L-1
23.7.23
26.7.23

Algorithm Analysis

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

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

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

P.T.O.

Algorithm Analize

(i) Correctness

(ii) Amount of work done (complexity)

(iii) Amount of space used (memory)

(iv) Simplicity, clarity

(v) Optimality

Running time:

(*) $y = m * x + b$

(*) $C = \frac{5}{9} * (\quad)$

(*) $Z = f(x) + \log(y)$

P.T.O.

Insertion Sort

1 2 3 4 5 6 7 8 — [Ascending Order]

```
Insertion Sort(A, n){
    for i = 2 to n {
        key = A[i];
        j = i-1;
        while (j>0 && A[j]>key) {
            A[j+1] = A[j];
            j = j-1;
        }
        A[j+1] = key;
    }
}
```

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

$T = t_1 + t_2 + t_3$

$f(n) = an + b \longrightarrow [Big \rightarrow \Omega]$

$f(n) = an^2 + bn + c \longrightarrow [Big \rightarrow O]$

* Asymptotic Performance:

→ Run time

→ Memory / storage requirement

→ Bandwidth / Power requirements / logic gates etc.

Worst case:

→ provide upper bound on running time.

→ An absolute guarantee of requirement

Average Case:

→ provides the expected running time.

→ Random inputs

→ Real life inputs.

Upper bound notation

A function $f(n)$ is $O(g(n))$ if there exists positive constants C and n_0 such that

$f(n) \leq C * g(n)$ for all $n \geq n_0$

$O()$ → "Big - O"

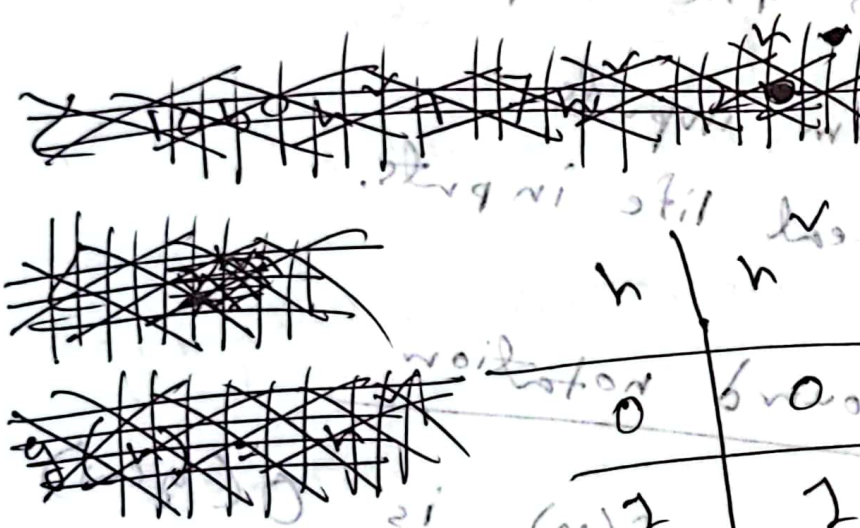
$f(n) = 5n^2 + 3n + 2 \leq 10n^2$

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

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

*

$f(n) = 1050\, n^2 + 7 \cdot 2^n$



n	n^2	2^n
0	0	1
1	1	2
2	4	4
3	9	8
4	16	16
5	25	32
6	36	64

$1050 \cdot 2^n + 7 \cdot 2^n$

$2^n (1057)$

$g(n) = 2^n$

$1057 \cdot g(n)$

$n_0 = 4$

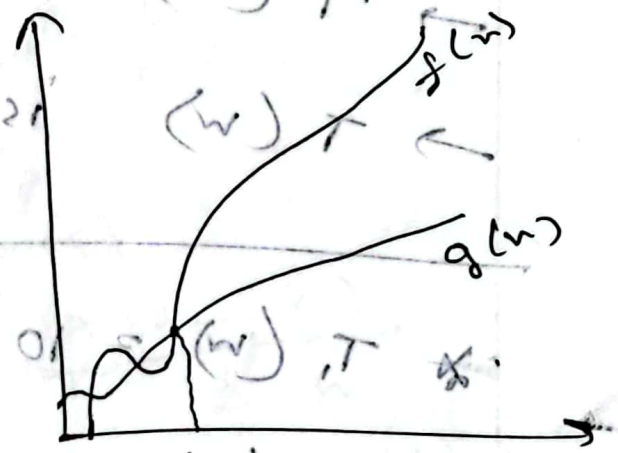
G.W
30.7.23

Lecture-3

DSA-2

* Lower bound:

if $c * g(n) \leq f(n)$ where $n \geq n_0$ and $c, n_0 > 0$, $\Omega \in \mathbb{R}$. then we say $g(n)$ is an asymtotic lower bound for $f(n)$

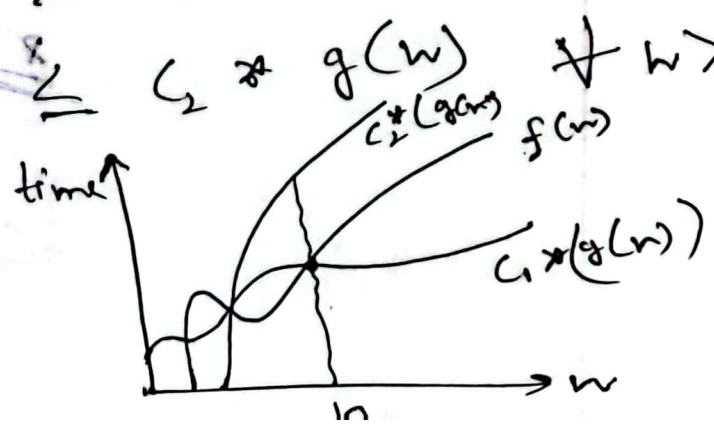


$\Omega(g(n))$ → Big Omega

* tight Bound: (Big theta)

$\theta(g(n))$, if for any positive constants c_1, c_2 & n_0

$c_1 * g(n) \leq f(n) \leq c_2 * g(n) \quad \forall n \geq n_0$



* $T(n) = 5n^3 + 4n + \log(n) + n^3$

→ $T(n)$ is $O(n^3)$

→ $T(n)$ is $\Omega(n^3)$

→ $T(n)$ is $\Theta(n^3)$

→ $T(n)$ is $o(n^4)$

---

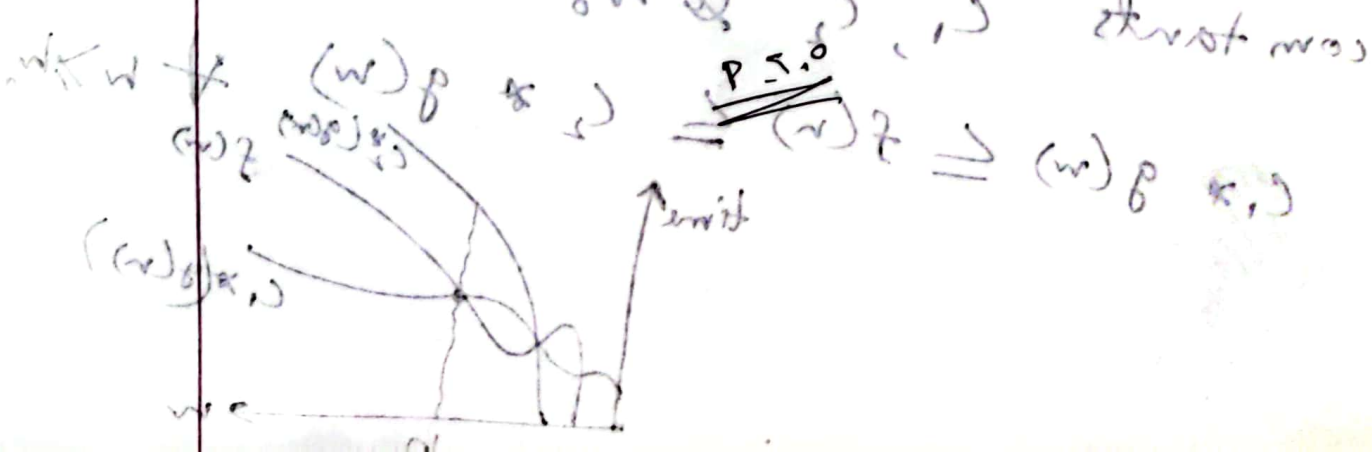
* $T_1(n) = 10^{10} n^2$

$T_2(n) = 10^{-8} n^3$

$T_1 = T_2$ where $n_0 = 10^{18}$

if $n \leq 10^{18}$, $T_1(n) \geq T_2(n)$

if $n \to \infty$, $T_2(n) \geq T_1(n)$



Exact cost Analysis :

1. for i, 1 to n ; → n+1
2. if arr[i] % 3 == 0 ; → n
3. print (arr[i]) ;

→ Best

→ Worst

→ Avg → $\frac{(1+2+\dots+n)}{n}$

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

→ $\frac{n}{2} \cdot \frac{n+1}{2}$

$O(n)$
$\Omega(n)$
$\Theta(n)$
} full run time

```
1 → for (i = n; i >= 0; i = i - 5) {
2 → if (A[i] < 100)
3 → Break;
4 → for (k = 1; k <= n; k * 2)
5 → print (A[k]);
```

line	Worst	Best
1	$\frac{n}{5} + 1$	1
2	$\frac{n}{5}$	1
3	0	1
4	$\frac{n}{5}(\log_2 n + 1)$	0
5	$\frac{n}{5} \log_2 n$	0
$O(n \log_2 n)$ $\Omega[1]$		

for(k = 1 ; k ≤ n ; k * 2) → $\log_2 n$

→ 1 for (i = n ; i >= 0 ; i = i - 3)

→ 2 if (A[i] < 0)

→ 3 break ;

→ 4 for (k = n ; k >= 1 ; $k = \frac{k}{4}$)

→ 5 print A[k]

	worst	best
1	$\frac{n}{3} + 1$	1
2	$\frac{n}{3}$	1
3	0	1
4	$\frac{n}{3} (\log_4 n + 1)$	0
5	$\frac{n}{3} \log_4 n$	0

$$\log_4 n = \frac{\log_2 n}{\log_2 4}$$

$$= \frac{1}{2} \log_2 n$$

$\#\ O(n \log_2 n)$

$\#\ \Omega(1)$ [n] A string

GW
3.10.23

* Exact cost Analysis

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

1 → n + 2

2 → $\frac{n(n+1)(2n+1)}{6} - 1$

$1^2 + 2^2 + 3^2 - - - n^2$

$\Rightarrow \frac{n(n+1)(2n+1)}{6} - 1$

Big O (n^3)

```
* c = 0;
1 → for (i = n/2 ; i <= n ; i++) {
2 → for (j = 2 ; j <= n ; j = j * 2) {
3 ⟶ c = c + k + n/2 ;
    }
}

4 → for (i = 0 ; i <= n ; i++) {
5 → for (j = 2 ; j <= n ; j = j + 2) {
6 → c = k + n/2 ;
    }
}
```

1 $\longrightarrow$ $\frac{n}{2} + 2$ $\longrightarrow$ $\frac{n}{2} + c$

2 $\longrightarrow$ $\log_2 n + 2$ $\longrightarrow$ $\log_2 n + c$

3 $\longrightarrow$ $\log_2 n$ $\longrightarrow$ $\log_2 n$

4 $\longrightarrow$ $m + 2$ $\longrightarrow$ $m + c$

5 $\longrightarrow$ $\frac{n}{2} = 2 + 1$ $\longrightarrow$ $\frac{n}{2} + c$

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

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

$= O(n \log n)$ if $\log n > m$

$O(nm)$ if $m > \log n$

P.T.O

* Derive the best & worst case running time equations & express in big O notations.

* *

* Prove that running time $T(n) = n^3 + 20n + 1$ is $O(n^3)$

→ By the Big-Oh definition, $T(n)$ is $O(n^3)$ if $T(n) \leq c \cdot n^3$ for some $n \geq n_0$.

Let us check this condition:

if $n^3 + 20n + 2 \leq c \cdot n^3$

$\Rightarrow 1 + \frac{20}{n^2} + \frac{1}{n^3} \leq c$

P.T.O

Therefore, the big-Oh notation holds for $n \geq n_0 = 1$ & $C \geq 22$

$= (1 + 20 + 1)$

$= 22$

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

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

$n^3 + 20n \geq C n^3$

$\Rightarrow n + \frac{20}{n} \geq C$

P.T.O

$f(n) \leq c\,g(n)$

$f(n) \leq C n^2$

$n = \sqrt{20} = 4.47$ when $C = 18.94$

$$\frac{d}{dn}\left(n + \frac{20}{n}\right) = 0$$

$$\Rightarrow 1 - \frac{20}{n^2} = 0$$

$$\Rightarrow \frac{20}{n^2} = 1$$

$$\Rightarrow n^2 = 20 \Rightarrow n = 4.47$$

$n \geq n_0 = 5$ && ($\leq$)

(i = 0; i <= n; i++) (n+2)

(i = 1; i <= n; i++) (n+1)

(i = 2; i <= n; i++) (n)

Divide & conquer

* Divide & conquer is a general algorithm design paradigm.

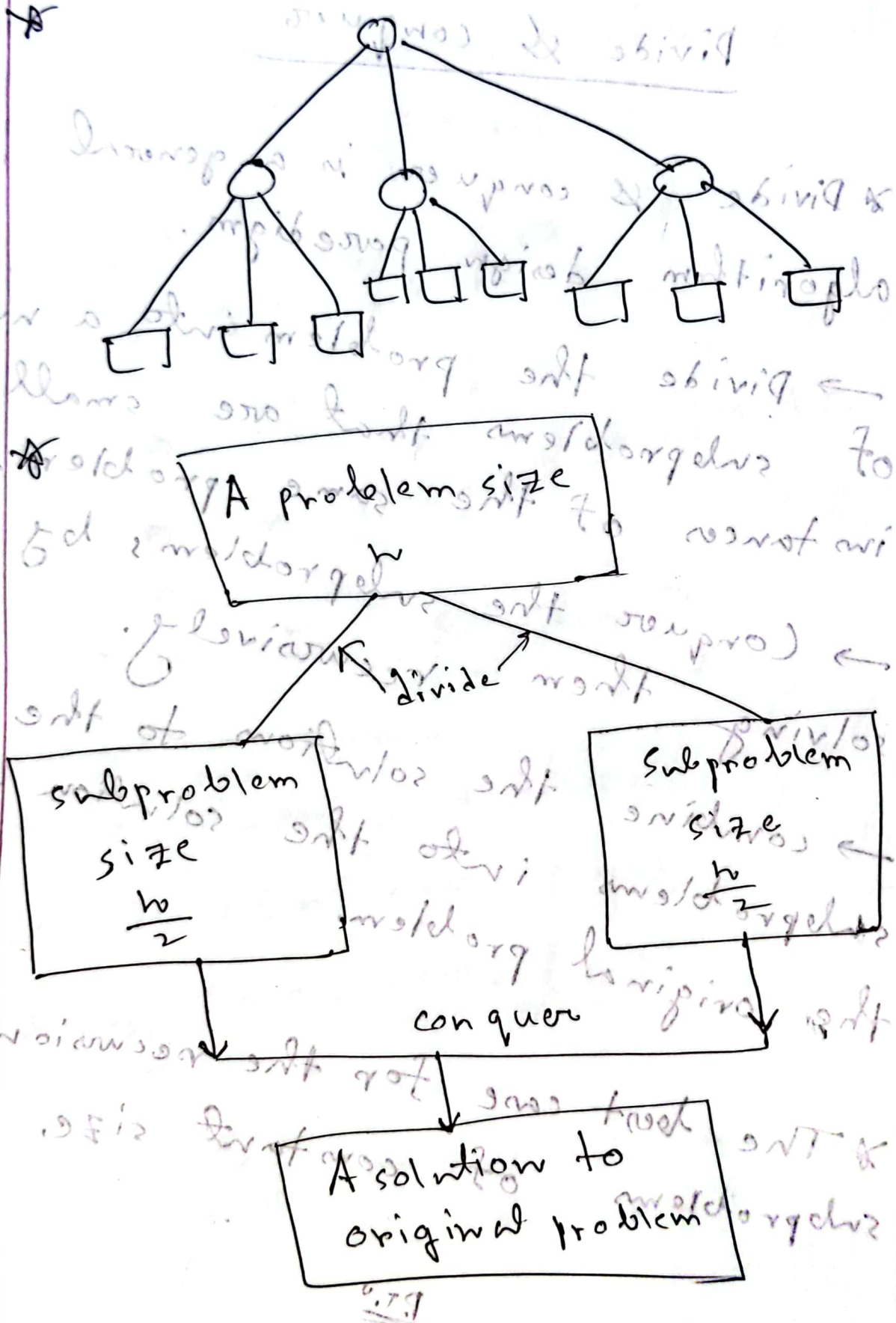
→ Divide the problem into a number of subproblems that are smaller instances of the same problem.

→ 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 are subproblems of constant size.

P.T.O


A problem size n
divide
subproblem size n/2
subproblem size n/2
conquer
A solution to original problem

Finding Maximum & Minimum

Input: An array A[1...n] to n numbers.

Output: The max & min value.

	1	2	3	4	5	6	7	8	9	10	11
A =	13	-12	-25	20	-3	16	18	12	-22	15	7

List ⟶ List 1 | List 2

n elements | $\frac{n}{2}$ elements | $\frac{n}{2}$ elements

min ⟶ Min (min 1, min 2)

max ⟶ Max (max 1, max 2)

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]);

}

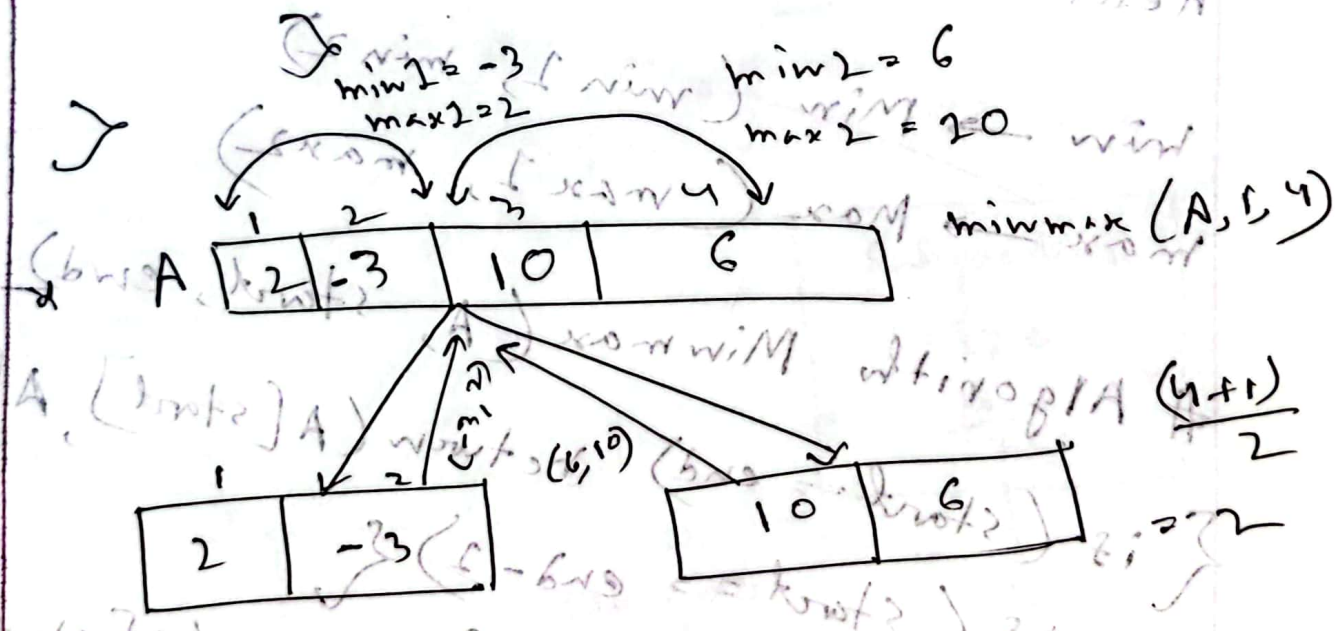
else {

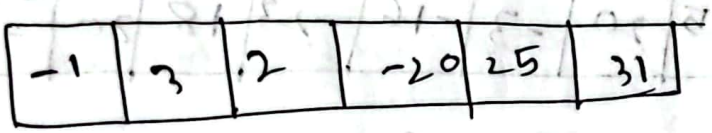
mid = ⌊(start + end) / 2⌋;

(min1, max1) = Minmax (A, start, mid);

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

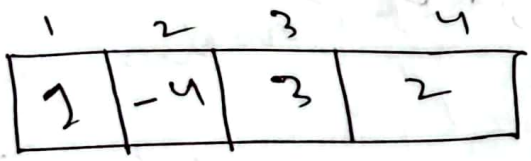
return (Min (min1, min2), Max (max1, max2));

