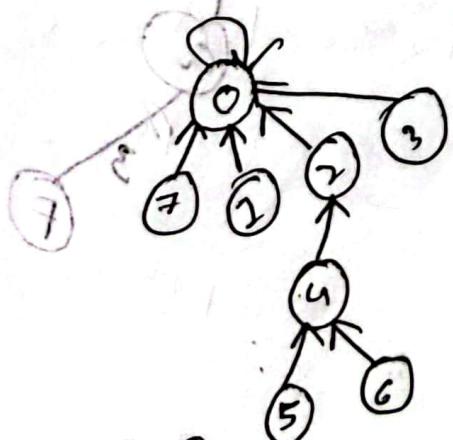


(ii) Union(3,7)

(iii) Union(5,10)



findset(9)=0
(iv)

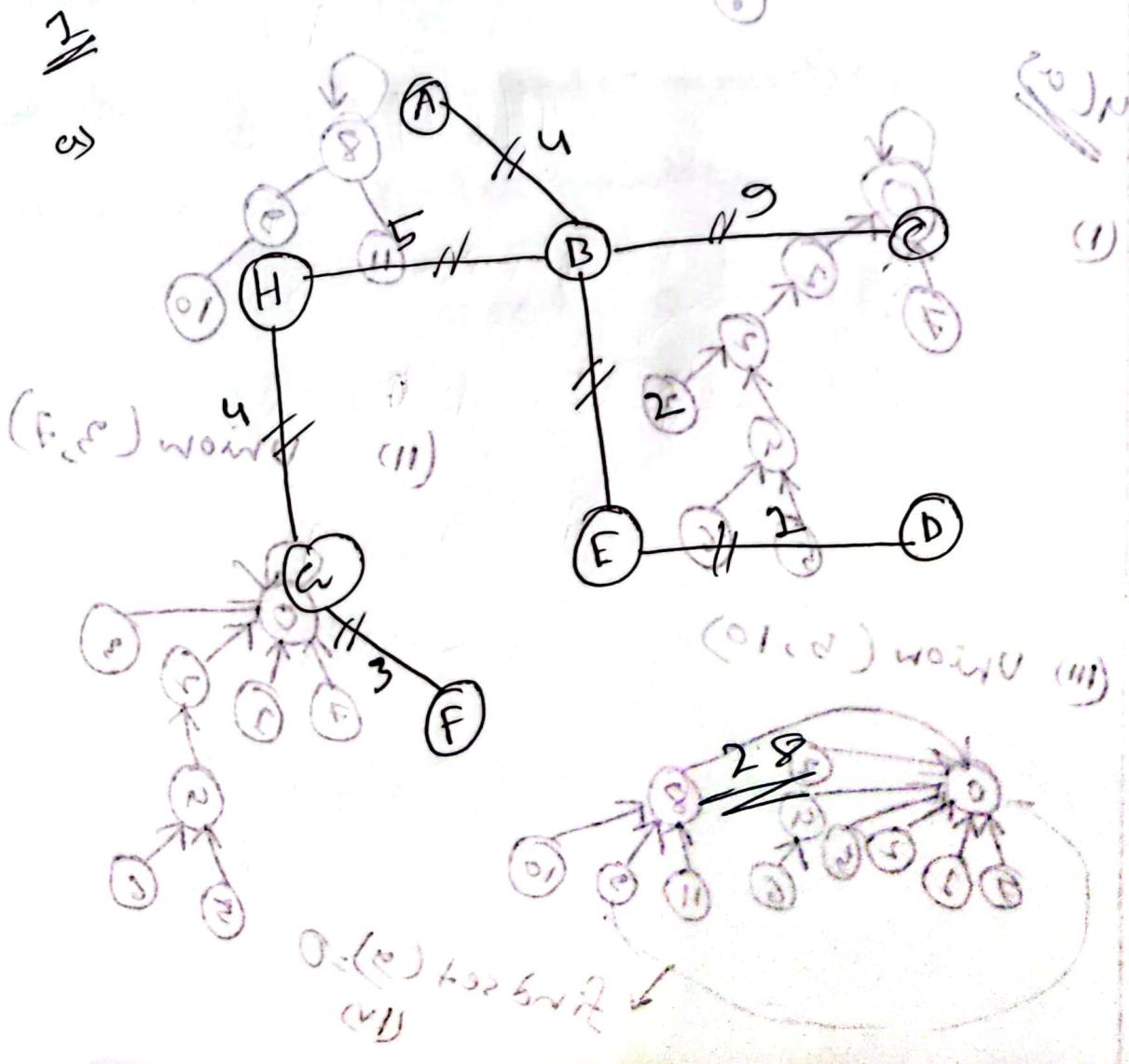
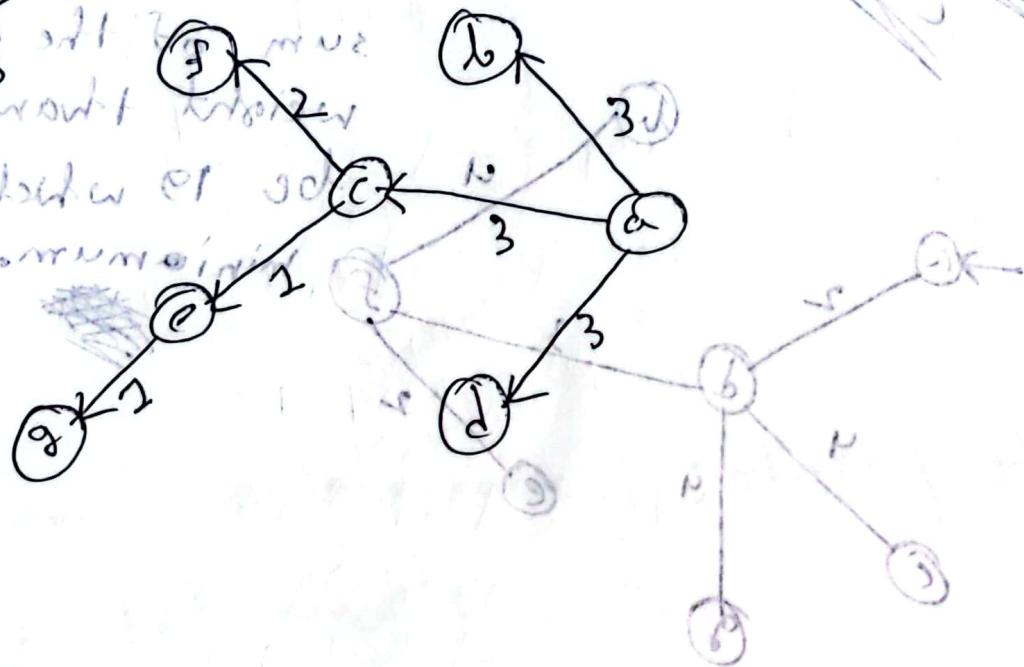