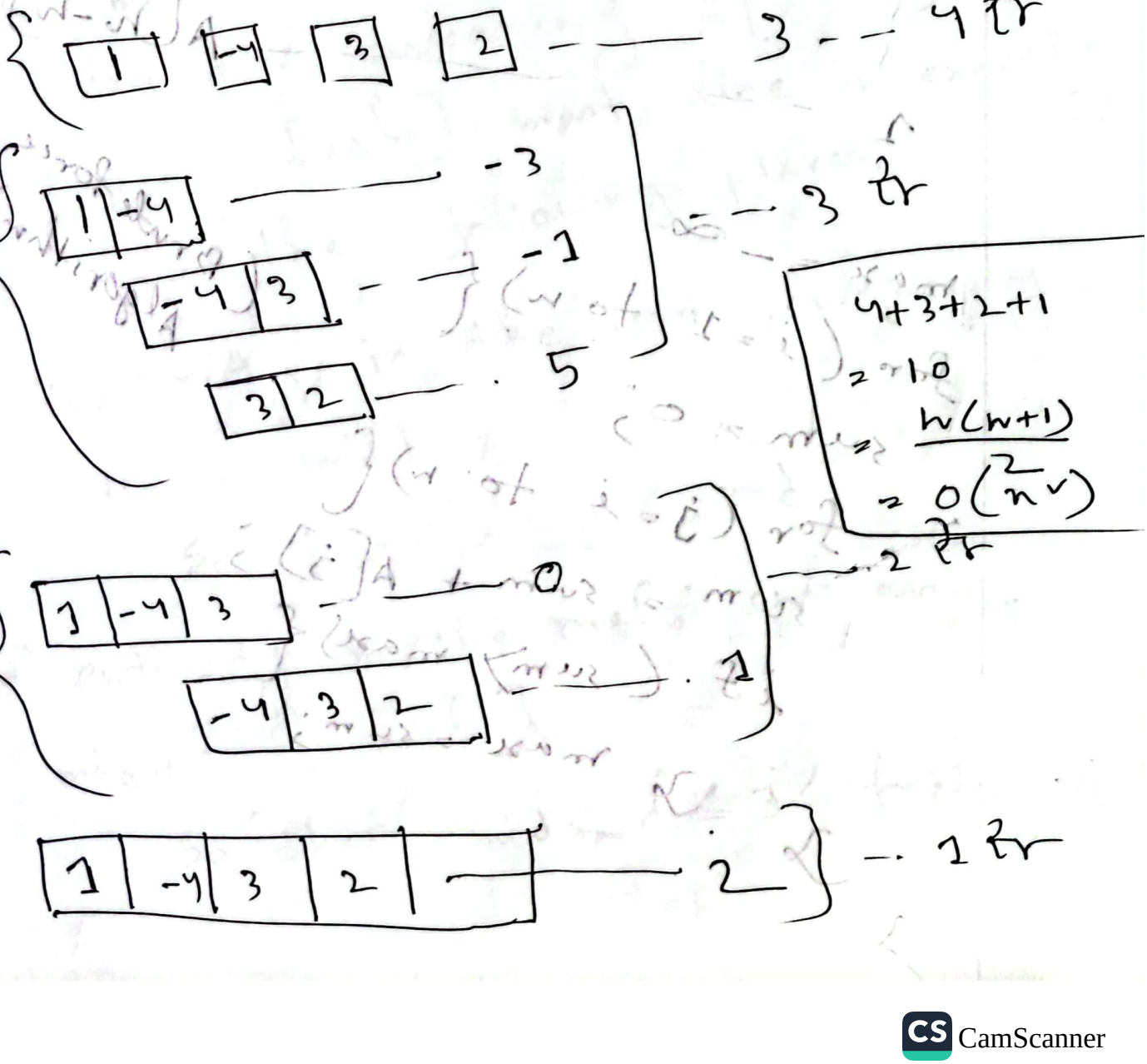




CW 10.10.23

Maximum Sum Subarray — Divide & Conquer





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

* n — A[1-1], A[1-2], A[1-3] ... A[1-n]
* n-1 — A[2-2], A[2-3], A[2-4] ... A[2-n]
* n-2 — A[3-3], A[3-4], A[3-5] ... A[3-n]

1 — A[n-n]

(Brute force Algorithm)

```
max = -∞
for (i = 1 to n) {
    sum = 0;
    for (j = i to n) {
        sum = sum + A[j];
        if (sum > max) {
            max = sum;
        }
    }
}
```

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

① entirely in the left sub array A[low ... mid]

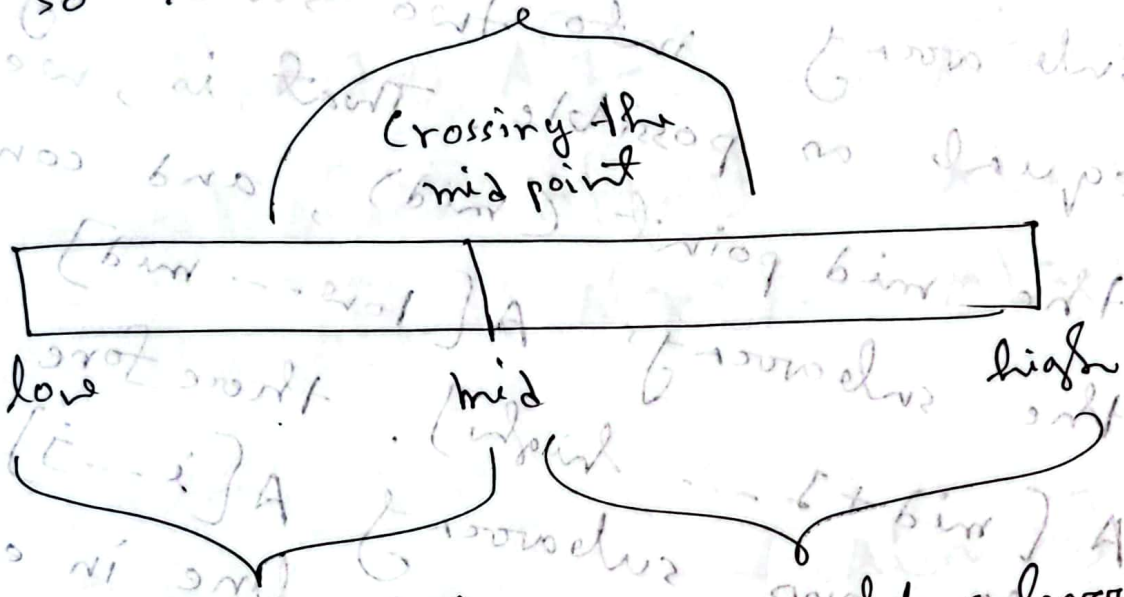
So $low \le i \le j \le mid$

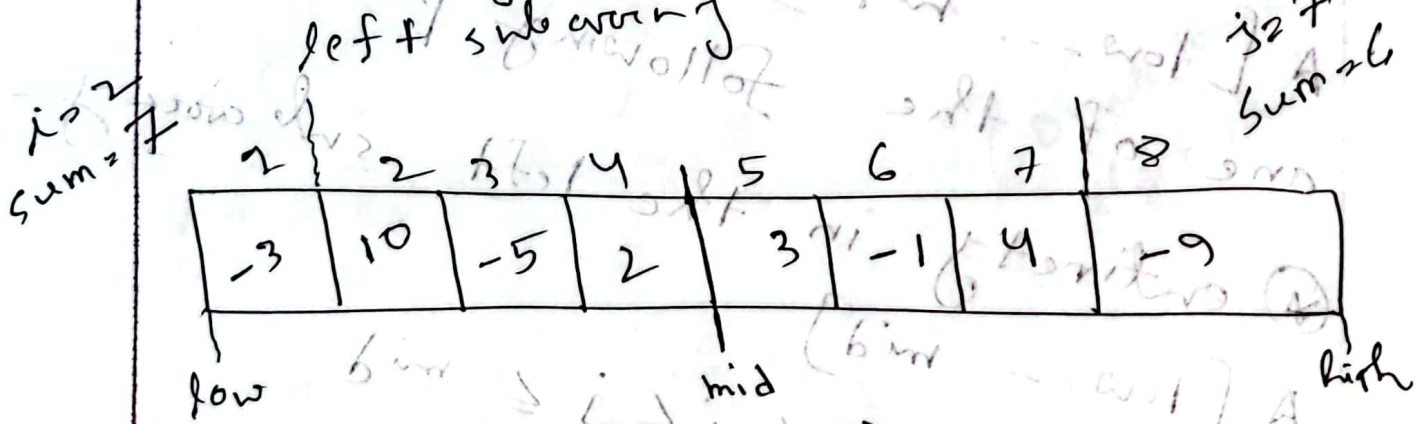
② entirely in the right sub array [mid+1 ... high]

So that $mid \le i \le j \le high$

(*) Crossing the mid point,

so $low \le i \le mid < j \le high$





2 3

-3 2

7 (max) 6 (max)

4 3

(2, 7, 7+6)

```
Find_max_crossing_subarray(A, low, mid, high)

left_sum = -∞;
sum = 0;
for( i = mid ) down to low) {
    sum = sum + A[i]
    if( sum > left_sum){
        left_sum = sum;
        max_left = i;
    }
}
right_sum = -∞
sum = 0;
for( j = mid+1 to high) {
    sum = sum + A[j];
    if( sum > right_sum) {
        right_sum = sum;
        max_right = j
}
```

return (max_left, max_right,
left_sum + right_sum);

⇒ O(n)

Find_max_subarray (A, low, high)

if (low == high)

return (low, high, A[low]);

mid = (low + high) / 2;

(left_low, left_high, left_sum)

= Find_max_subarray (A, low, mid);

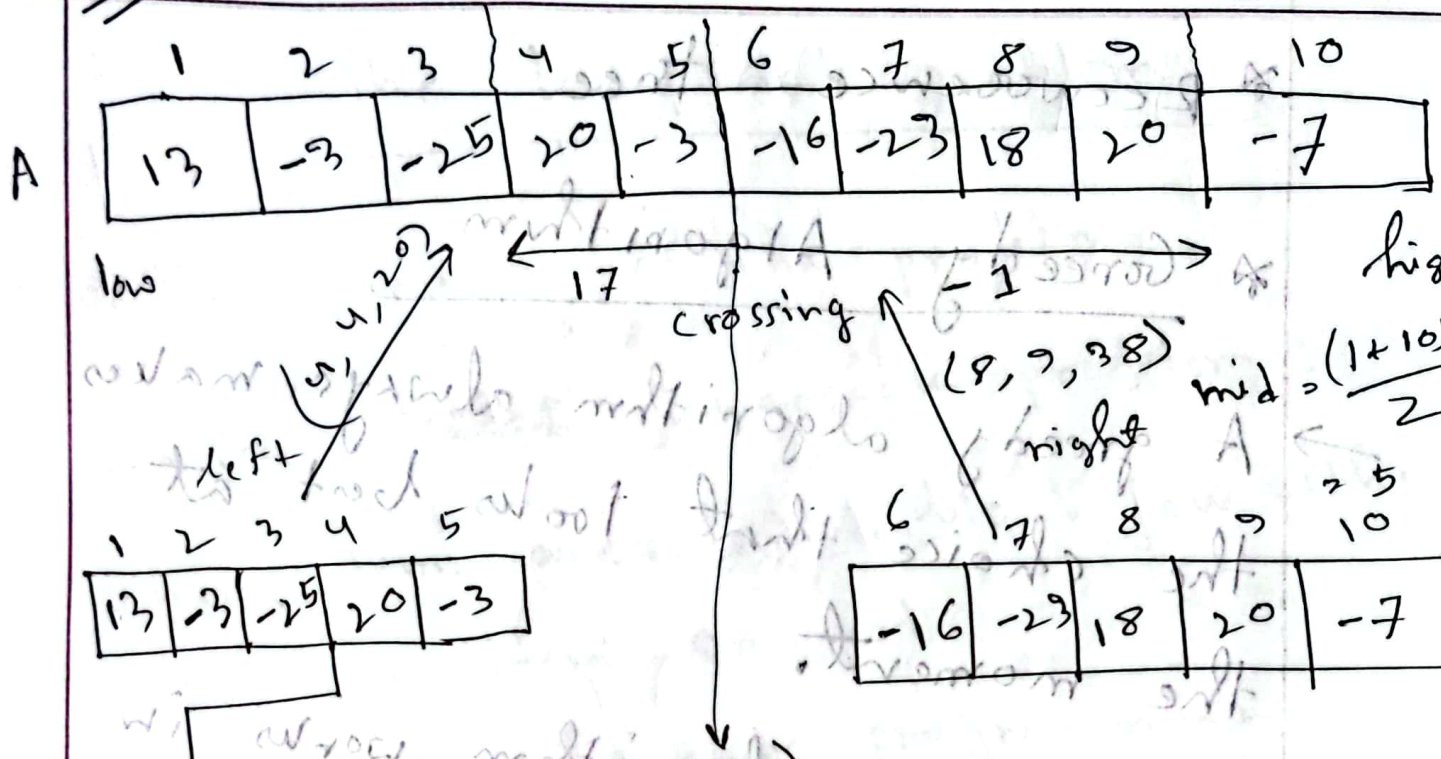
(right_low, right_high, right_sum)

= Find_max_subarray (A, mid+1, high)

CW 14.10.23

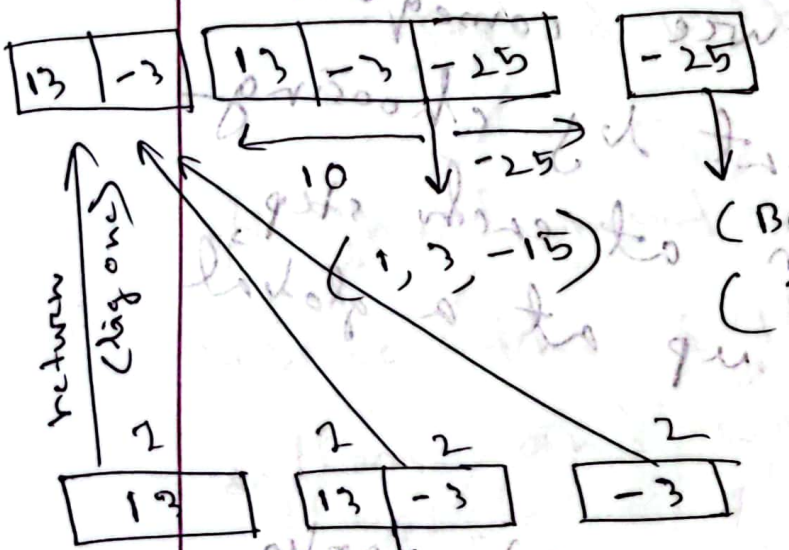
Divide & Conquer

DSA-2



(1,4,5)
- sum
- End
- start

(4,5,16)
- sum
- Ending index
- starting index



* Merge Sort for mid (simulation)

* Recurrence tree

* Greedy Algorithm

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

→ A greedy algorithm works in phases

→ you take the ~~place~~ 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 version of the problems.

→ The '0-1' knapsack problem:
here, items are indivisible: you either take an item or not.
solved by dynamic programming.

→ Fractional knapsack:
→ Items are divisible
→ solved with a greedy algorithm.

Theif must choose among n items, where the i'th item worth v_i dollars and weighs w_i pounds.
→ theif can carry W pounds.
maximize value.

* Fractional knapsack:

→ knapsack capacity : W

→ there are n item, the i^{th} item has value v_i and weight w_i

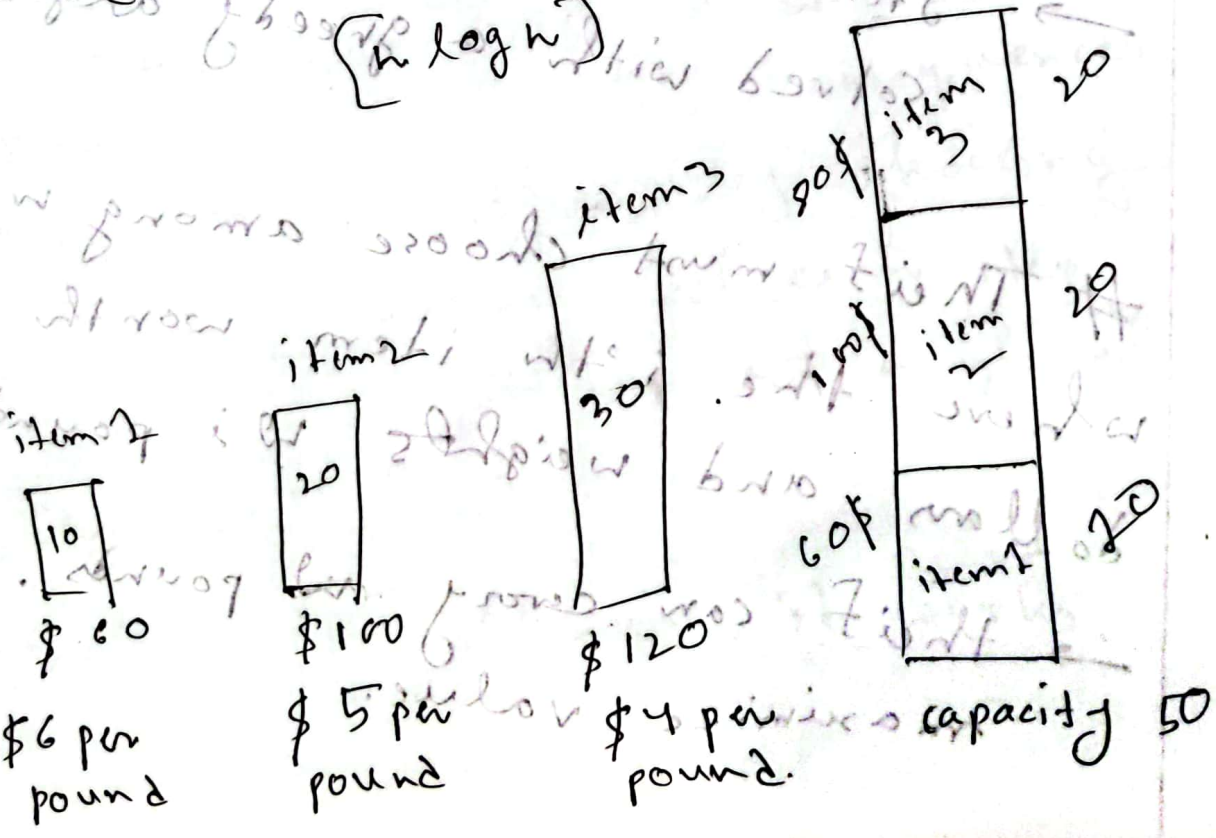
→ Goal: find x_i such that for all $0 \le x_i \le 1$

$i = 1, 2, 3, 4, \ldots n$

$\sum w_i x_i \le W$ & $\sum x_i v_i$ is maximum

→ $(n \log n)$ [cost complexity]

[$n \log n$]



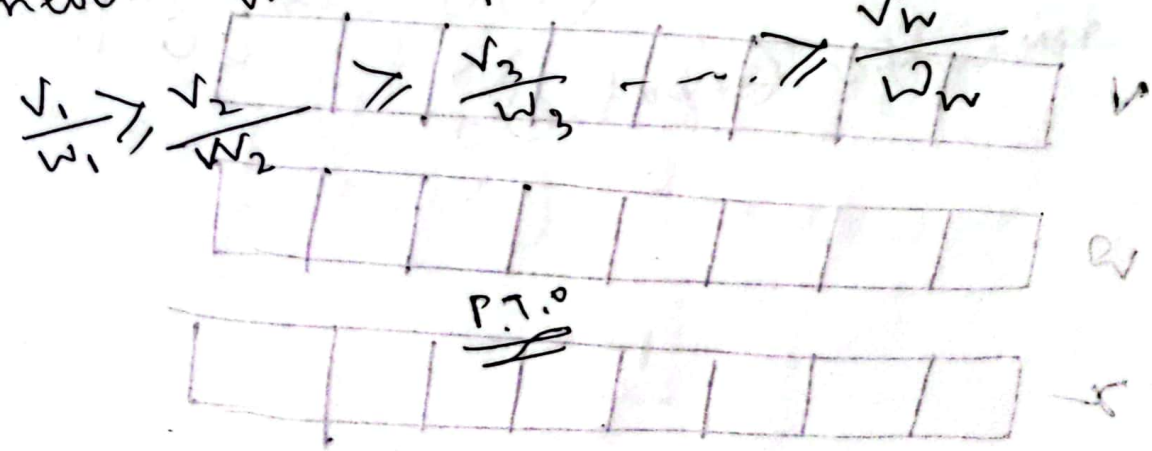
GW
16.10.23

28-10-23
Divide & Conquer
greedy

Fractional Knapsack

strategy

* Pick the item with the maximum value per pound 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.



P.T.O

V = value
W = weight
x =

* Fractional - knapsack (W, V[n], W[n])

{ which W>0 & there are items remaining pick item with $\frac{v_i}{w_i}$

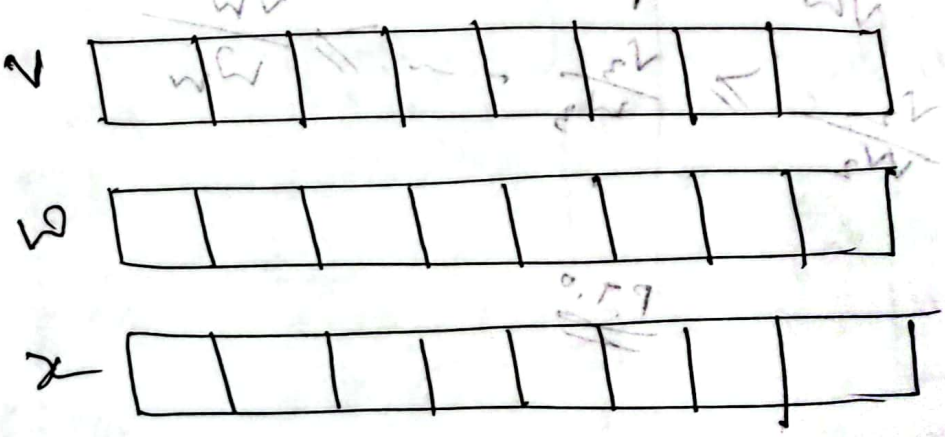
$x \leftarrow \min\left(1, \frac{W}{w_i}\right)$; [0≤xi≤1]

remove item i from list

$W \leftarrow W - x_i w_i$;

W → the amount of space remaining in the knapsack (initially, W = W)

* Running time:



Mathmatical simulation.