~~GP-23~~

~~Ex-ma?~~

sort 3 boards out of 7
also add 3 more
~~(as)~~ 11 forward passes

so whether it's ok
or not remains to be seen



WATSON

WATSON APPROPRIATE

P = T, G = C, A = G

P = 6

E - D - 2 ✓

for base

B - E - 2 white

G - F - 3 ✓

H - G - 4 ✓

A - B - 4 ✓

H - B - 5 ✓

F - E - 8 ✗

B - C - 7 ✓

H - A - 12 ✗

*

WATSON

APPROPRIATE

SWATSON

$P = \frac{P_x}{P_{\text{total}}} \times (1 - P_{\text{error}})$

~~SWATSON~~

~~SWATSON~~

~~SWATSON~~

~~SWATSON~~

~~SWATSON~~

~~A = 2, C = 2, G = 3, T = 4~~

~~$P = \frac{P_x}{P_{\text{total}}} \times (1 - P_{\text{error}})$~~

~~A G U C T A G U C G A G U C T A G~~

~~[writing] G A T C = F + G + C + T = 4~~

~~total~~

$P = A G U C T A G$

~~A G U C T A G~~

~~$P = (2 \times 4^5) + (3 \times 4^4) + (2 \times 4^3) + (4 \times 4^2) + (1 \times 4) + (3 \times 4^0)$~~

~~$= 1991 \cdot 1 \cdot 7 = 3882 N$~~

A G C T A C G G A H C T A H

A = 1, L = 2, W = 3, T = 4

A H C T A H

$t_0 = A H C T A H$

$$= 1991 \cdot 7 \cdot 3 \quad [\text{match found}]$$

d = 4

mod = 7

$t_1 = H C T A H C$

$$= \left\{ \left(1991 - \frac{1}{1024} \times 4^5 \right) \times 4 \right\} + 2$$

$$= \cancel{\left(1991 - \frac{1}{1024} \times 4^5 \right) \times 4} + 2 \quad \begin{matrix} \checkmark & 8 - 7 - 7 \\ \checkmark & 5 - 3 - 8 \\ \cancel{+} & 1 - A - 4 \end{matrix}$$

$t_2 = C T A H C H$

$$= \cancel{\left(1991 - \frac{1}{1024} \times 4^5 \right) \times 4} + 3$$

$$\text{WAT} \rightarrow \left\{ \left(3870 - 3 \times 4^5 \right) \times 4 \right\} + 3$$

$$= 31950 \cdot 7 = \cancel{3} \quad [\text{spurious hit}]$$

$t_3 = \frac{4}{7} A H C H A$

$$(P+1) = \left\{ \left(3195 - 2 \times 4^5 \right) \times 4 \right\} + 2$$

$$= 4589 \cdot 7 = 4$$

$$t_4 = AGC(CGA(GAAT) \times 4) + 3 \cdot 1.7$$

$$t_4 = \{4^5 \cdot 89 - 1 \times 4^5\} \times 4 + 3 \cdot 1.7$$

(starts with T)

$$= 1975 \cdot 1.7 = 2$$

$$t_5 = (AGCT(AAGT) \times 4) + 2$$

$$\rightarrow \{(1975 - 1 \times 4^5) \times 4\} + 2$$

(starts with A)

$$= 3806 \cdot 1.7 = 5$$

$$t_6 = (AGCT(AAGT) \times 4) + 4 \cdot 1.7$$

$$\rightarrow \{(3806 - 3 \times 4^5) \times 4\} + 4 \cdot 1.7$$

(starts with T)

$$= 20940 \cdot 1.7 = 0$$

starts with A

$$t_7 = (AGCT(AAGT) \times 4) + 2 \cdot 1.7$$

$$\rightarrow \{(20940 - 2 \times 4^5) \times 4\} + 2 \cdot 1.7$$

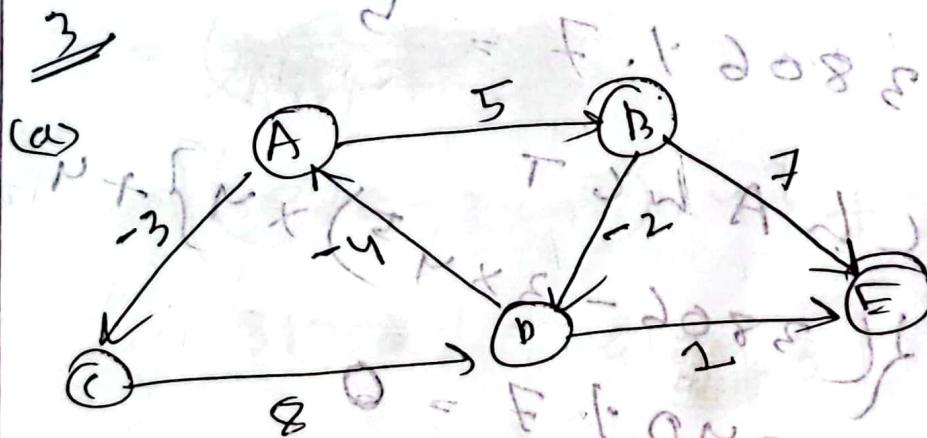
(starts with C)

$$= 3569 \cdot 1.7 = 6$$

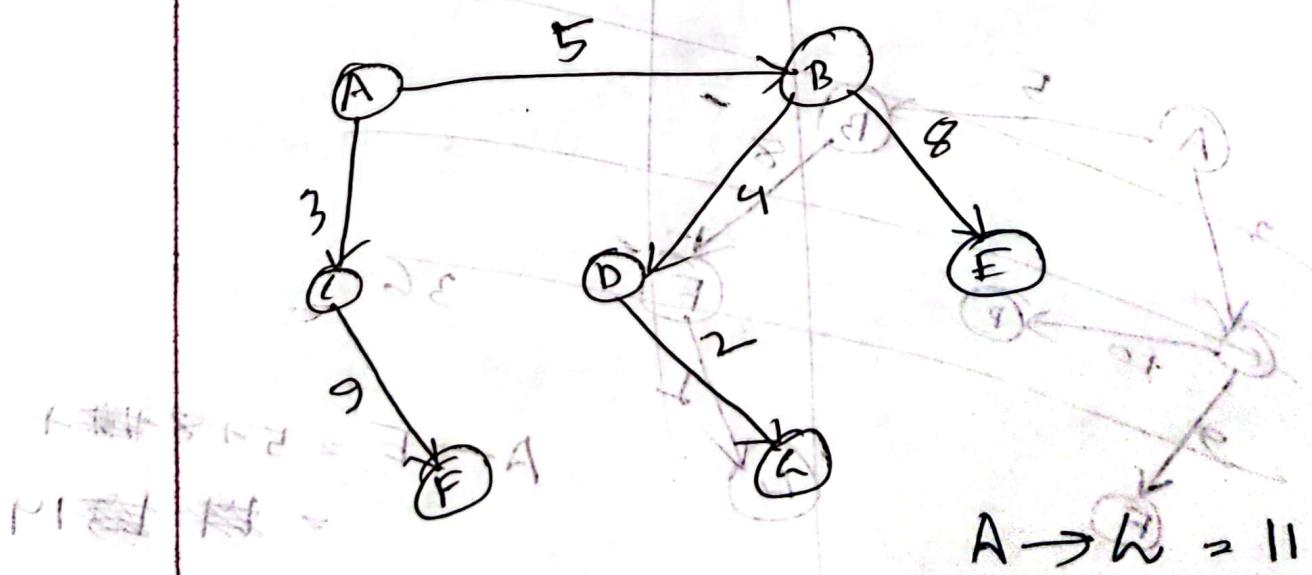
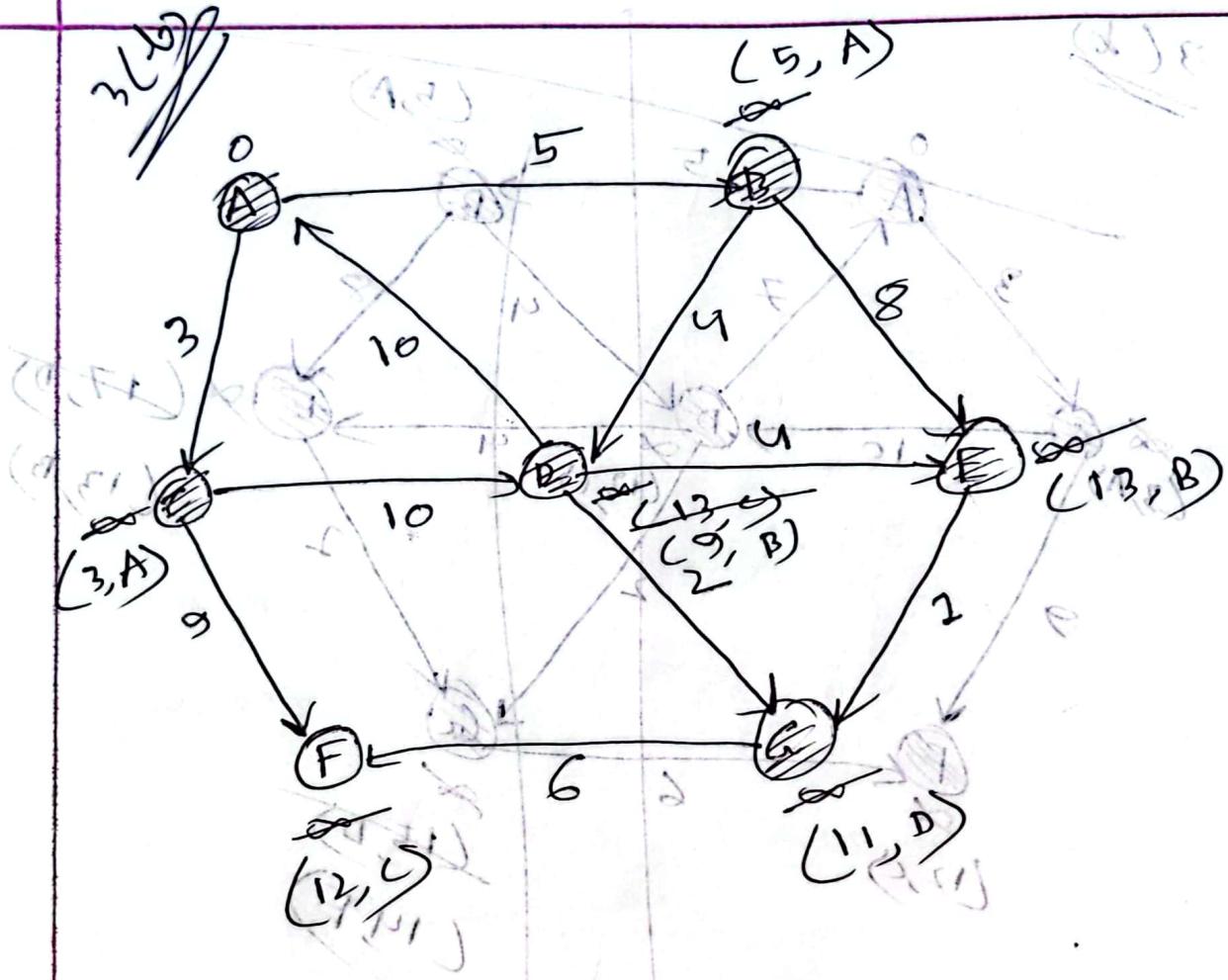
(starts with G)