* $W = 50$ [bag capacity]

$w_i = 10$ [item capacity] (gold)

$\frac{W}{w_i} = 5$

$x = \min(1, 5)$

$x_i = 1$

$W = 50 - 1 \times 10$

$= 40$

$w_i = 60$ kg [silver]

$\frac{W}{w_i} = \frac{40}{60} = \frac{2}{3}$

$x_i = \min\left(1, \frac{2}{3}\right)$

$= \frac{2}{3}$

$W = 40 - \frac{2}{3} \times 60$

$40 - 2 \times 20$

$= 0$

Running time: $O(n)$ if ordered

$O(n \log n)$ otherwise

*

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

* An activity selection problem:

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

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

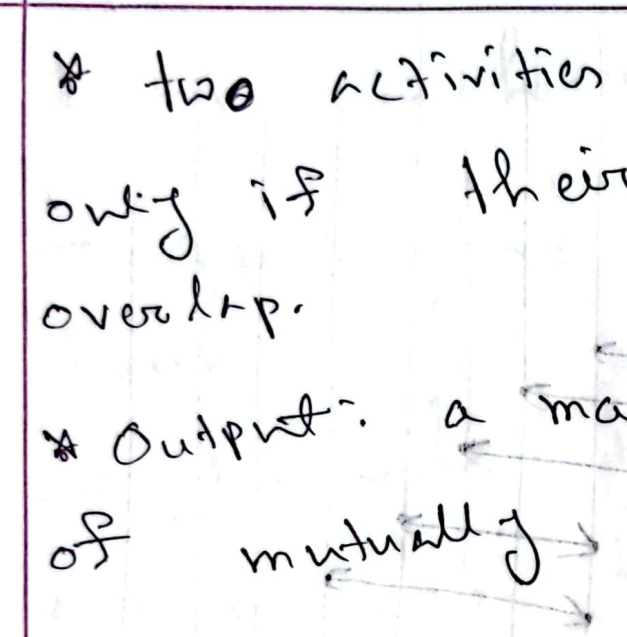
where $0 \leq s_i \leq f_i < \infty$

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

P.T.O

* two activities are compatible if they do not overlap.

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

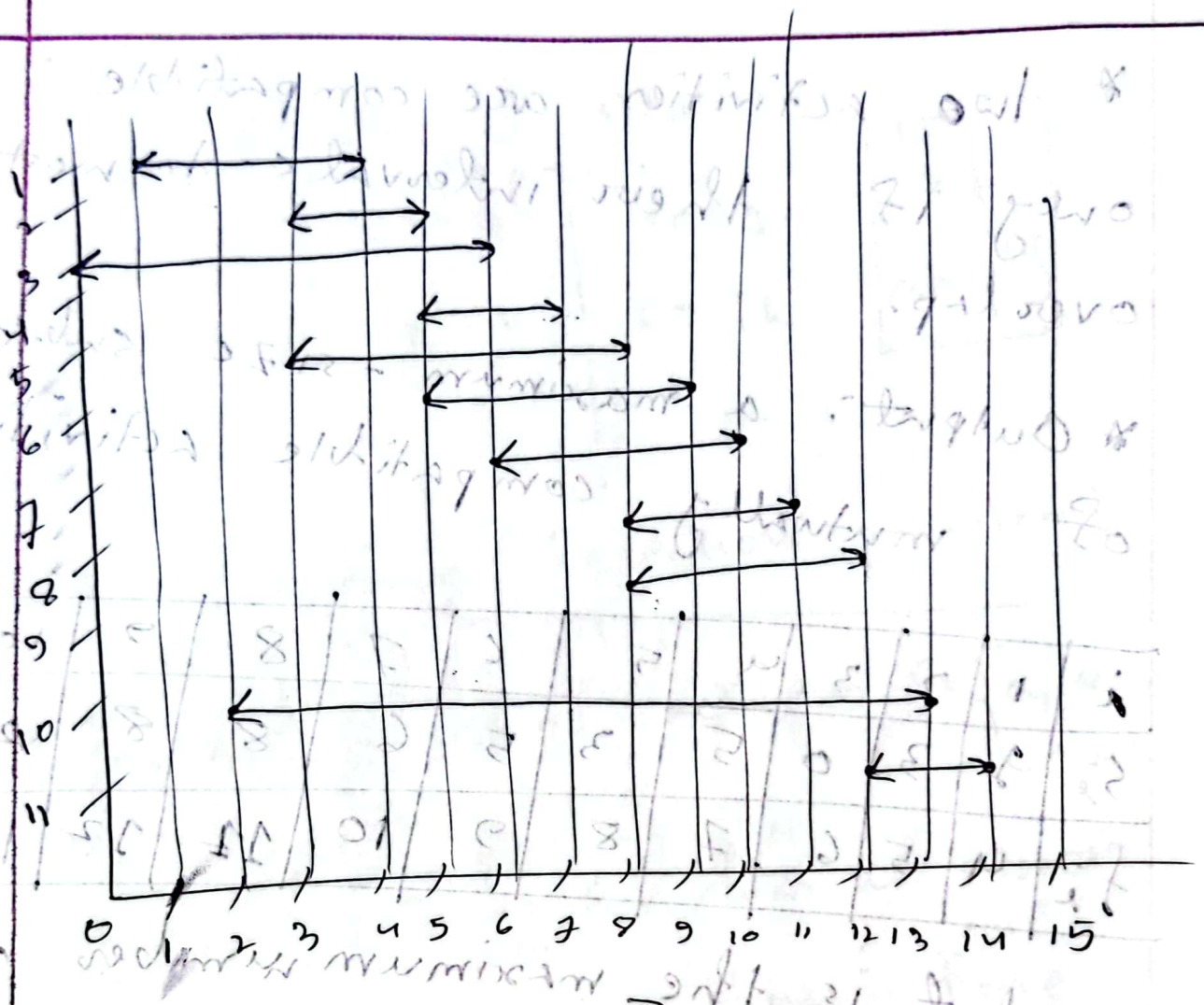


i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

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

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

PTO



$[a_1, a_4, a_8, a_{11}]$

Early finish Greedy

→ select the activity with earliest finish.

→ Eliminate the activities that could not be scheduled.

→ Repeat

Sudocode (Do it yourself)

Dynamic Programming

Greedy

Build up a global solution incrementally by optimising some local criteria.

Divide & Conquer:

Break up a problem into disjoint subproblems, solve the subproblems 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 largers

P.T.O

and larger subproblems. Typically some subproblems are generated repeatedly when a recursive algorithm is running.

* Fib(n){
  if (n = 0 or 1) {
  return n;
  }
  else {
  return Fib(n-1) + Fib(n-2);
  }
}

P_7.0

0-1 knapsack problems

→ knapsack capacity : W

→ n items to choose form.

→ pack the knapsack to achieve maximum total value.

Goal:

Find x_i such that for all

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

$i = 1, 2, 3, \ldots n$.

$\sum w_i x_i \leq W$

$\sum v_i$ is maximum.

P.T.O

a b c d e (W)

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

(knapsack এর size W ধরে, 1 থেকে i পর্যন্ত item ব্যবহার করে max profit কত?)

Case : 1

i তম item নিলে

$P(i, w) = V_i + P(i-1, w-w_i)$

2. i তম item নিলে না.

$P(i, w) = P(i-1, w)$

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

capacity, $W = 5$

item \ W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

CW
21.10.2...

Dynamic Programming

* 0-1 knapsack

item	wieght	value
1	2	12
2	1	10
3	3	20
4	2	15

Capacity, W = 5

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

item \ weight	1	2	3	4	5
0	0	0	0	0	0
1	0	0	12	12	12
2	0	10	12	22	22
3	0	10	12	22	30
4	0	10	15	25	30

item {4, 2, 1}

* না দিলে ~~ইনপুট~~ value ও দিবে,
* দিলে weight অনুযায়ী বসাব, ২ ধরণ ইনপুট ও value ~~আসবে~~।
তার মধ্যে main item value বসাব হবে।
* Runtime : O(n * w)
* Sudocode : Do it yourself

Coin change Problem

* given unlimited amount of coins of value $C_1, C_2, \dots C_d$, give change for amount M with the least number of coins.

P.T.O

Ex: C_1 C_2 C_3 C_4

$C = \{25, 10, 5, 1\}$ and $M = 48$

Solution:

$25 * 1 + 10 * 2 + 1 * 3$

$= C_1 * 1 + C_2 * 2 + C_4 * 3$

6

Given the denominations 1, 3 & 5, what is the minimum number of coins needed to make change for a given value.

$M = 10$

(TV)

values	1	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 | 2+1 | 3+1 | 2+1 | 3+1

1+1 | 2+1 | 1+2 | 2+1 | 3+1

1+1 | 3+1 | 1+1 | 2+1 | 1+1

How many previous coin needed + 1 → always need

Algorithm note

Assuming activities are sorted by finish time:

⇒ Greedy-Activity-selector (s, f) {

n ← length[s];

A ← {a₁};

for m ← 2 ton

do if sm ⩾ fi ;

then A ← A ∪ {aₘ};

i ← m;

return A;

}

* Merge-sort (A, P, r)

if P < r

then q ← ⌊(P+r)/2⌋

Merge-sort (A, P, q)

Merge-sort (A, q+1, r)

Merge (A, P, q, r)

Merge sort running time

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

$$= 2\left\{\left(2T\left(\frac{n}{4}\right)\right) + b\frac{n}{2}\right\} + bn$$

$$= 4T\left(\frac{n}{4}\right) + 2b\frac{n}{2} + bn$$

$$= 4T\left(\frac{n}{4}\right) + bn + bn$$

$$= 4T\left(\frac{n}{4}\right) + 2bn$$

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

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

$$2^i = n$$

$$\Rightarrow i = \log n$$

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

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

C.W 30.10.23

DSA 1J

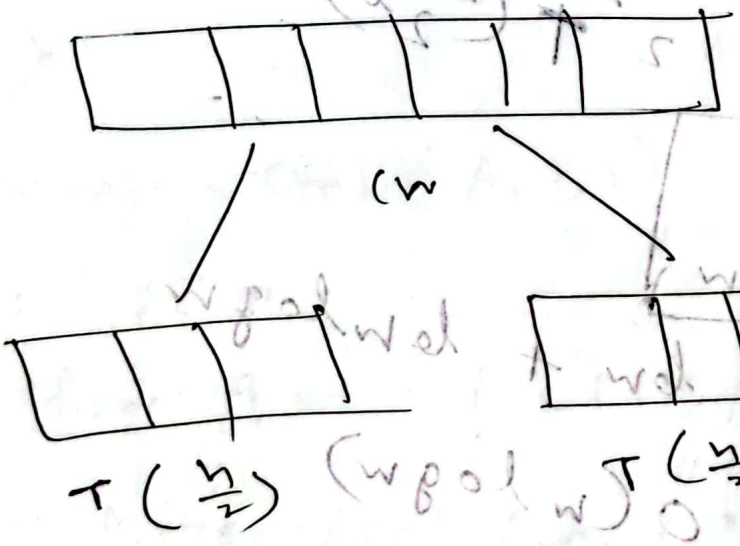
Asymtotic Analysis for Recurrence

Three method

→ substitution method

→ Recursion Tree method

→ Master method



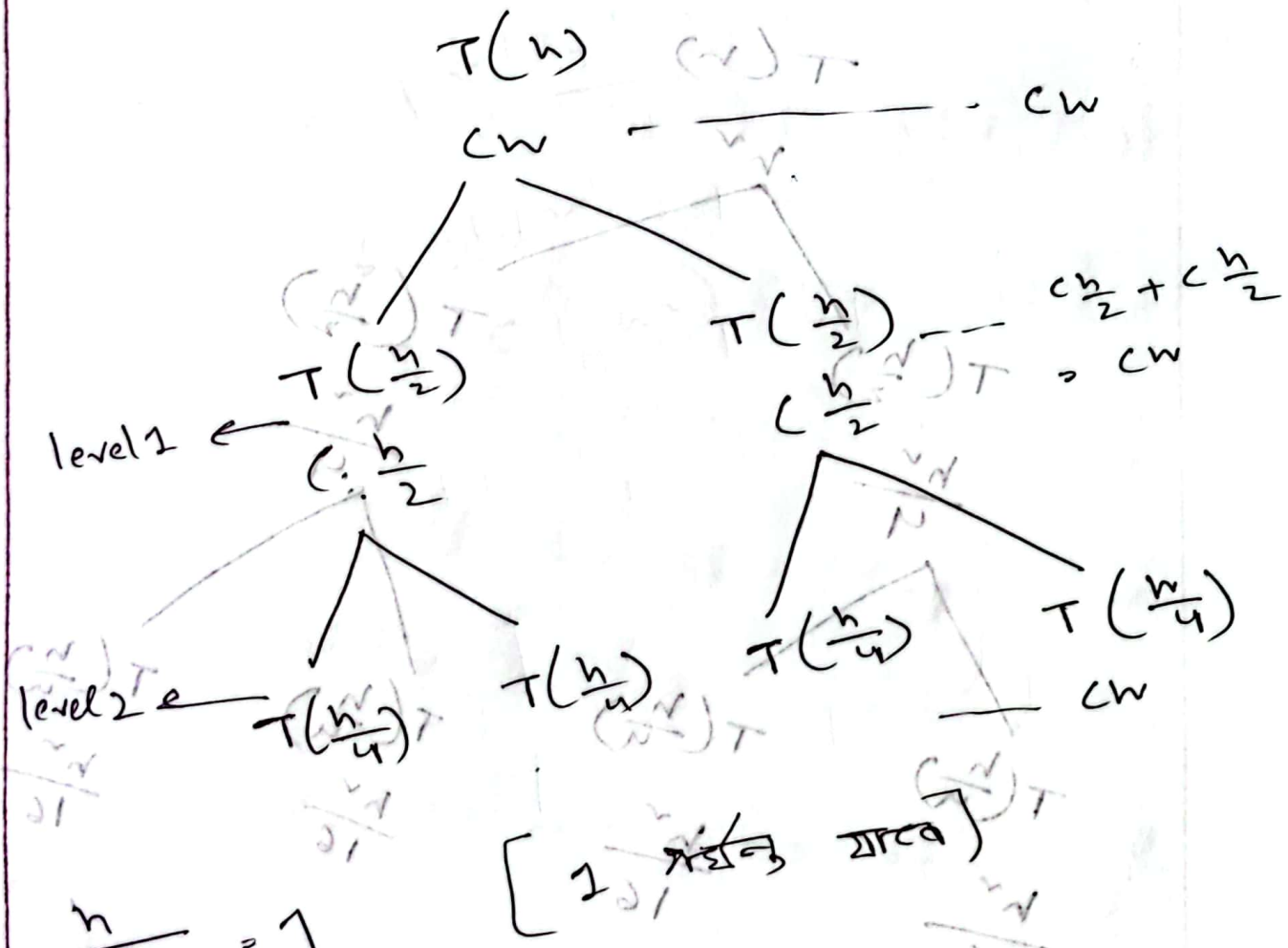
$T(n) = T(\frac{n}{2}) + T(\frac{n}{2}) + cn$

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

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

$n \to \frac{n}{2} \to \frac{n}{4} \to \frac{n}{8} \cdots \frac{n}{2^k}$

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



[1 লেভেল যাবে]

$\frac{n}{2^k} = 1$

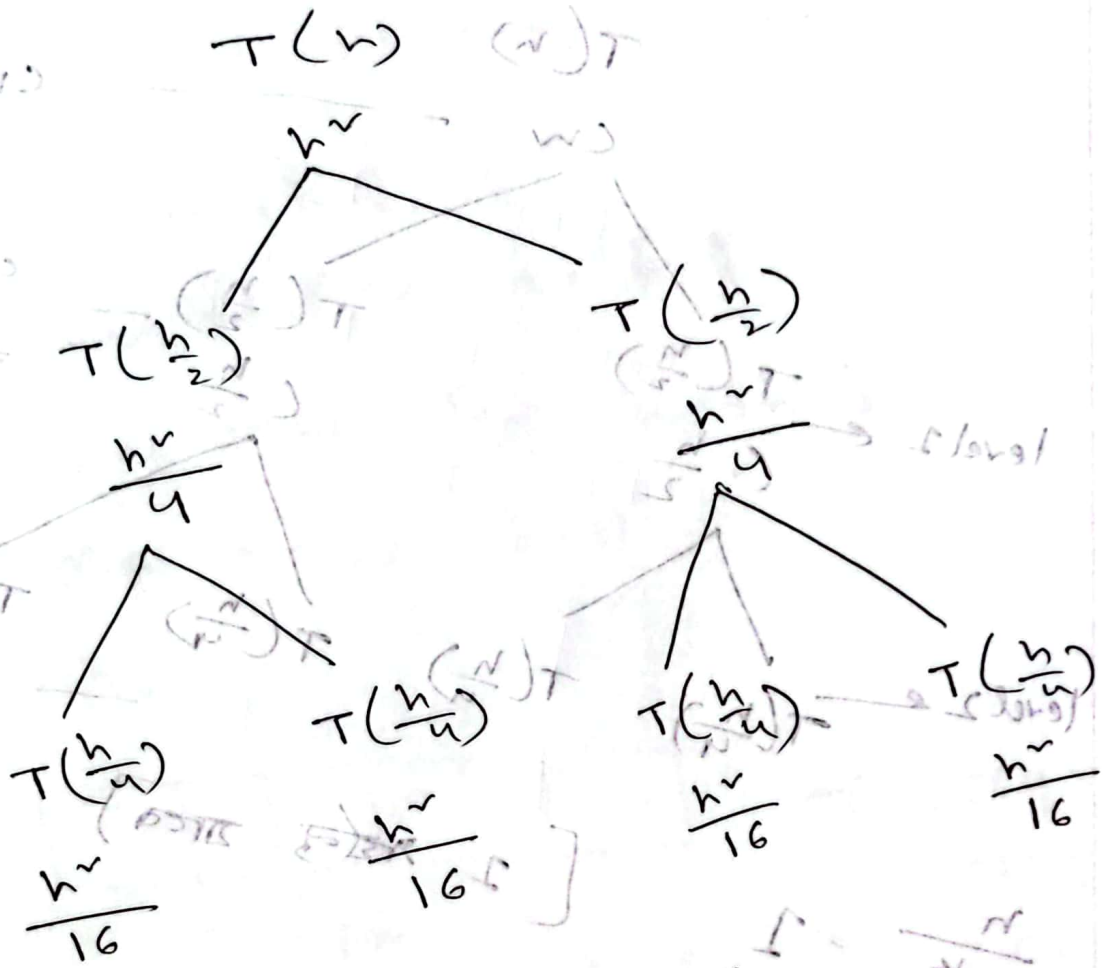
$\Rightarrow k = \log_2 n$

$T(n) = cn + cn + cn + \cdots$

$= cn \log n$

$= O(n \log n)$

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



$$T(n) = n^2 + 2 \cdot \frac{n^2}{4} + 4 \cdot \frac{n^2}{16} + \cdots$$

$$= n^2\left(1 + \frac{2}{4} + \frac{4}{16} + \cdots\right)$$

$$= n^2\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots\right)$$

P.T.O

* Recurrance equation Master Formula

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

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

n = input size of the problem.