$$\begin{aligned}
 t_8 &= A \oplus (T \oplus A) \oplus A = A \\
 &= \{(3569 - 3 \times 4^5) \times 4\} + 3 = 7.7 \\
 &\Rightarrow 1991 \cdot 1.7 = 3 \quad [\text{match found}]
 \end{aligned}$$

number of occurrences = 2
 spurious hit = 2 - effective



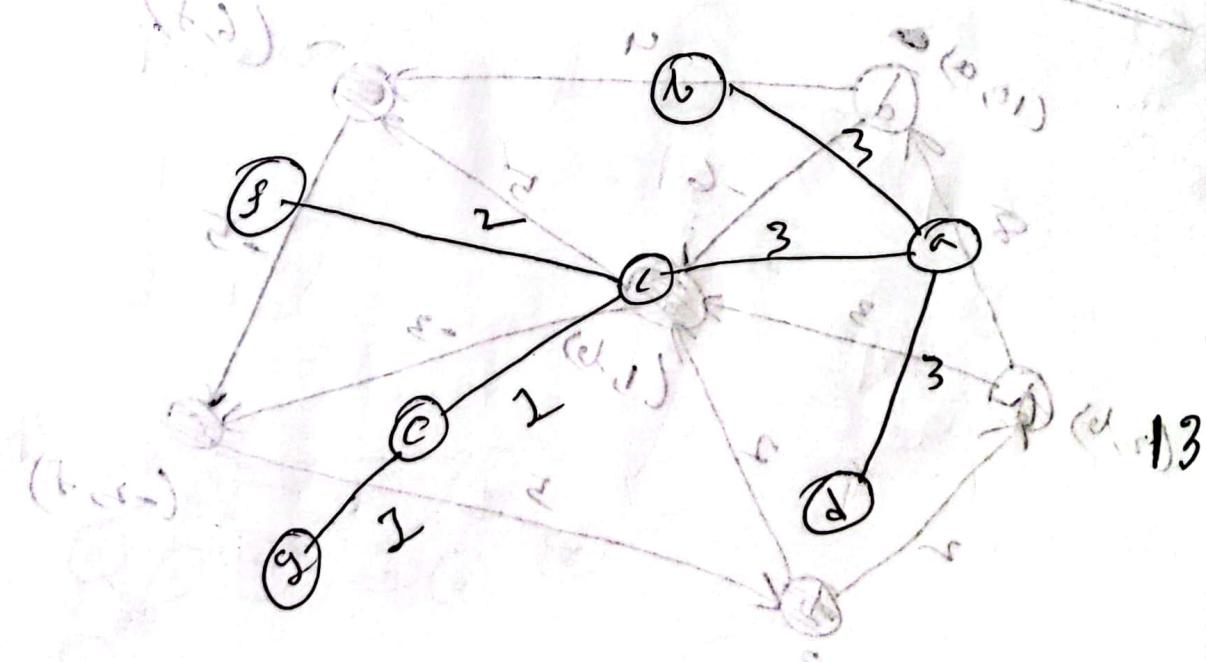
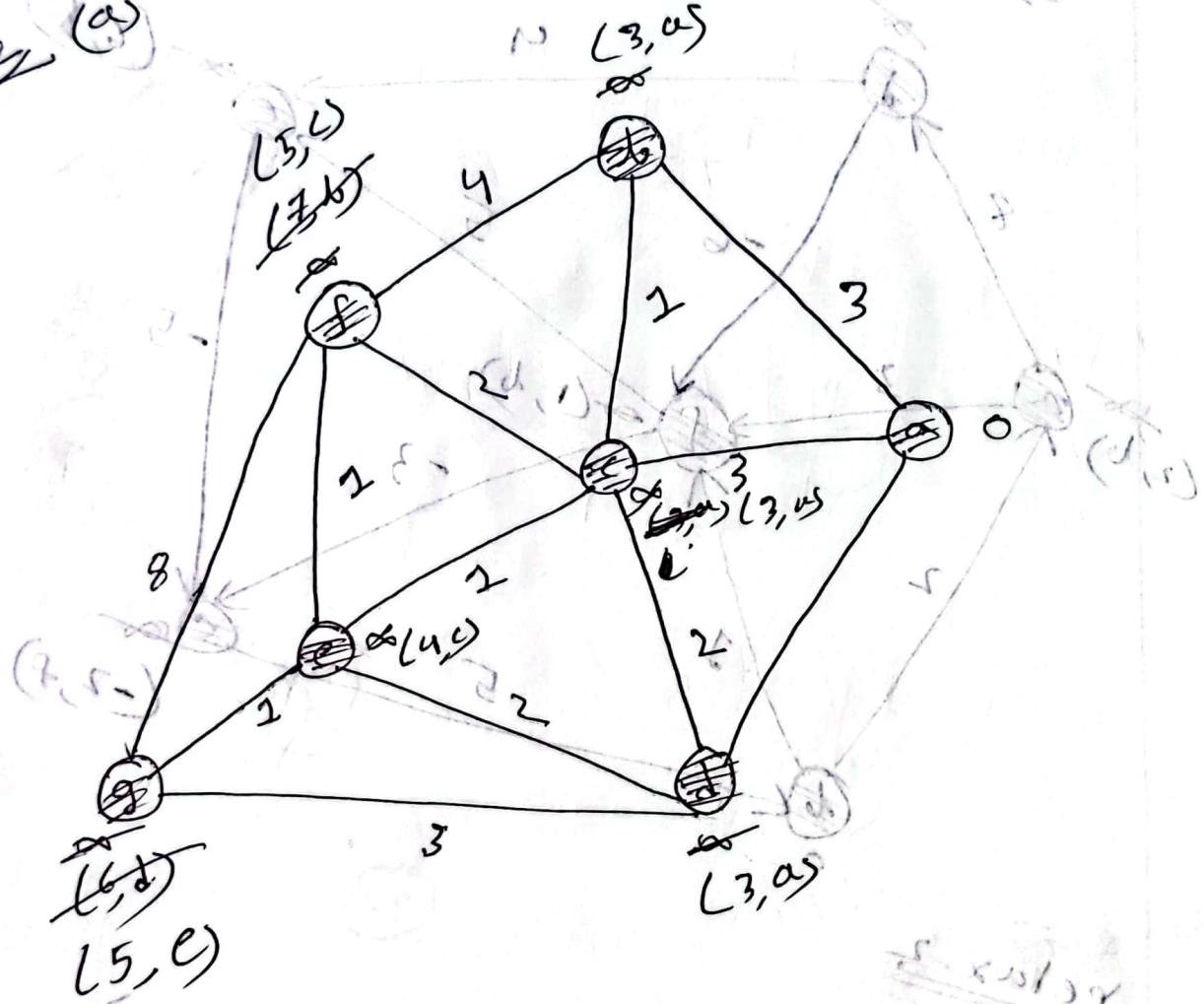
(a) In ABD there is a negative weight cycle.
 So, we can't find the shortest path vertex A to E.