a = number of subproblems

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

(i) if $a > b^k$, $T(n) = \theta\left(n^{\log_b a}\right)$

(ii) if $a = b^k$

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

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

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

$\log^0 k = 1$

$\log^1 k = \log k$

(iii) $a < b^k$,

if $p >= 0$, $T(n) = \theta(n^k \log^p n)$

if $p < 0$, $T(n) = \theta(n^k)$

* $T(n) = 2T(\frac{n}{2}) + n^2$

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

$b^k = 2^2 = 4$

$a < b^k$

$T(n) = \theta(n^2 \log^0 n)$

$= \theta(n^2)$

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

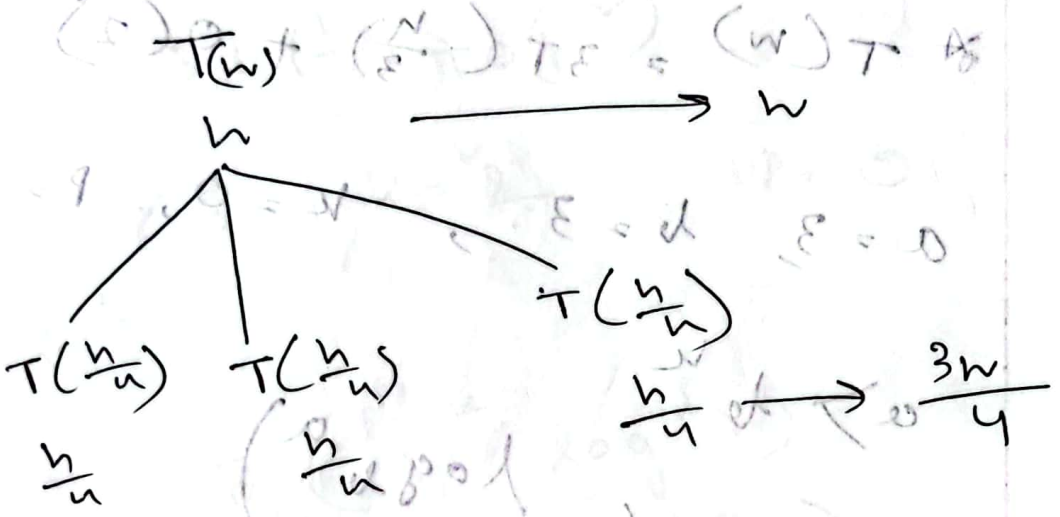
$a = 2, b = 2, k = 1, p = 0$

$a = b^k$

$\therefore T(n) = \theta(n^{\log_b a} \cdot \log^{p+1} n)$

$= \theta(n^{\log_2 2} \cdot \log n)$

$= \theta(n \log n)$



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

$a = 3, \quad b = 4, \quad k = 1, \quad p = 0$

$a = 3, \quad b^k = 4$

$a < b^k \qquad p \geq 0$

$$\therefore \quad T(n) = \theta\left(n^k \log^p n\right)$$

$$= \theta\left(n^1 \log^0 n\right)$$

$$= \theta(n)$$

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

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

$a > b^k$

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

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

$= \Theta(n)$

*

$\log^0 n = 1$

$\log^2 n = \log n$

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

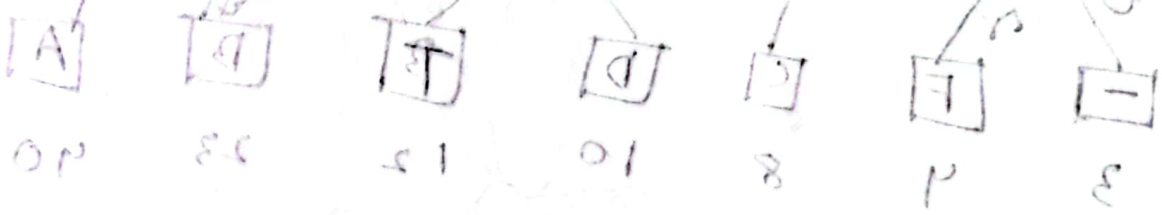
$a = 3, \quad b = 3, \quad k = 2 \qquad p = 0$

$a < b^k$

$\therefore T(n) = \theta(n^k \log^p n)$

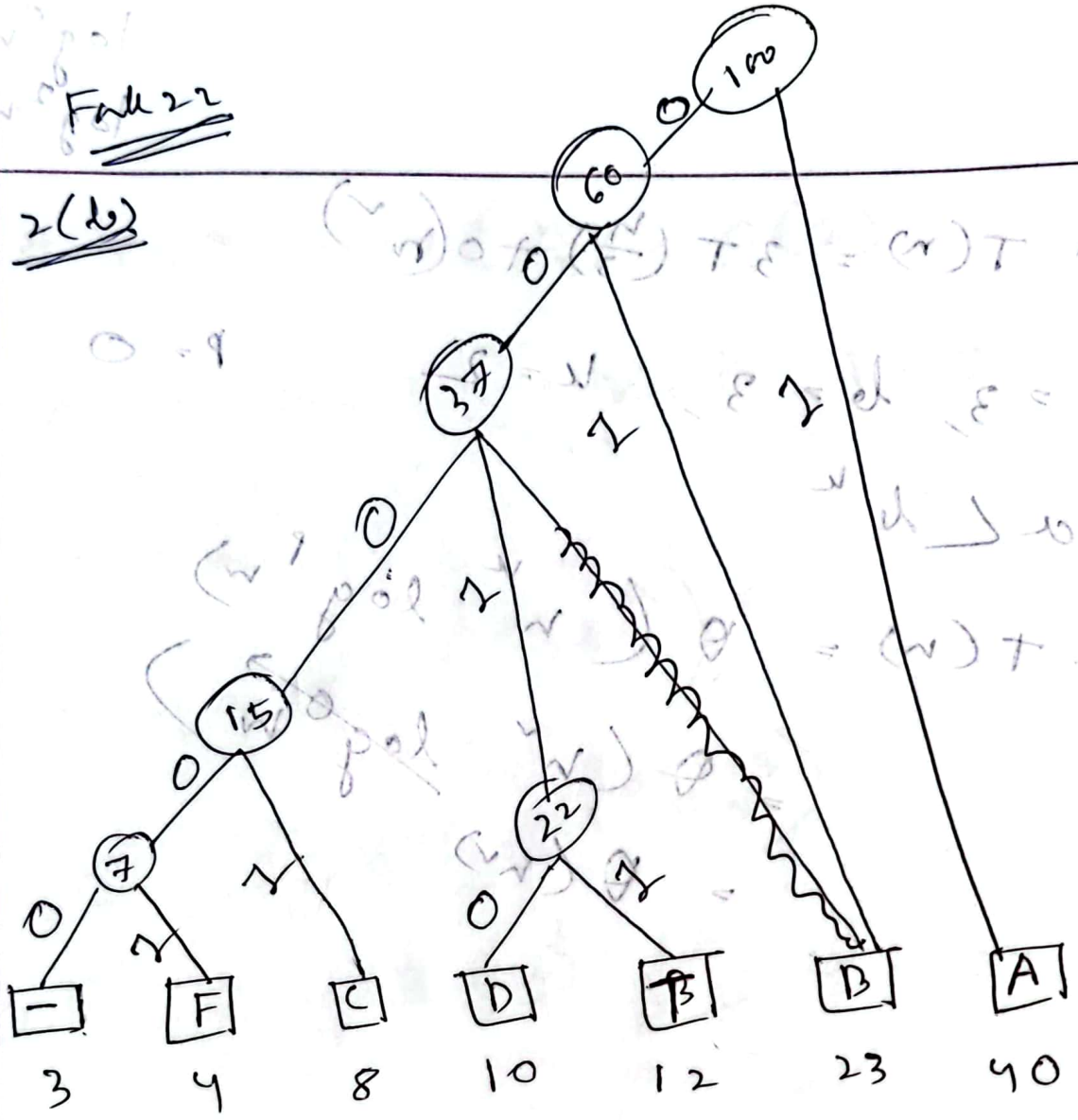
$= \theta(n^2 \log n)$

$= \theta(n^2)$



Fall 22

2(b)



0 1 1 0 0 0 1 1 1 1

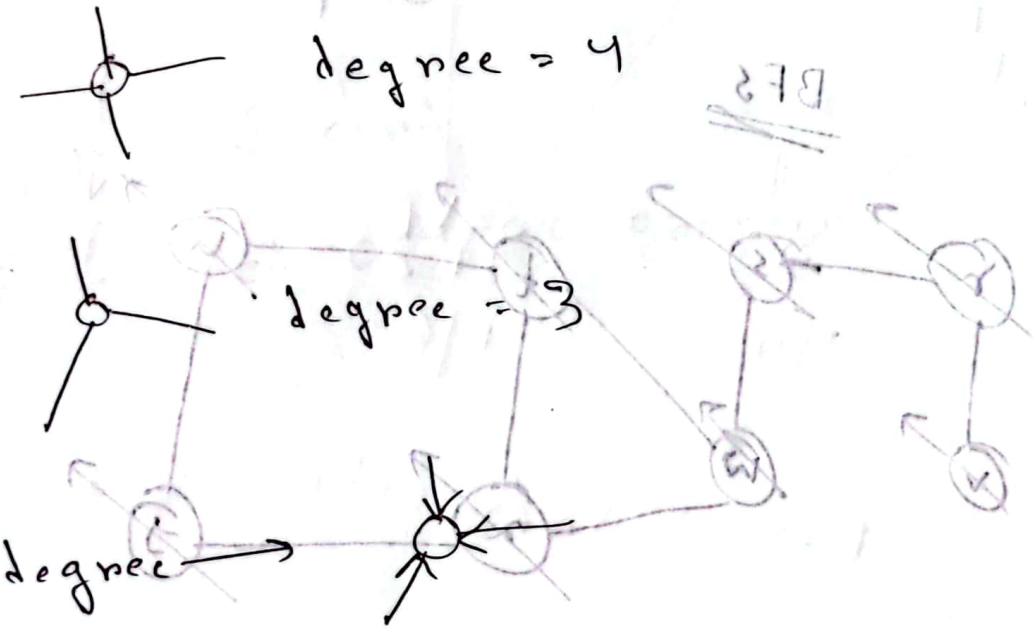
B A E C A A A

BAECAAA

Graph representation

→ Adjacency matrix

→ Adjacency list



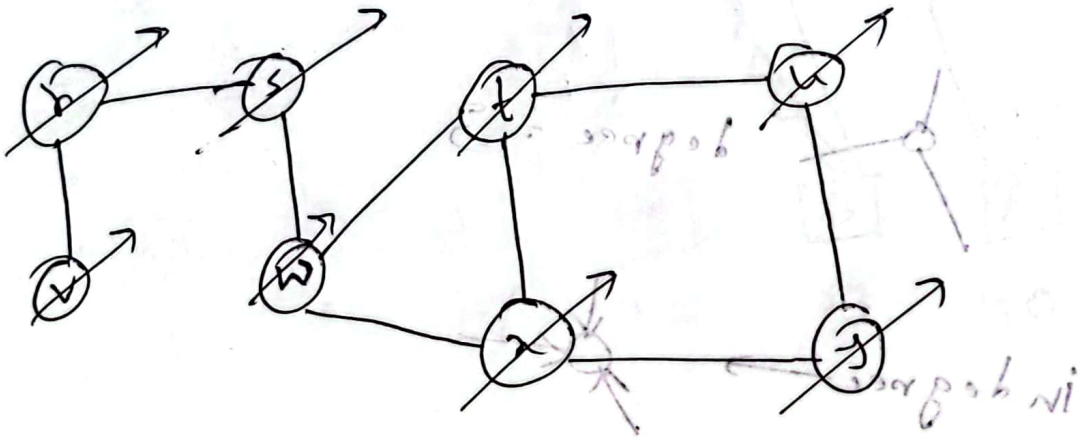
check:

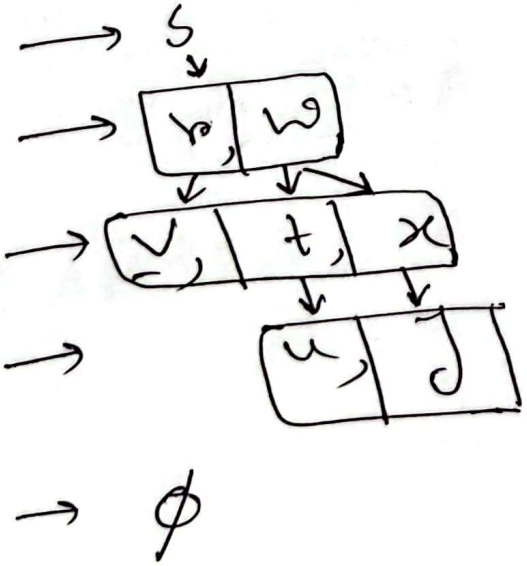
indegree → row check

outdegree → column check

Complexity → O(n)

BFS

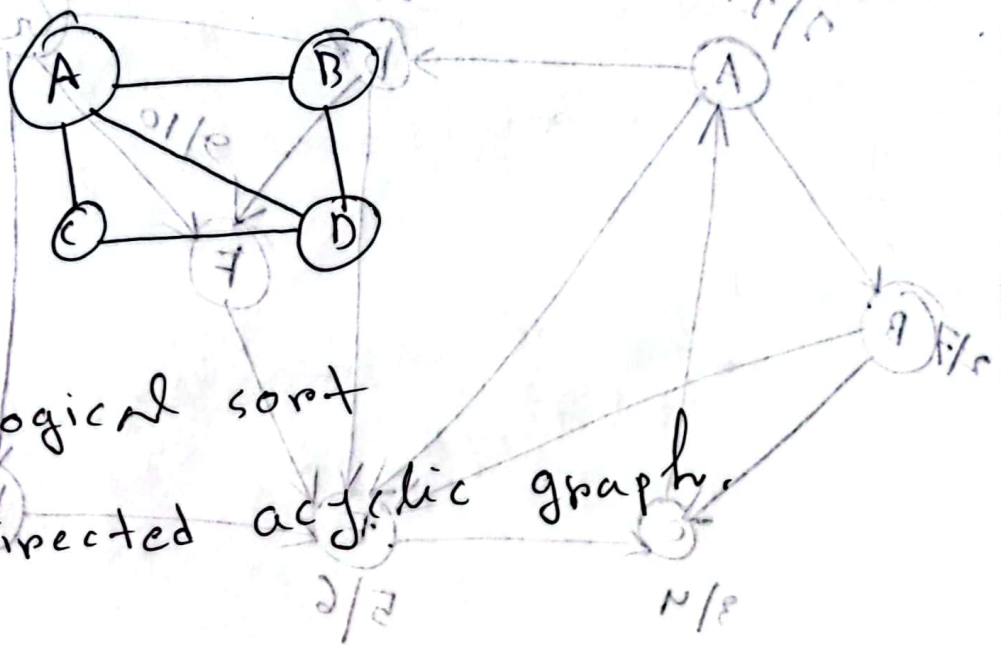




$T(n) = O(V+E)$

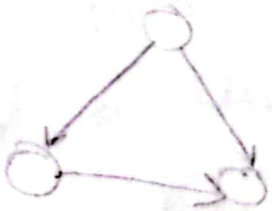
if visited,
no need to check
2nd time

BFS → Disconnected graph



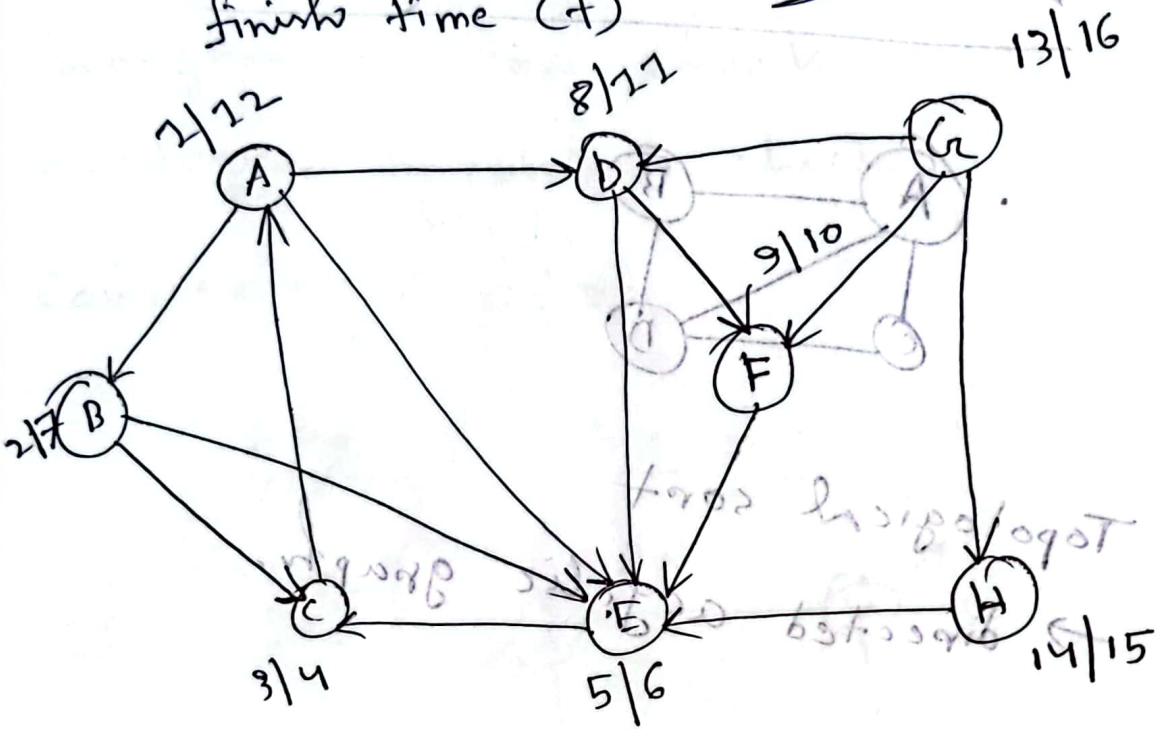
⇒ Topological sort

⇒ directed acyclic graph.

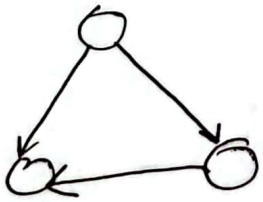


14.11.23

discover time (d) (DFS)
finish time (f)

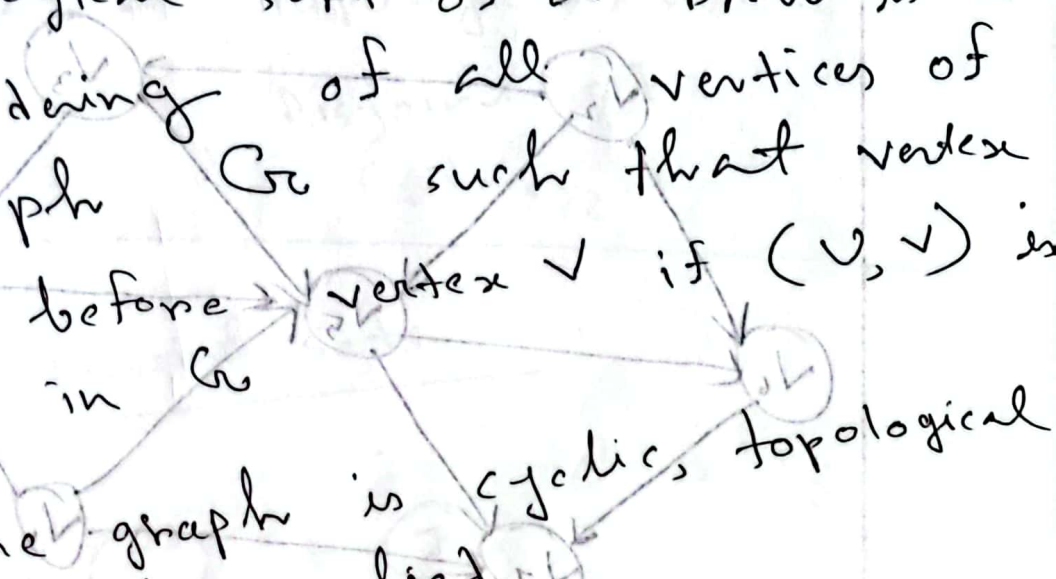


Directed Acyclic Graphs (DAGs)



P.T.O

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

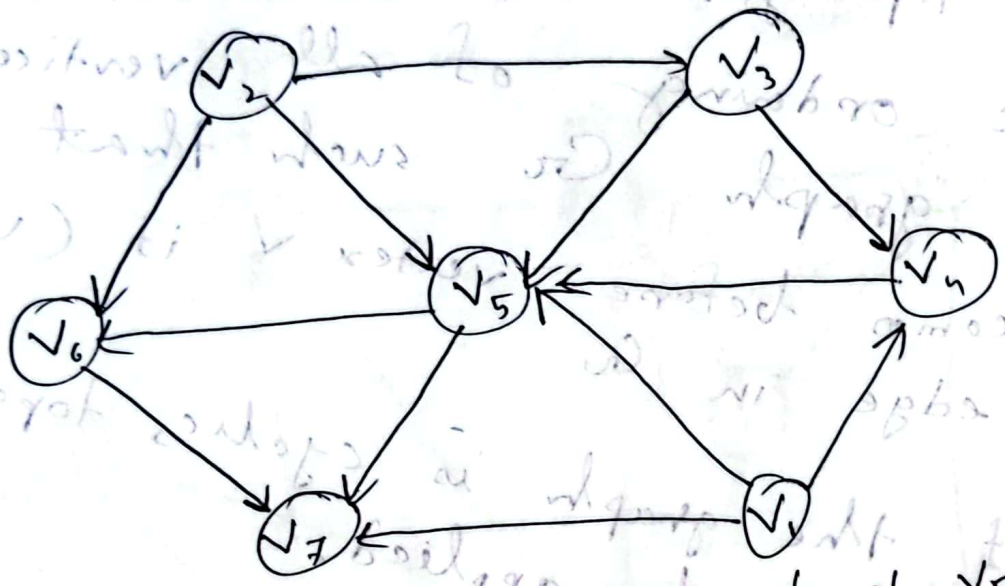


if the graph is cyclic, topological sort can't be applied.

Topological sort (G)

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

Time: O(V+E).