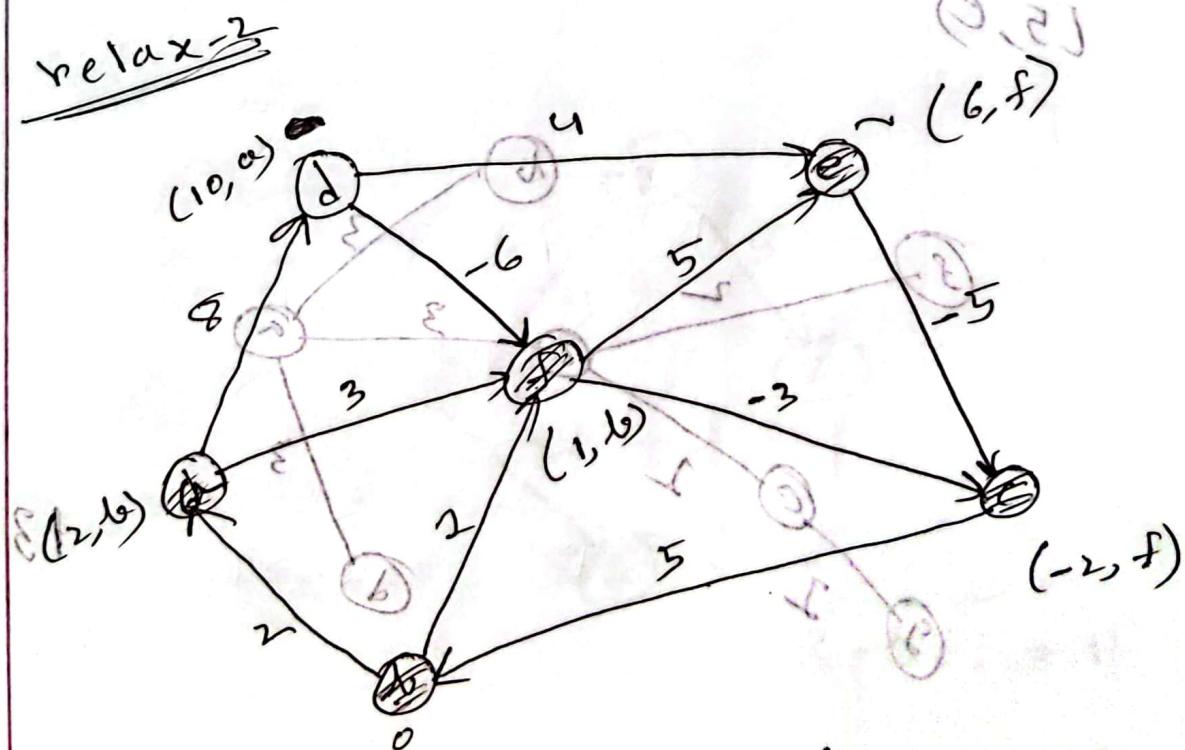
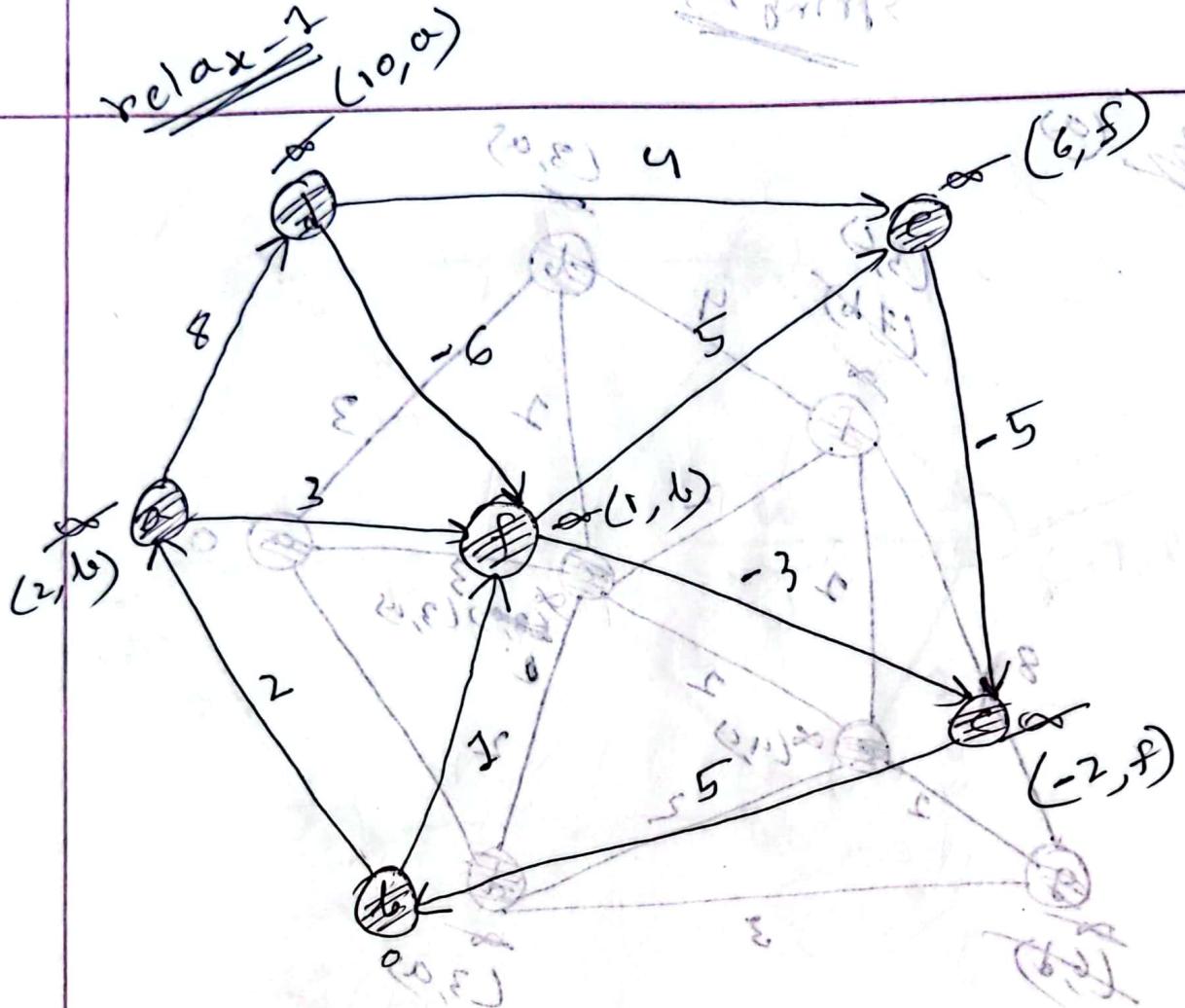


$$\text{sum} = 32$$

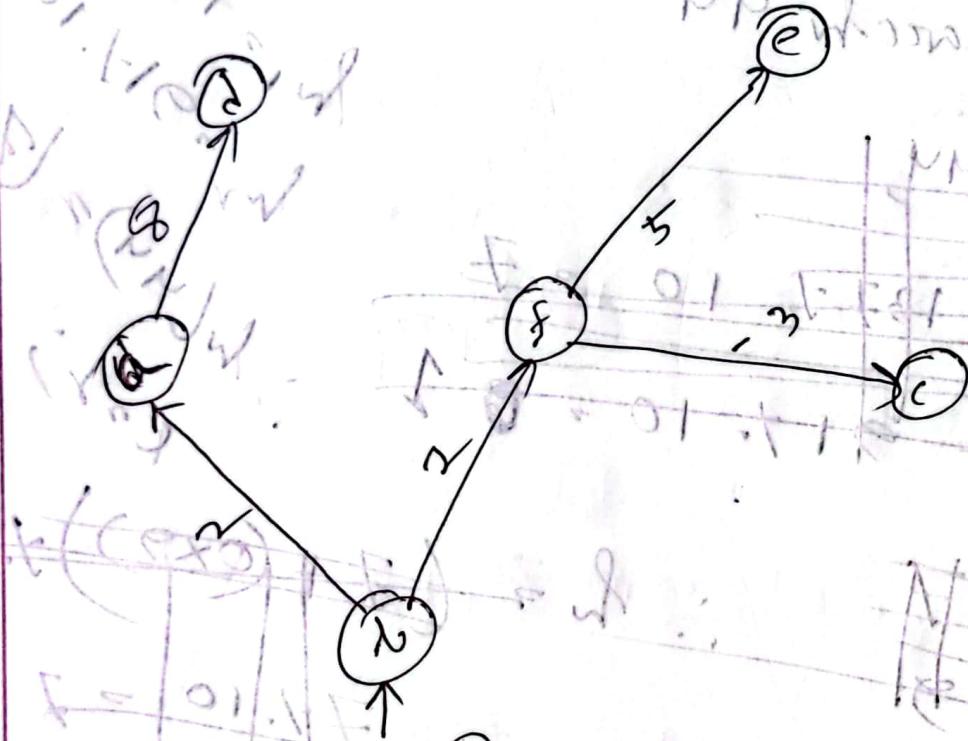
Spring 23

3Y (a)

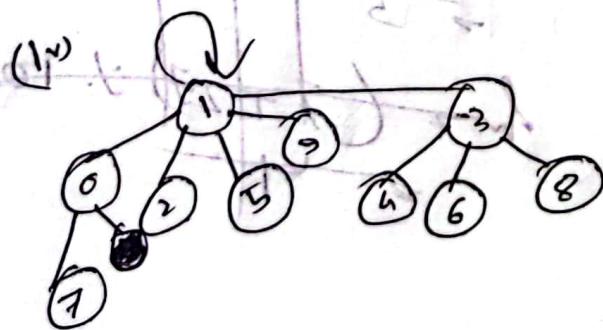
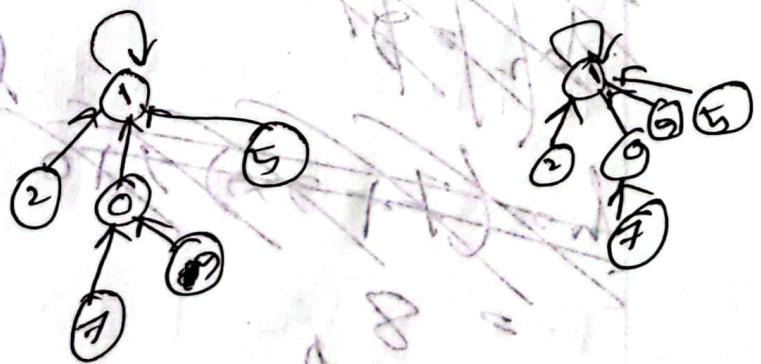




relax 3, 4, 5 will not change the value of weight.