$V_1, V_2, V_3, V_4, V_5, V_6, V_7$

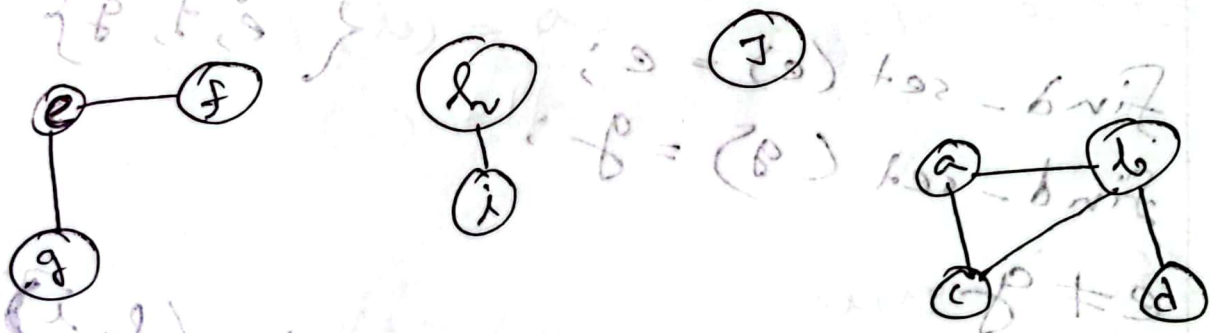
Vertex	indegree
V_1	~~0~~
V_2	~~0~~
V_3	~~1~~ 0
V_4	~~2~~ ~~1~~ 0
V_5	~~4~~ ~~3~~ ~~2~~ ~~1~~ 0
V_6	~~2~~ ~~1~~ 0
V_7	~~3~~ ~~2~~ ~~1~~ 0
"	

22 → 11 → 23 → topological sort

Disjoint set

C.W
26.11.23

Disjoint sets



Make-set(x):

{a}, {b}, {c}, {d}, {e}, {f},
{g}, {h}, {i}, {j}

1. e - f
2. e - g
3. h - i
4. a - b
5. a - c
6. b - c
7. b - d
8. j

Algorithm:

for each edge (u,v) in E(G)
if find_set(u) != find_set(v)
Union(u, v),

2. e-f

find_set (e) = e;

find_set (f) = f;

e ≠ f

union (e, f)

{ e, f }

2. e-g

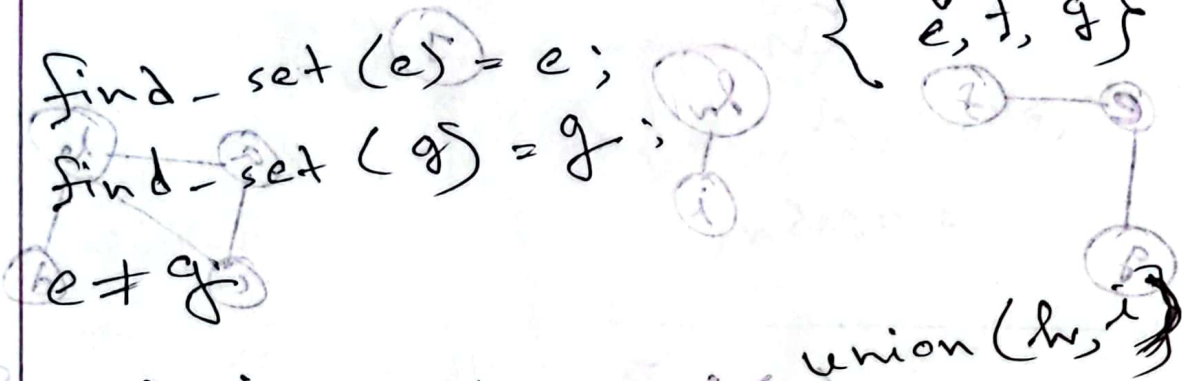
find_set (e) = e;

find_set (g) = g;

e ≠ g

union (e, g)

{ e, f, g }



3. h-i

find_set (h) = h;

find_set (i) = i;

h ≠ i

union (h, i)

{ h, i }

4. a-b

find_set (a) = a;

find_set (b) = b;

a ≠ b

union (a, b)

{ a, b }

5. a - c

union(a,c)

~~union~~

find_set(a) = a

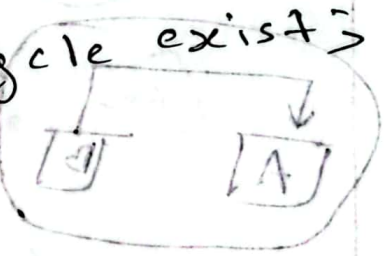
find_set(c) = c;

$a \neq c$

6. b - c

find_set(b) = a;

find_set(c) = a;



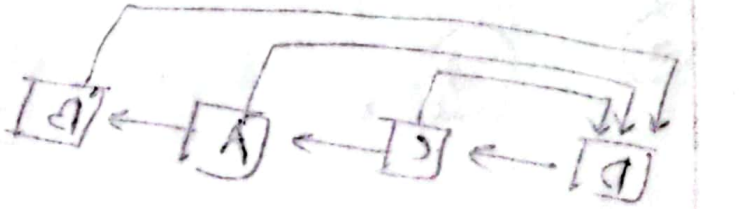
7. b - d

union(a,d)

find_set(b) = a;

find_set(d) = d;

$a \neq d$



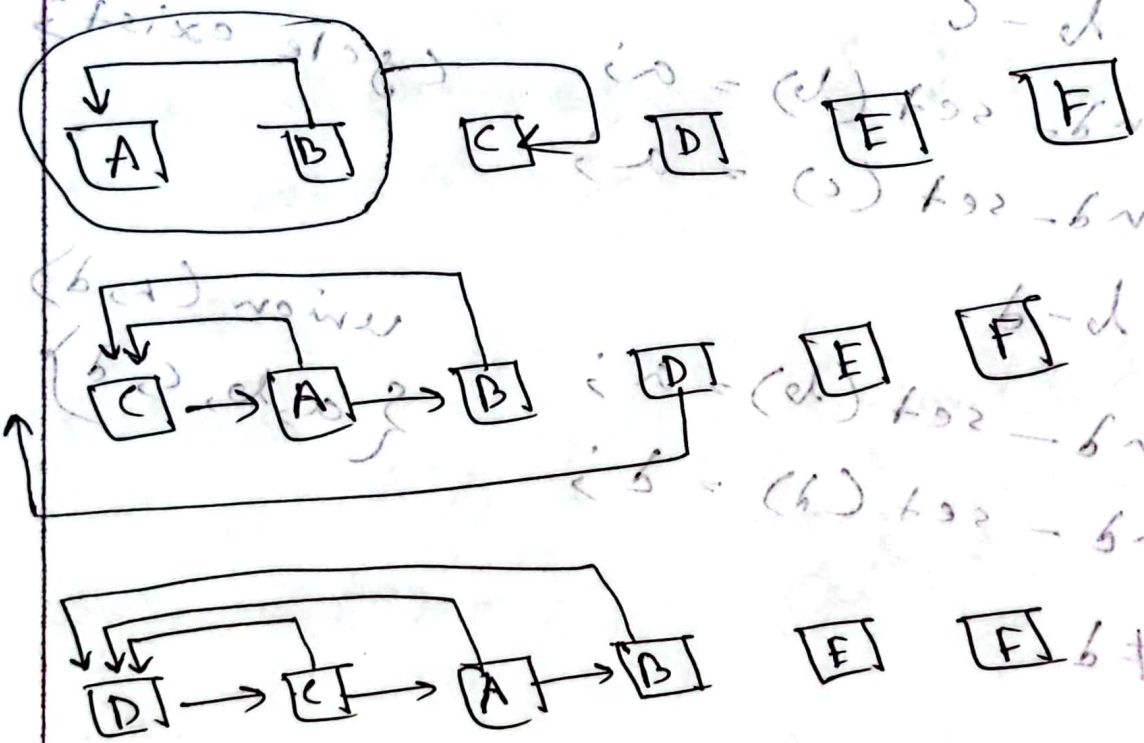
8. { }.

Cost analysis

Union & call = $|V| - K$

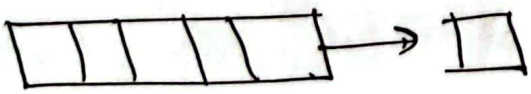
Find_set = $2|E|$

Connected components

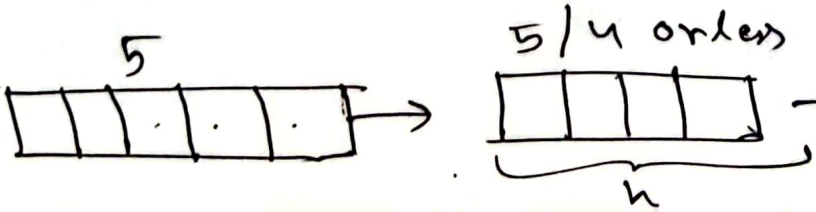


$1 + 2 + 3 + \dots + n-1$

Worst case = $O(n^2)$



$O(n)$



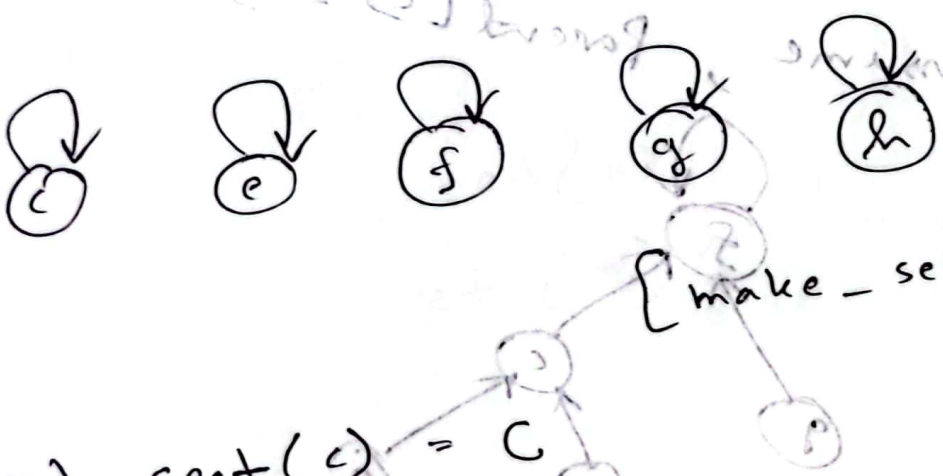
$\rightarrow O(\log n)$

*

c → h

c → e

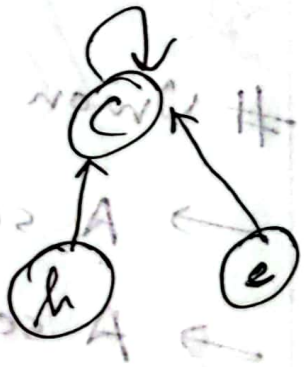
f → g



find_set(c) = c

find_set(h) = h

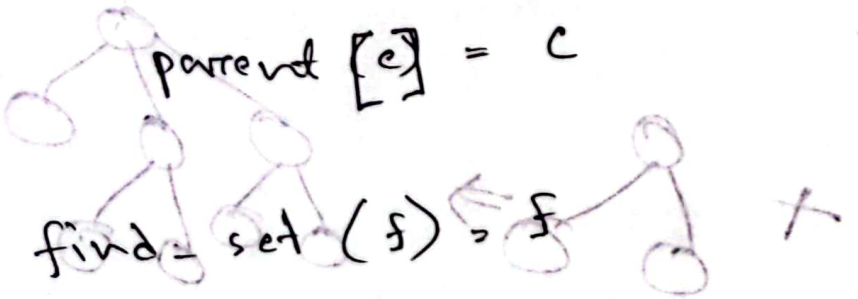
Parent[h] = c



find_set(c) = c

find_set(e) = e

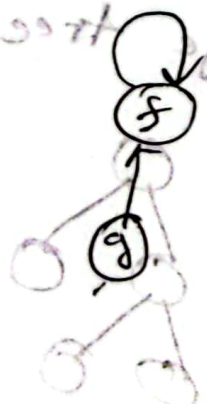
parent[e] = c



find_set(f) = f

find_set(g) = g

parent[g] = f

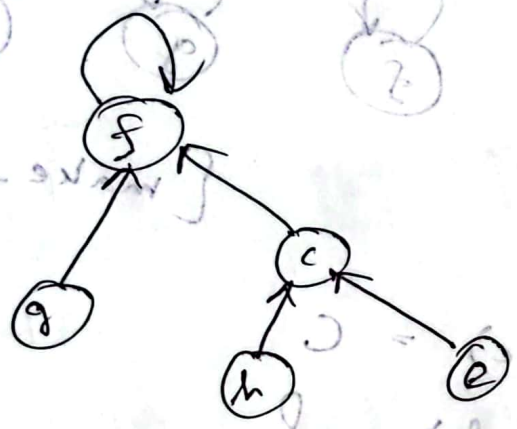


Union (e, g)

find_set(e) = c

find_set(g) = f

assume Parent[c] = f

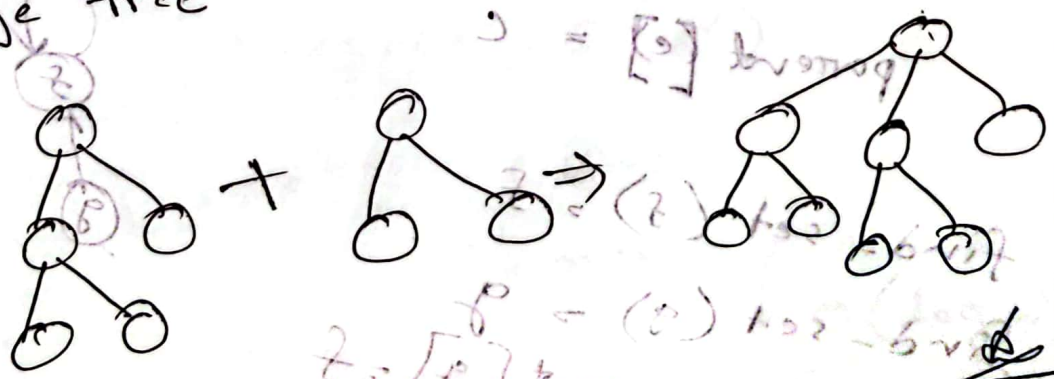


Union by rank :-

→ A set of large tree

→ A set of small tree

small tree will be the child of large tree



Disjoint - Set

Make_set (x)

P[x] ← x

Union (x, y)

a = Find_set (x)

b = Find_set (y)

P[a] = b

Find_set (x)

if x ≠ P[x]

return Find_set (P[x])

return P[x]

path compression

P[x] = Find_set (P[x])

P.T.O

Union by rank:

Make_set(x).

```
rank[x] ← 0;
```

Link(x, y)

* if rank[x] > rank[y]
  than p[y] ← x;
  else p[x] ← y;
* if rank[x] == rank[y]
  than rank[y]++;

```
union(x, y){
link( find_set(x), find_set(y))
};
```

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 (matric)

	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

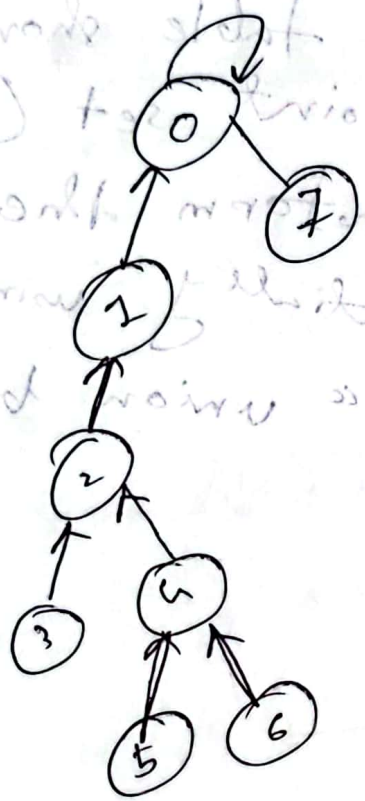
(I) Draw the disjoint set forest

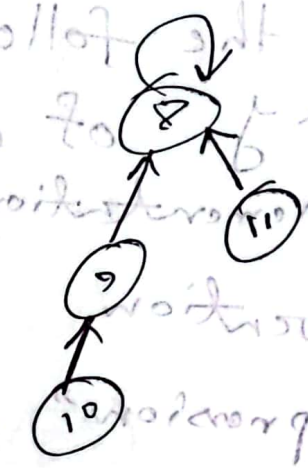
(II) ~~Reduce~~ Redraw the forest after union(3,7)

(III) " " " " (5,10)

(IV) what will be returned by find_set(9) ?

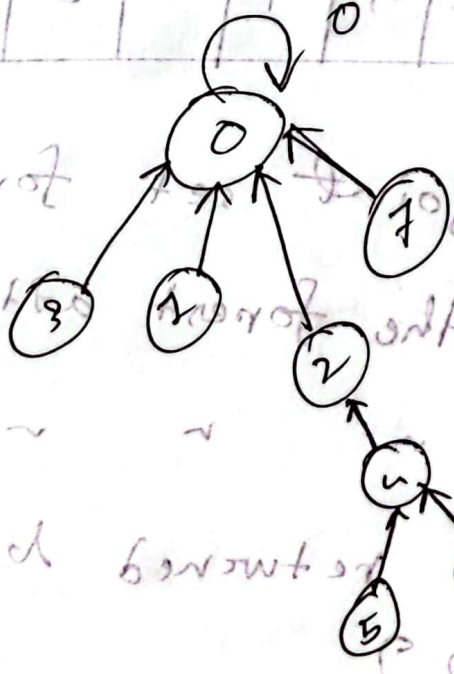
(i)





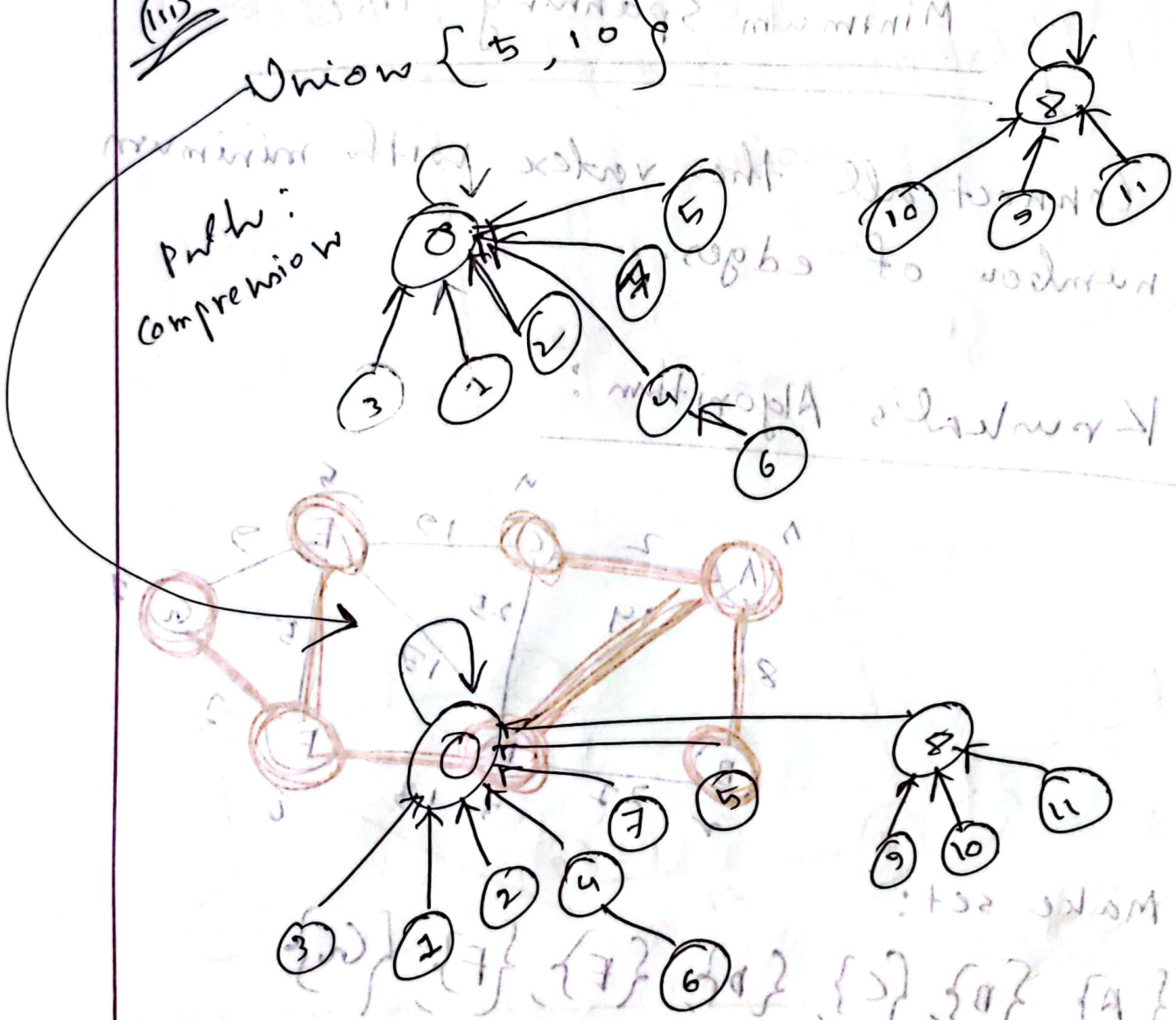
(ii)

Union {3, 7}



(iii)

Union {5, 10}

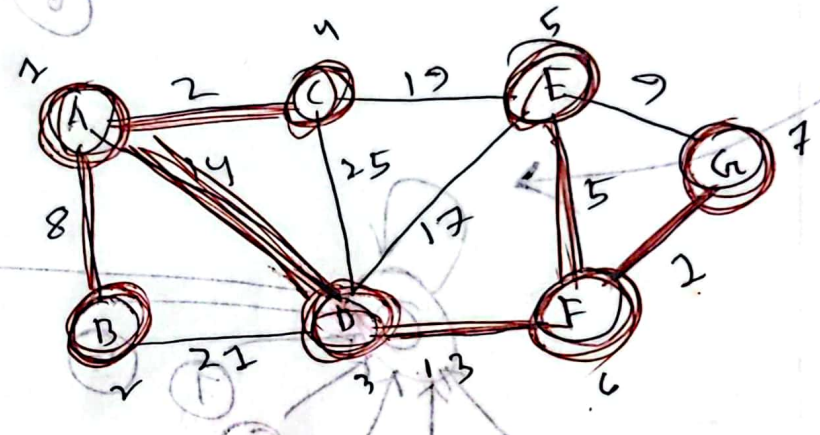


Find_set (5) = 0 ;

Minimum Spanning tree

Connect all the vertex with minimum number of edges.

Kruskal's Algorithm:



Make set:

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

Sort the edges in Ascending Order:

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

connection check

~~F~~

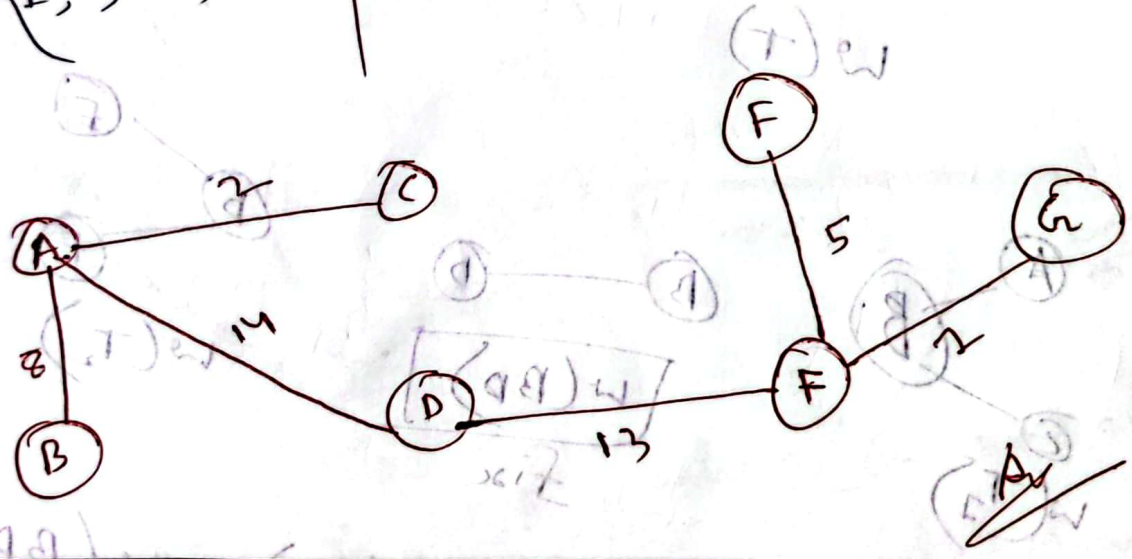
∴ {F, G}

{A, C}

{F, F}

{E, F, G}

```
if (not connected){
    union
}
else{
    dont;
}
```



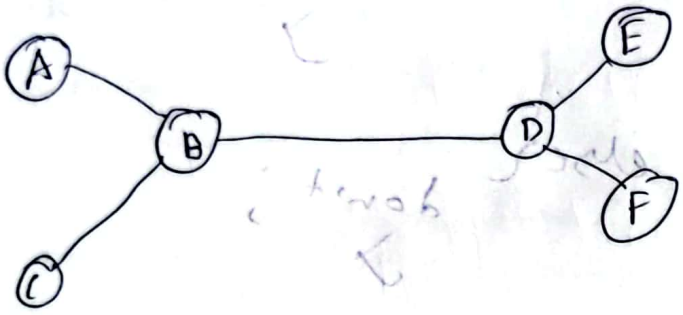
Next Tuesday

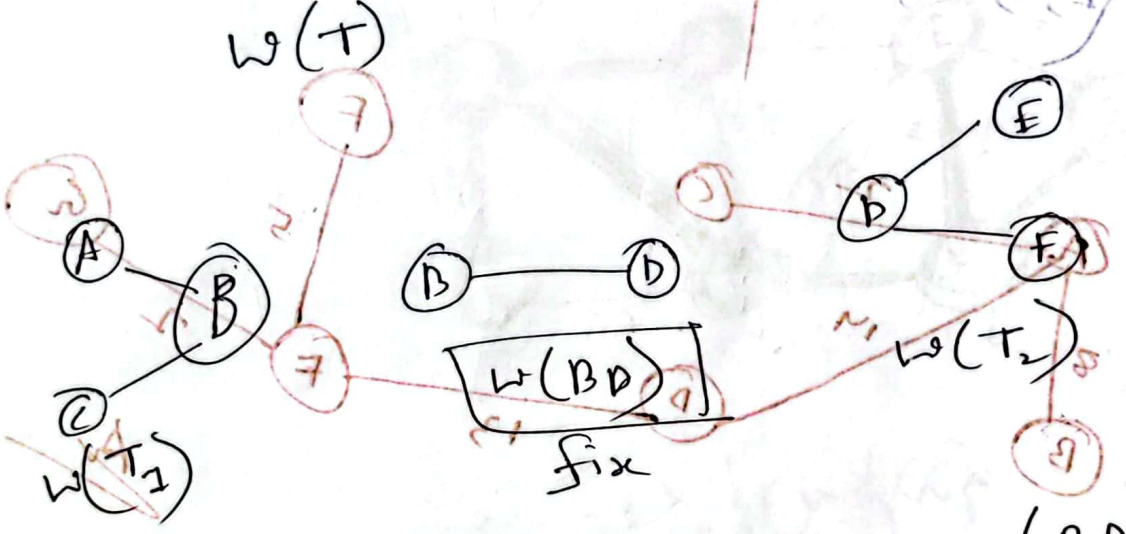
19.12.23

MST (Prims, kruskal)

Shortest Path (Dijkstra, Bellmanford)

Optimal Substructure in MST

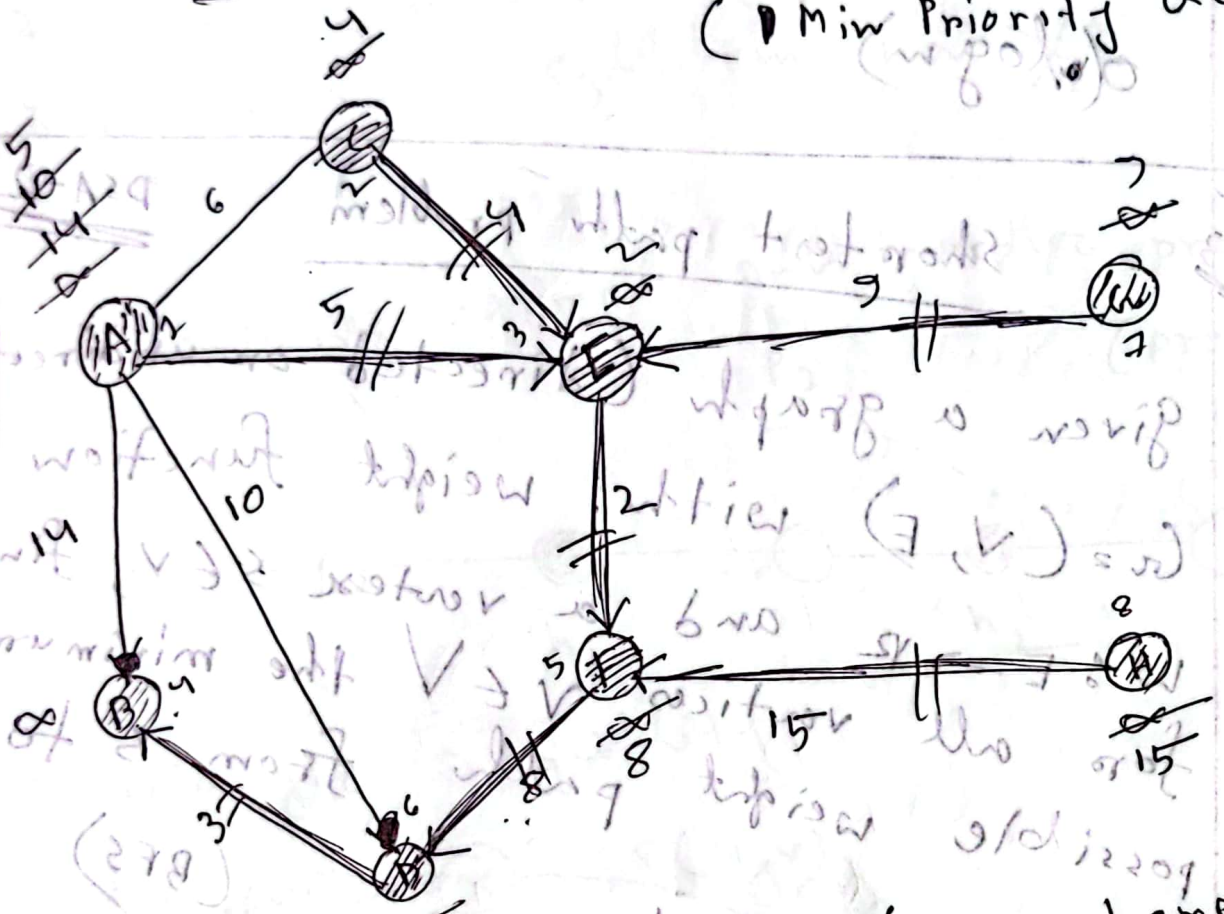




$$w(T) = w(T_1) + w(T_2) + w(BD)$$

So they are optimal.

Minimum Spanning Tree (Min Priority Queue)




```
MST_Prim (G, w, R)
Q = V[G]
for each v ∈ Q
    key[v] = ∞
key[R] = 0;
P[R] = NULL;
```

```
while (Q not empty)
U = ExtractMin(Q)
for each v ∈ Adj[U]
if (v ∈ Q && w(u,v) <
    key[v]){
    P[v] = u
key[v] = w[u,v];
```

Min Priority Queue

$O(\log n)$

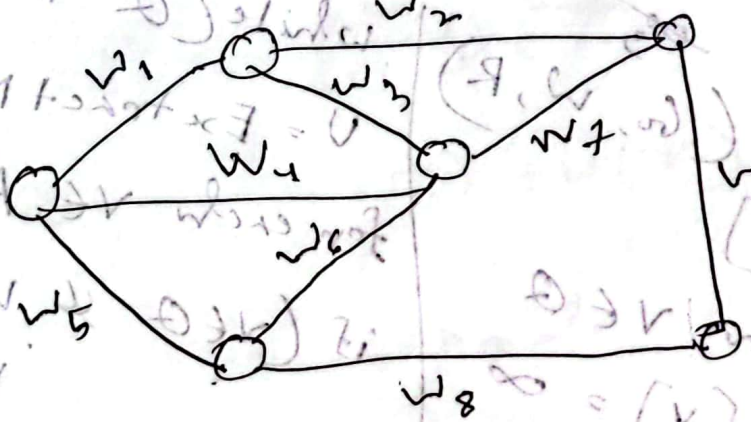
C.W
7-12-23

Shortest path problem

DSA-2

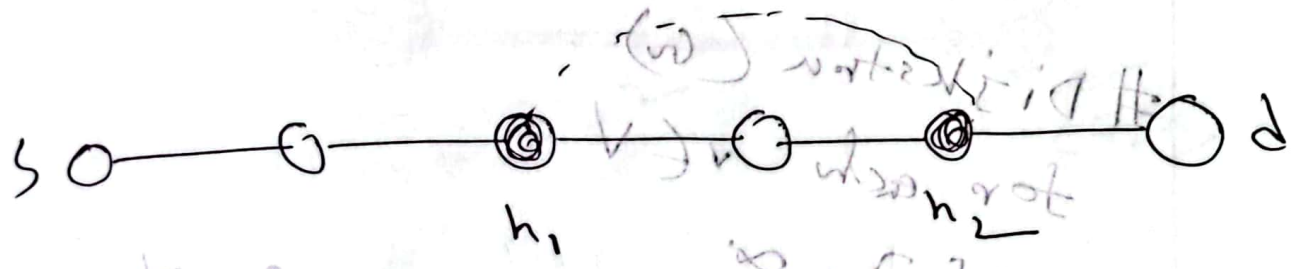
Given a graph (directed or undirected) $G = (V, E)$ with weight function $W : E \to R$ and a vertex $S \in V$, find for all vertices $V \in V$ the minimum possible weight path from S to V

(BFS)

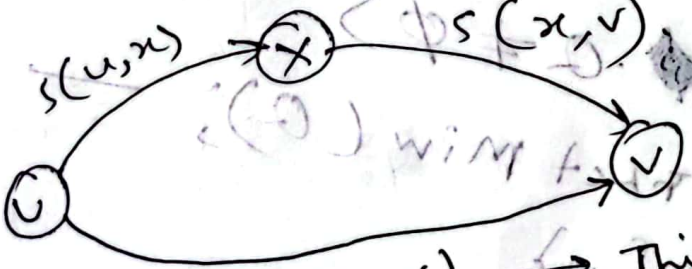


* simple source ✓
* single pair ✓
* All pairs → syllabus nai ✗

→ It follows optimal substructure property. (prove)



→ Triangle inequality.

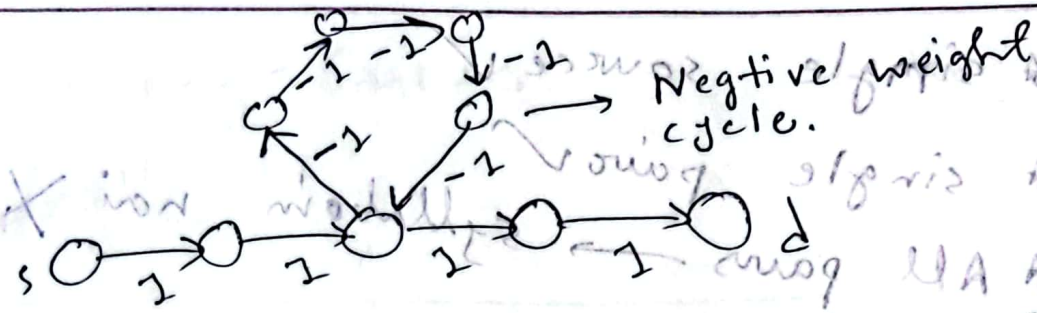


$s(u,v)$ → This the shortest path

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

Pr.o

*



→ Every graph can not give a shortest path result.

Dijkstra (G)

for each $v \in V$

$d[v] = \infty$,

$d[s] = 0$, $S = \phi$; $Q = V$

while ($Q \neq \phi$)

u = Extract Min (Q); → $\log V$

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

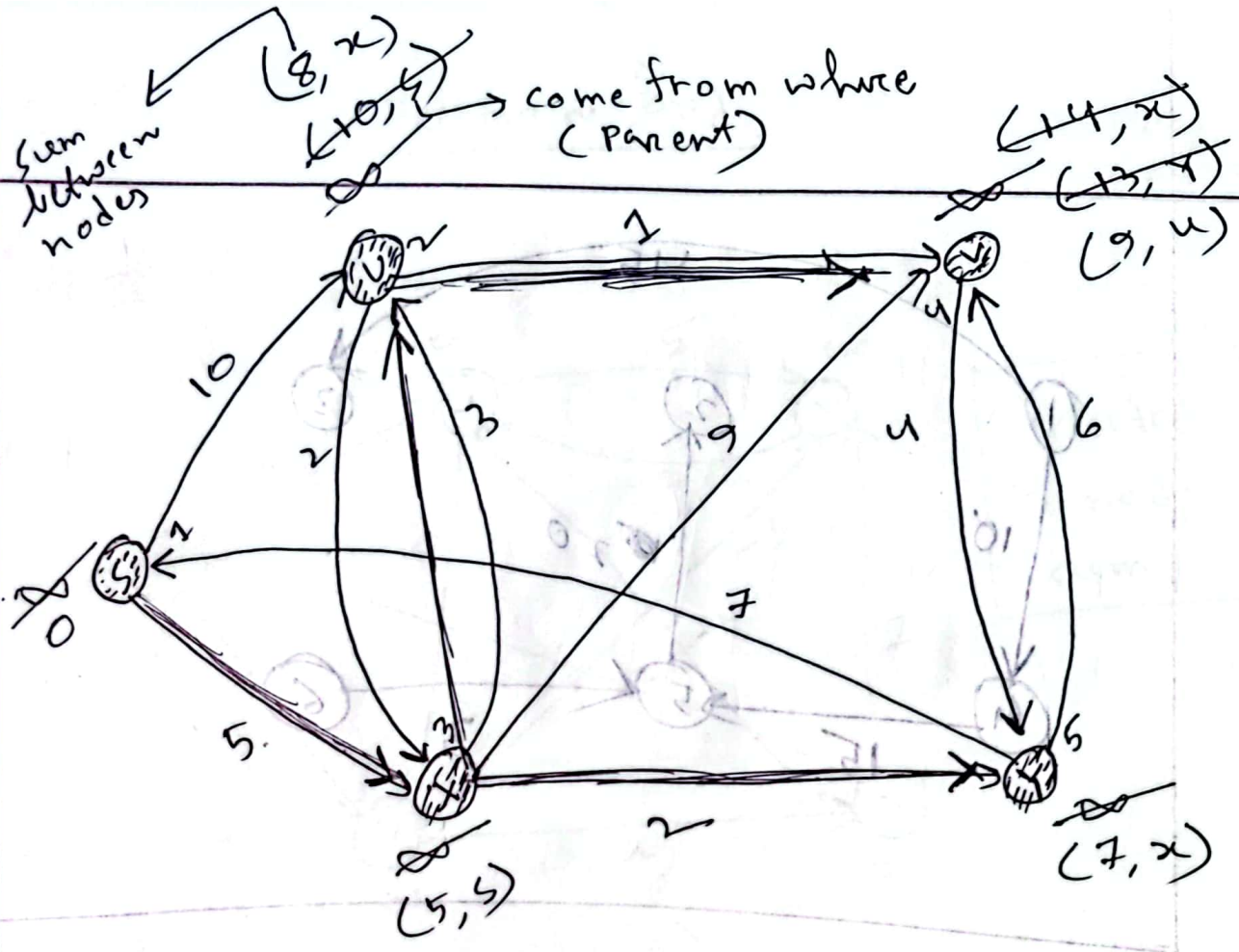
for each $v \in u \rightarrow$ Adj []

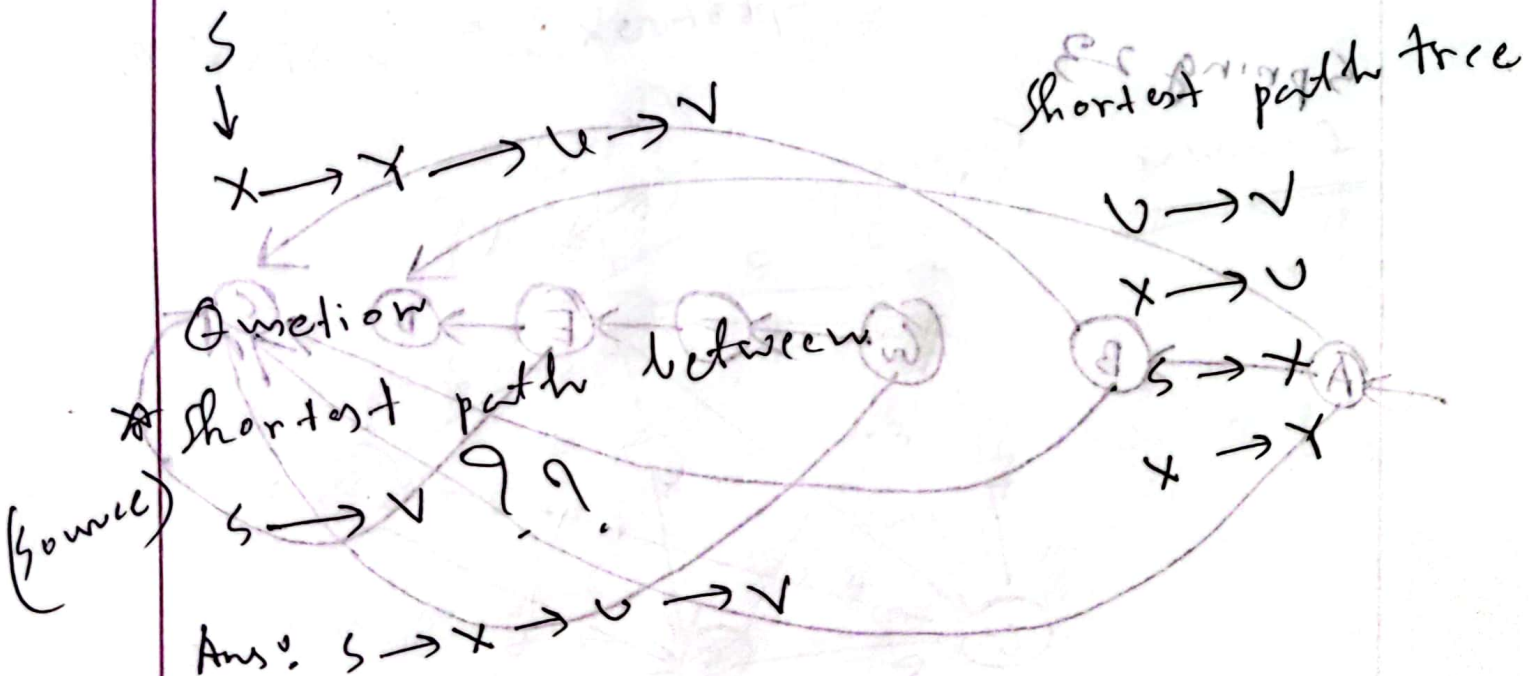
if ($d[v] > d[u] + w(u,v)$)