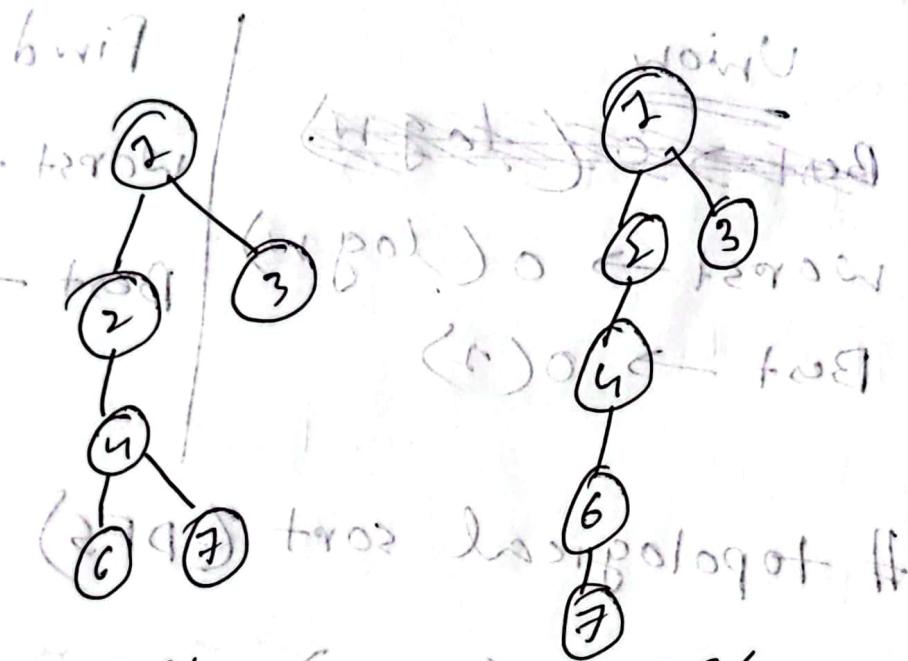
(1) find set(0)



BFS (queue) search

2246

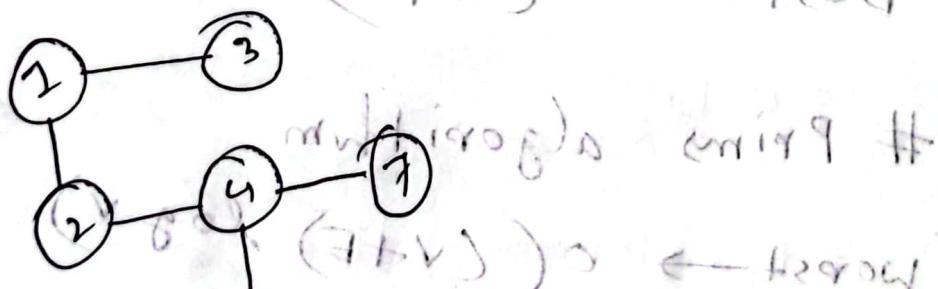
for Traversal #



BFS

$(\text{left}) \circ \leftarrow \text{root}$

DFS



$(\text{left}) \circ \leftarrow \text{root}$

$(\text{left}) \circ \leftarrow \text{root}$

$(\text{left}) \circ \leftarrow \text{root}$

$(\text{left}) \circ \leftarrow \text{root}$

$\text{smash} \leftarrow \text{bad}$

Time Complexity

Disjoint set

Union
~~Best $\rightarrow O(1)$~~

worst $\rightarrow O(\log w)$
Best $\rightarrow O(1)$

Find

worst $\rightarrow O(\log w)$
Best $\rightarrow O(1)$

topological sort (DFS)

worst $\rightarrow O(V+E)$
Best $\rightarrow O(V+E)$

Prims algorithm

worst $\rightarrow O((V+E) \log V)$

Best $\rightarrow O(E + V \log V)$

Kruskal algorithm

worst $\rightarrow O(E \log E)$ or $O(E \log V)$

Best \rightarrow same

Dijkstra algorithm

worst \rightarrow Bin heap $\rightarrow O((V+E) \log V)$

Fib heap $\rightarrow O(E + V \log V)$

Best $\rightarrow O(E + V \log V) \rightarrow$ Fib heap

Bellman Ford algorithm

worst $\rightarrow O(V \times E)$

Best $\rightarrow O(V \times E)$

Rabin Karp algorithm

worst $\rightarrow O((n+m) \times q)$

$n \rightarrow$ text length

$m \rightarrow$ pattern length

$q \rightarrow$ mod value

Best $\rightarrow O(n+m)$

P.T.O

All hash table methods are O(1) H

worst $\rightarrow O(n)$ search with \leftarrow loops

Best $\rightarrow O(1)$

BFS

worst $\rightarrow O(V+E)$ visit every node

Best $\rightarrow O(V)$ \leftarrow explore

DFS

worst $\rightarrow O(V+E)$ visit every node

Best $\rightarrow O(V+E)$ \leftarrow explore

PHP tutorial 114

order book exp

(m+n) \leftarrow total

279