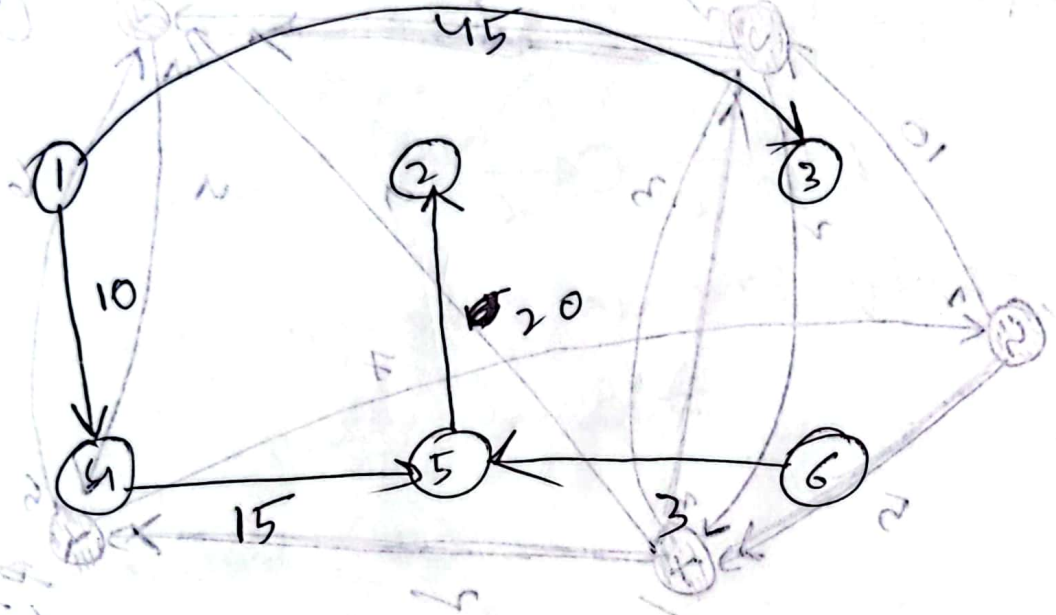
$d[v] = d[u] + w(u,v)$;

$P[v] = u$;

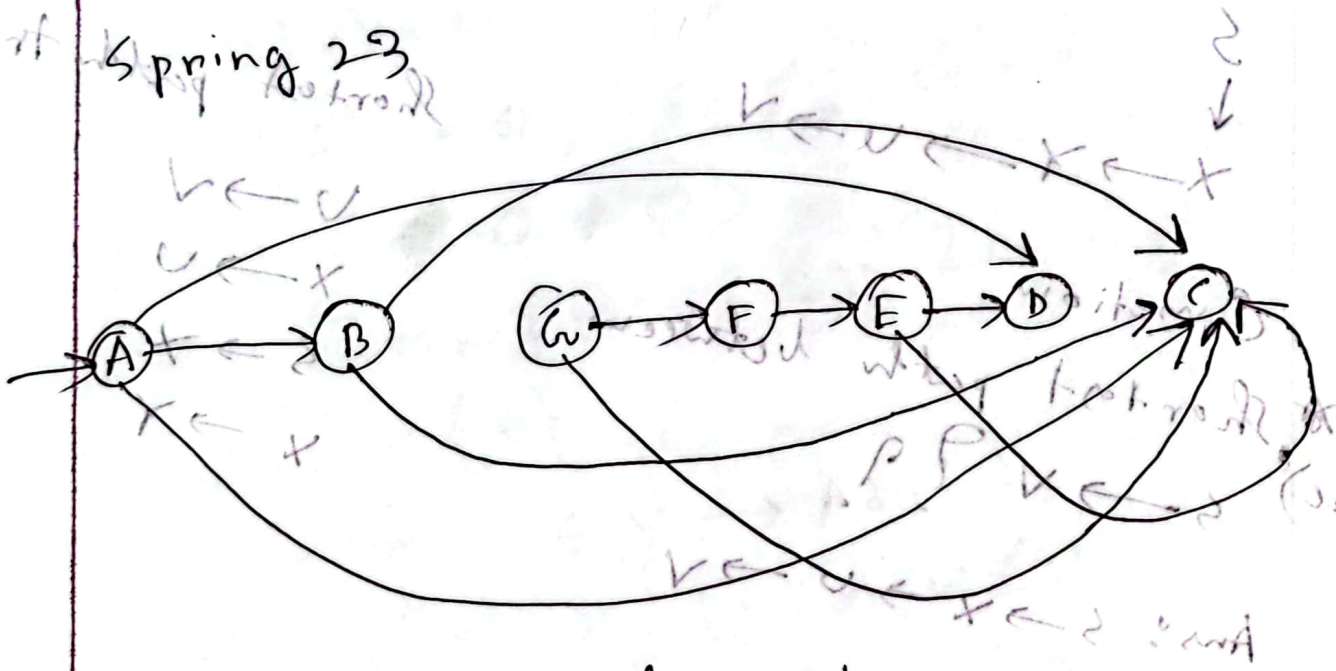
time complexity $O(E \log V)$


(8, x)
(10, z)
come from where
(Parent)
sum between nodes
(14, x)
(13, y)
(9, u)
1
10
2
3
9
4
6
7
0
5
3
11
2
5
(5, 5)
(7, x)


s
x → x → u → v
Question
Shortest path between
(source)
s → v ? ?
Ans: s → x → u → v
Shortest path tree
v → v
x → u
s → x
x → y



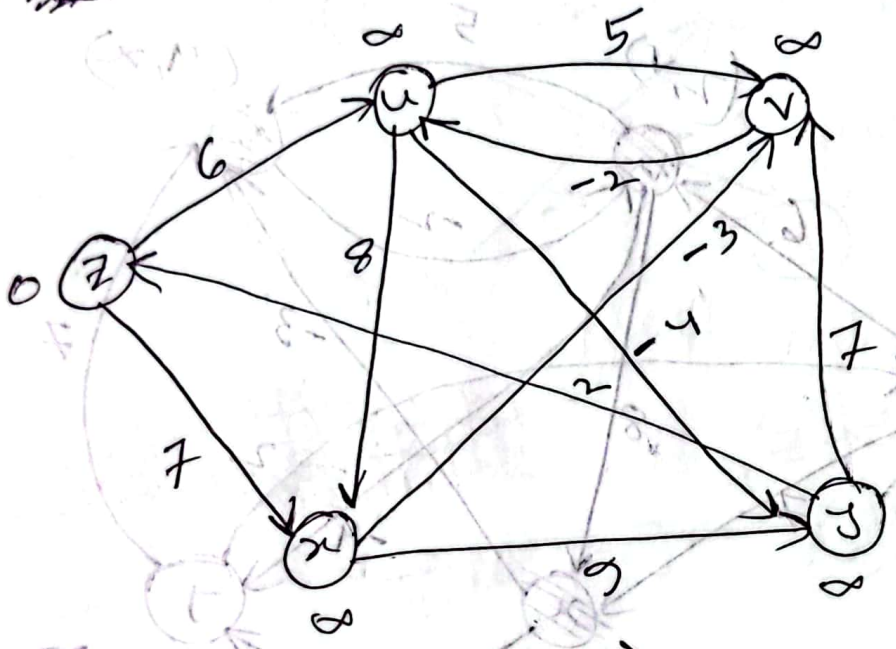
Spring 23



Topological sort

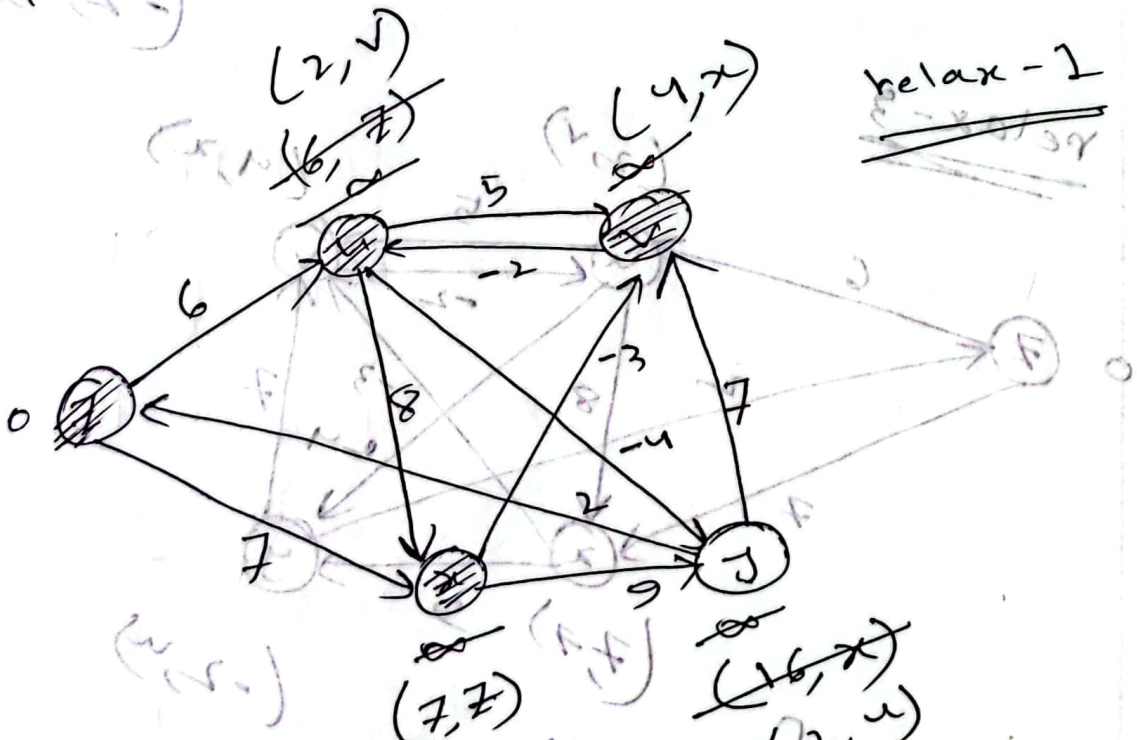
Dependency graph

Bellman ford

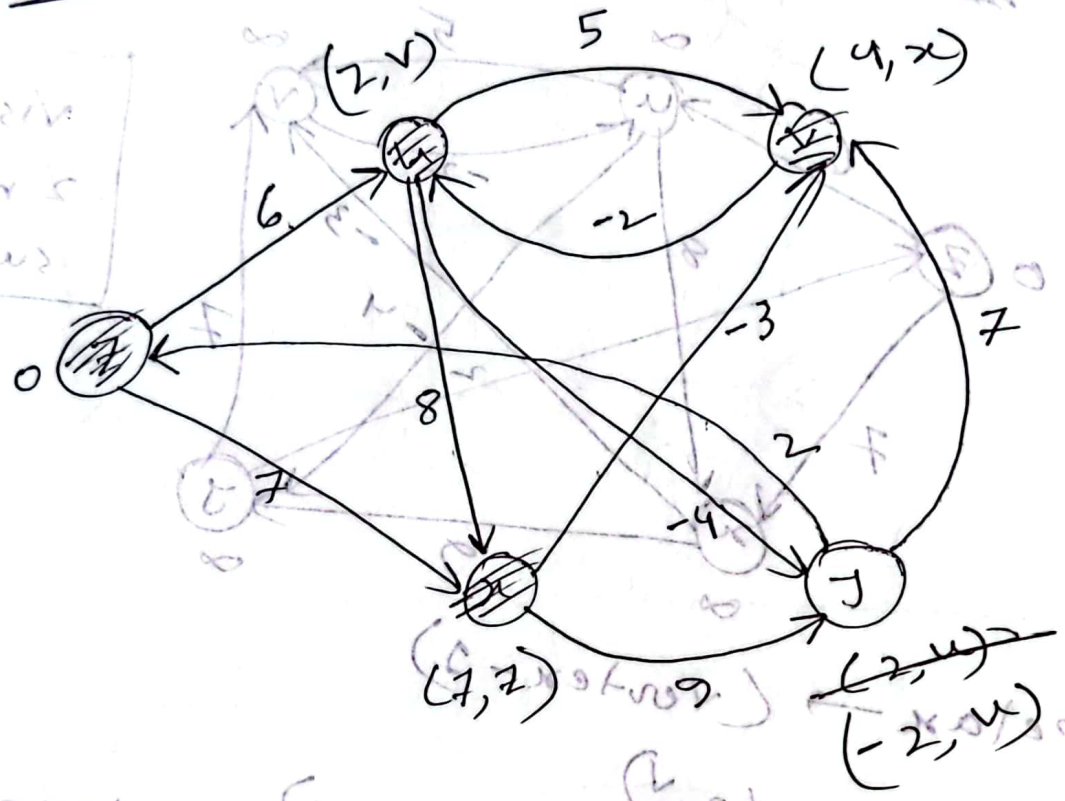


visited
2 nodes
sum

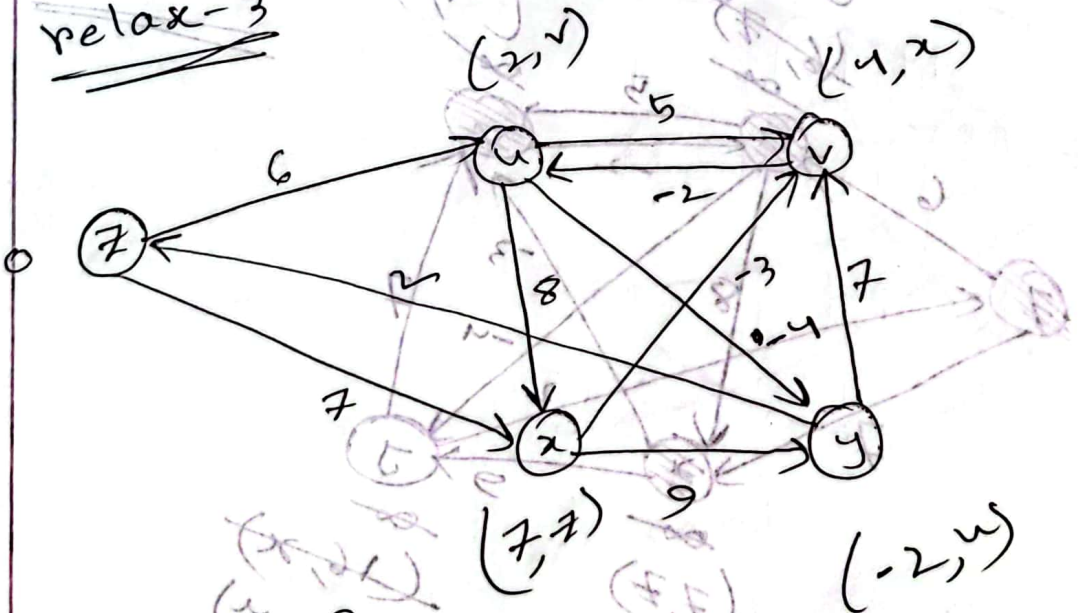
* relax → (vertex - 1)



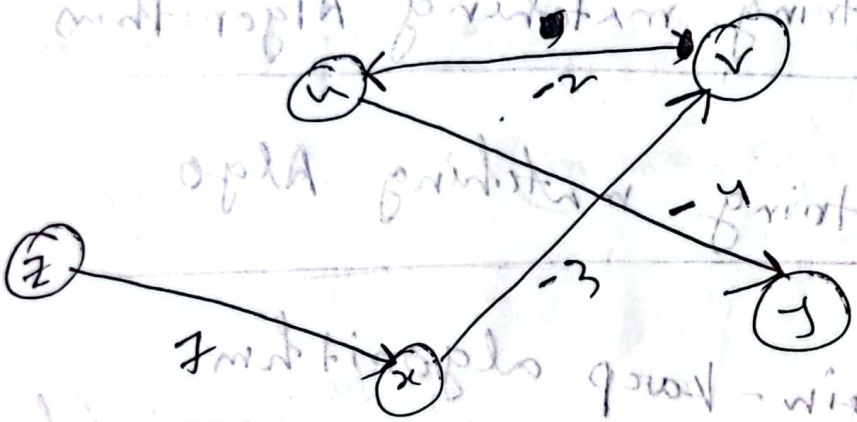
relax-2



relax-3



relax ~~এটি~~ করা (V-1) target এর মধ্যে update হয়ে যায় তাহলে আর update হবে না।



$z \to x \to v \to y \to J$

$(z \to J)$

String matching Algorithms

Naive string matching Algo

Rabin-Karp algorithm (hashing)

Formula

t_1

a b c a b d e a

t_0

$t_0, t_1, t_2, t_3 \ldots$

t_1

T = 4 3 1 4 1 5 9 2 ...

t_0

$t_0 = 4314$

$\{(4314 - 4000) \times 10\} + 2$

$= 3140 + 2$

$= 3142$

$t_1 = 3142$

(rolling hash)

* $(a+m) \bmod q$

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

* $(am) \bmod q$

$= \left((a \bmod q) \left(m \bmod q \right) \right) \bmod q$

T = "daeiahrea"

P = "aei"

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

$= (100 + 50 + 9)$

$= 159 \bmod 13$

$= 3$

$t_0 = dae = 415 \bmod 13$

$= 12$

$t_1 = aei = 10(12 - d \times 10^2) + i$

$= -3872 \bmod 13 = 3$

a = 1
b = 2
c = 3
d = 4
e = 5
f = 6
g = 7
h = 8
i = 9

13) 159 (12
13
29
26
3

$= e^{i\omega}$

t_2 = {10 (t_1 এর result − t_2 এর আগের Digit এর Value $\times 10^{y}$)} +

t_2 এর Last digit এর Value

$= \{10(3-100)\} + 2$

$= -969 \mod 13$

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

$= 6 \mod 13$

$t_2 = 591 \% 13$

$= 6$

$969 \div 13$

$= 75$ ✓

13) 591 (45
52
71
65
6

* $q = 13, \quad d = 10$

$P = 159 \% 13$

$= 3$

13) 159 (12
13
29
26
3

String matching

Rabin - Karp

T = "daeiahea"

T_1 T_2 T_3 T_4 T_5 T_6

P = "hea"

a = 1
b = 2
c = 3
d = 4
e = 5
f = 6
g = 7
h = 8
i = 9

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

$= 851 = 6$

$h(T_0) = dae$

$= 4 \times 10^2 + 1 \times 10^1 + 5 \times 10^0$

$= 415 \% 13 = 12$

$h(T_1) = aei$

$= \{(415 - 4 \times 10^2) \times 10\} + 9 \times 10^0$

$= 159 \% 13 = 3$

daeiahea

$h(T_2)$ = eia

$= \{(159 - 1\times10^{2})\times10\} + 1$

$= 591 \% 13 = 6$ (Spurious hit)

$h(T_3)$ = iah

$= \{(591 - 5\times10^{2})\times10\} + 8$

$= 918 \% 13 = 8$

$h(T_4)$ = ahe

$= \{(918 - 9\times10^{2})\times10\} + 5$

$= 185 \% 13 = 3$

$h(T_5)$ = hea

$= \{(185 - 1\times10^{2})\times10\} + 1$

$= 851 \% 13 = 6$ [match found]

$h(P) = h(T_5)$ (pattern found)

a = 1
b = 2
c = 3
d = 4
e = 5
f = 6
g = 7
h = 8
i = 9

13) 415 (31
39
25
13
12

13) 851 (65
78
71
65
6

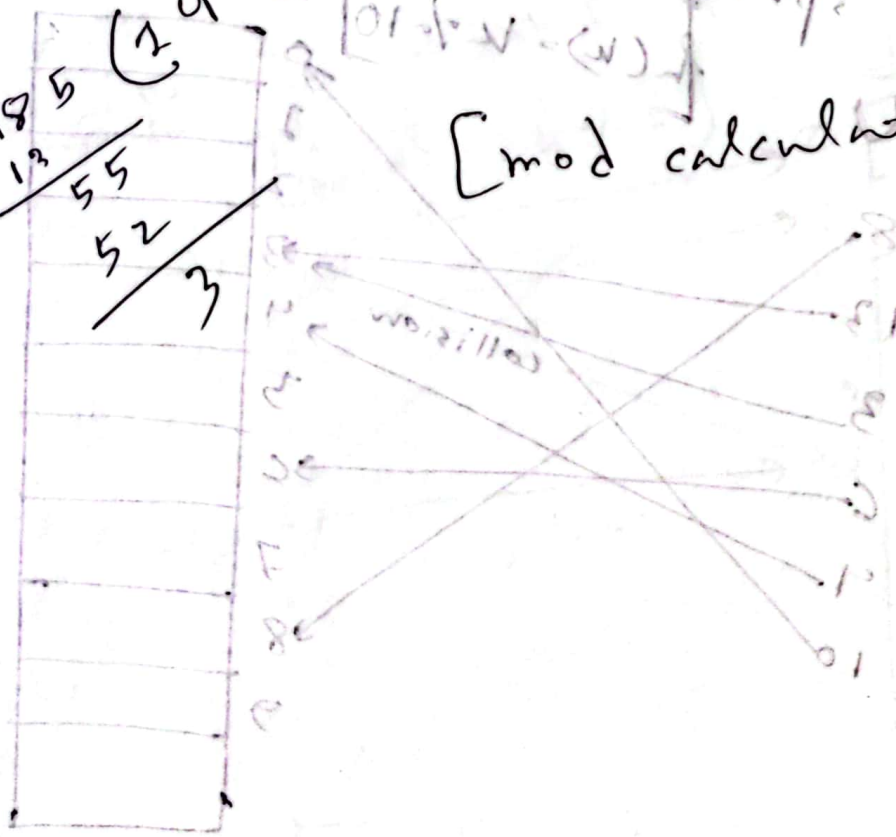
13) 159 (12
13
29
26
3

13) 591 (45
52
72
65
6

13) 918 (70
910
8

13) 185 (14
13
55
52
3

[mod calculation]



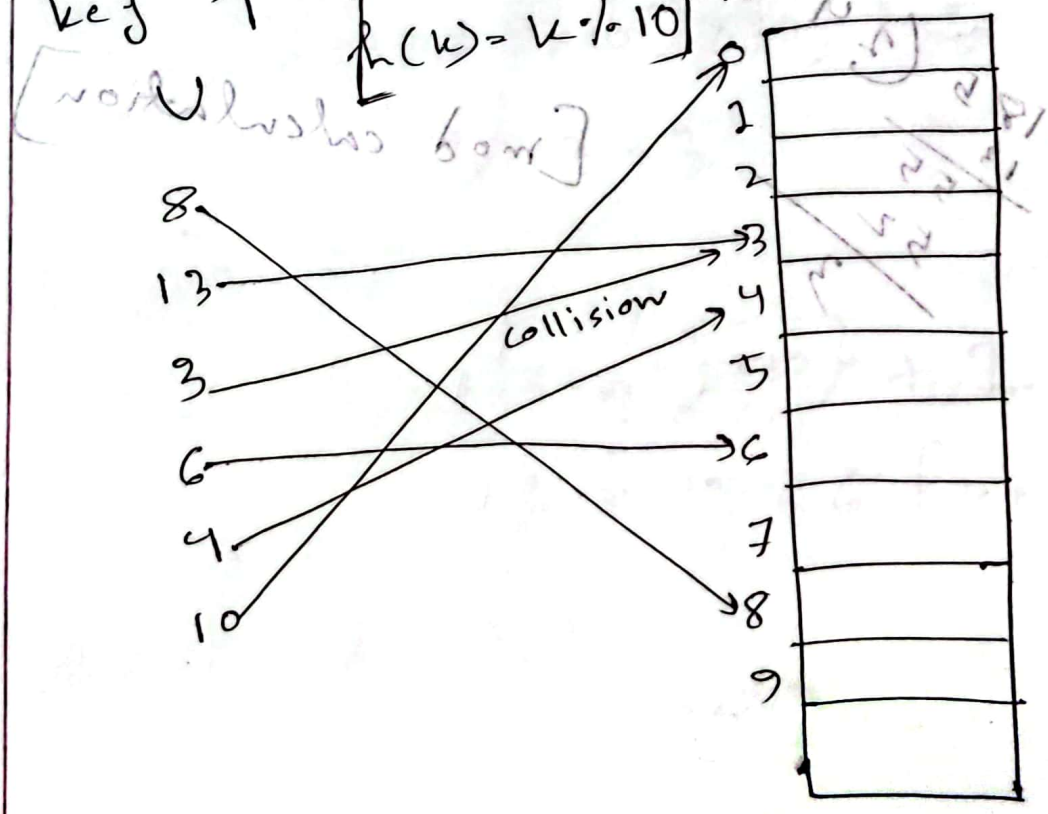
Hash tables

key → index

$h(k) = k$, [ideal hashing]

hashing:

Mapping from U to the slots of a hash table T[0 ---- m-1]

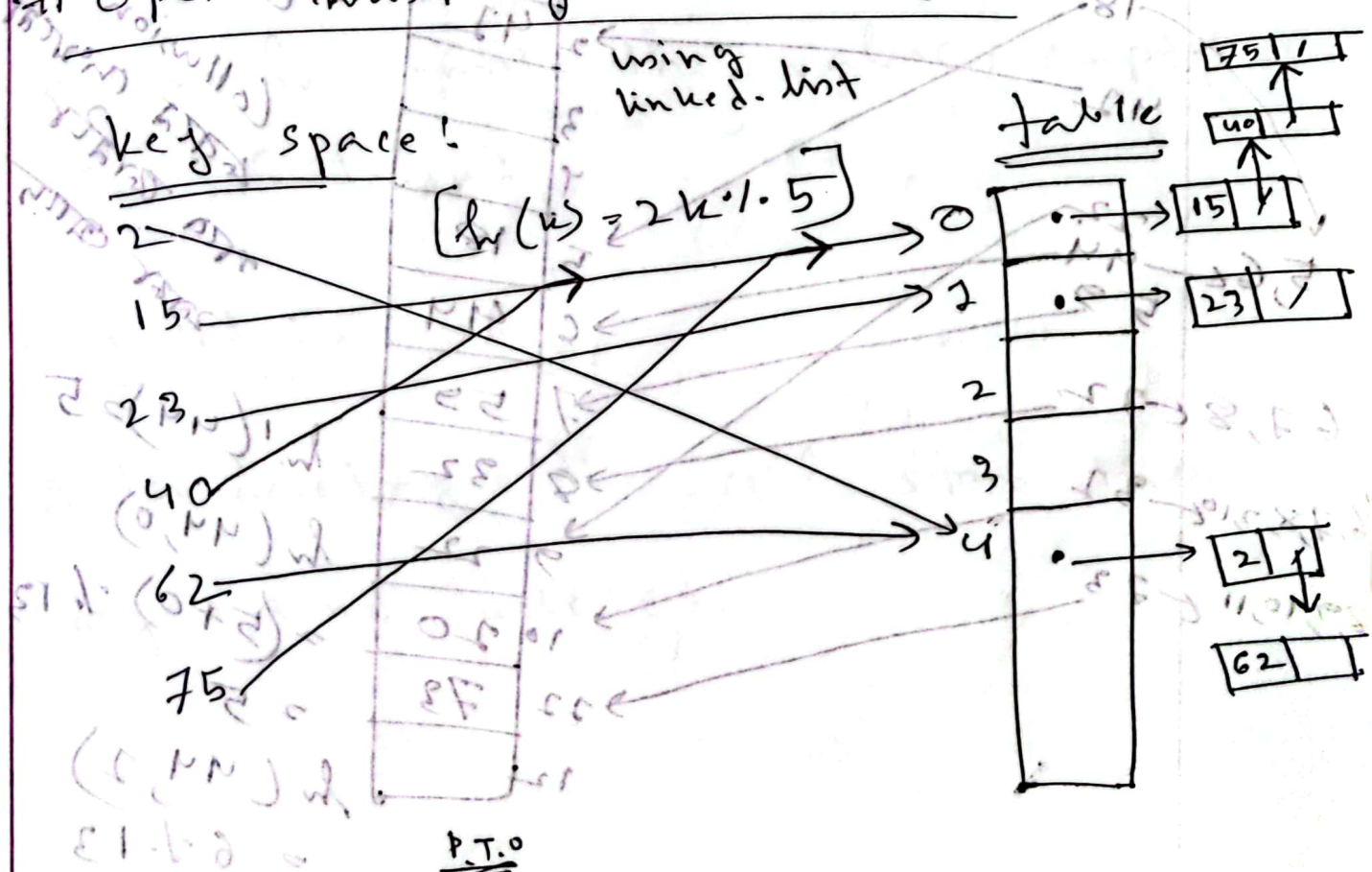


Avoiding collision:

1. Open ~~addressing~~ hashing
   → chaining

2. closed - hasing / open addressing
   → linear probing
   → Quadratic probing
   → double hashing

Open - hashing (chaining)



P.T.O

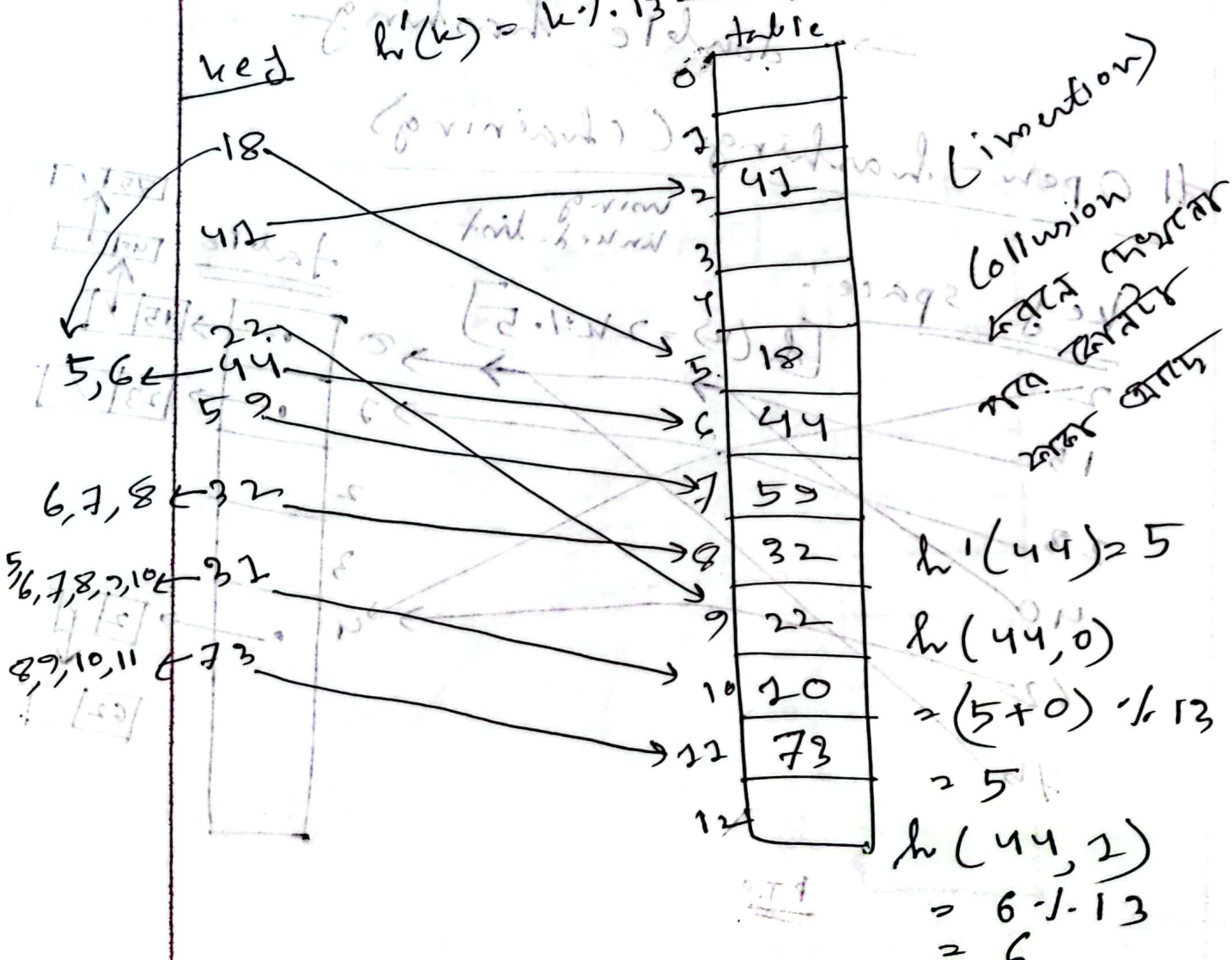
* storing all the elements are guaranteed.
* waste of space

closed hashing.

① linear probing

$h(k) = (h'(k) + i) \% 13$

$h'(k) = k \% 13$



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

$= 8$

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

$= 8$

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

$= 9$

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

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

→ search untill the value / empty cell is found;

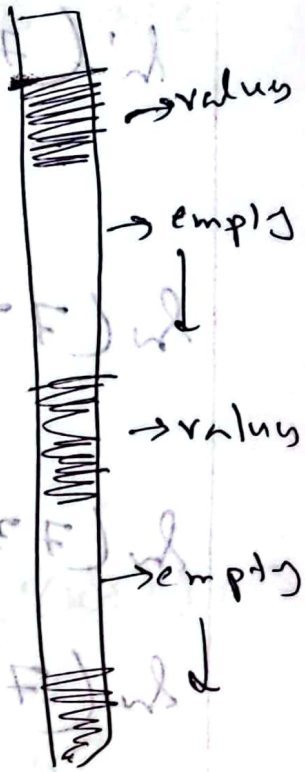
Delete:

After deleting the cell store their tag name deleted.

সমস্যা আছে.

→ Primary clustering.

কোন ফাকা থাকে যাবে না



Quadratic probing

$h(k)$

$(h'(k) + i^2) \bmod m$

0, 1, 2, 3 ...

0, 1, 4, 9 ...

problem

→ secondary clustering.

P.T.

Double hashing

$$h(k) = (h'(k) + i\,h''(k)) \bmod m$$

k, d

$h'(x) = h'(y)$ } ×

$h''(x) = h''(y)$

$i = 0, 1, 2, 3, \ldots$

NP Completeness

P (Polynomial) problems:

P problems refer to problems where an algorithm would take a polynomial amount of time to solve, or 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-time would not be polynomial it would be exponential (2^n, n^n, $n!$ etc).

However, this class of problems can be given a specific solution and checking the solution would have a polynomial run-time.

Therefore, NP class problem don't have a polynomial run-time to solve, but have a polynomial runtime to verify solution!

Reduction:

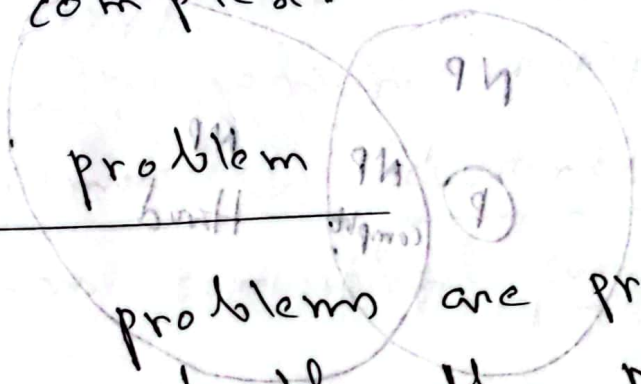
Suppose, we have two problems, A & B and we know problem B is a P class problem. If problem A can be reduced, or converted, to problem B, and this reduction takes a polynomial amount of time, then we can say that A is also P class problem. (A is reducible to B)

NP hard problem:

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

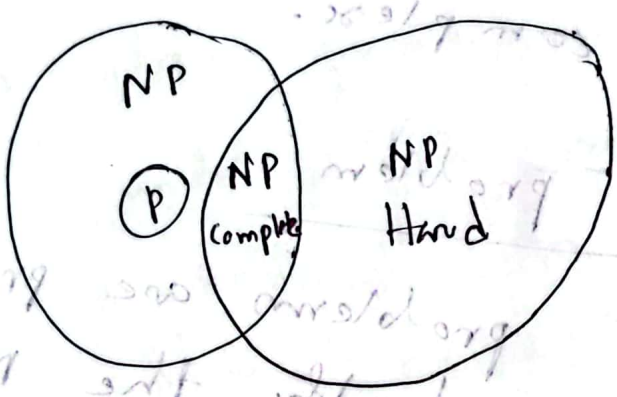
then we can say, this problem is at least as hard as any NP problem but it could be much harder or more complex.

NP-complete problem



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

* Abdul Bari



P is the subset of NP.

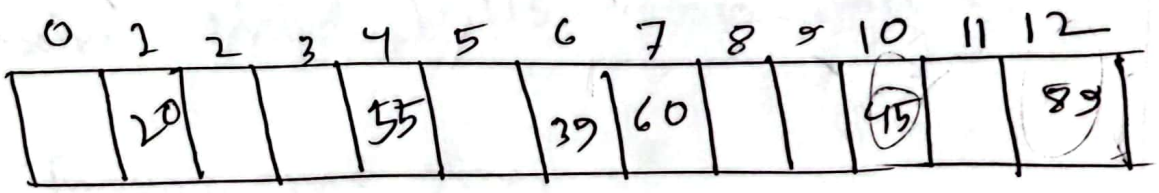
prove that tomorrow the Non deterministic algorithm converted into polynomial time deterministic algorithm.

$P \subseteq NP$

$P = NP$ (Prove)

(a)

k	$h_1'(k)$	$h_2'(k)$	$h(i,k)$
89	12	4	(0, 12) ✓
55	1	3	(0, 1) ✗ (1, 4) ✓



Spring-22

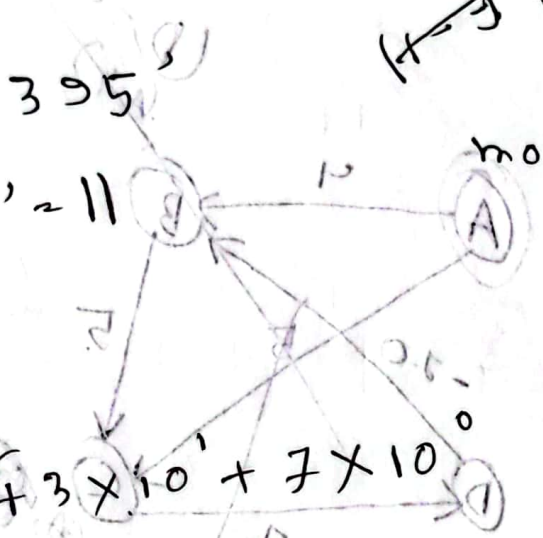
1)

$T = '237395'$

$P = '739' = 11$

mod = 13

d = 10



$T_0 = 237$

$= 2 \times 10^2 + 3 \times 10^1 + 7 \times 10^0$

$= 237 = 3$

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

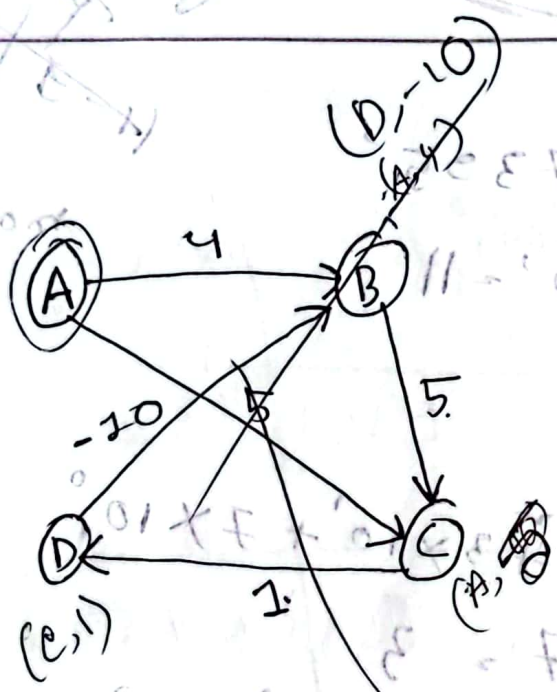
$= 373 = 9$

$T_2 = \{(373 - 3 \times 10^2) \times 10\} + 9$

$= 739 = 11$ [match found]

$T_3 = \{(739 - 7 \times 10^2) \times 10\} + 5$

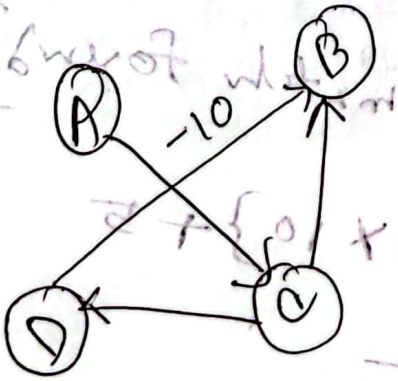
$= 395 = 5$



$(5+1-10) + (5+1-10)$

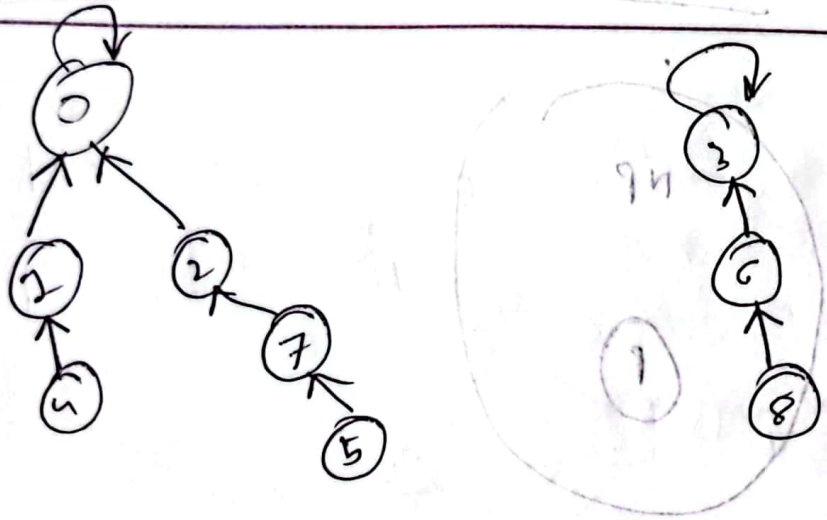
$= -4 - 4 = -8$

(Shortest path পাওয়া যাবে না)

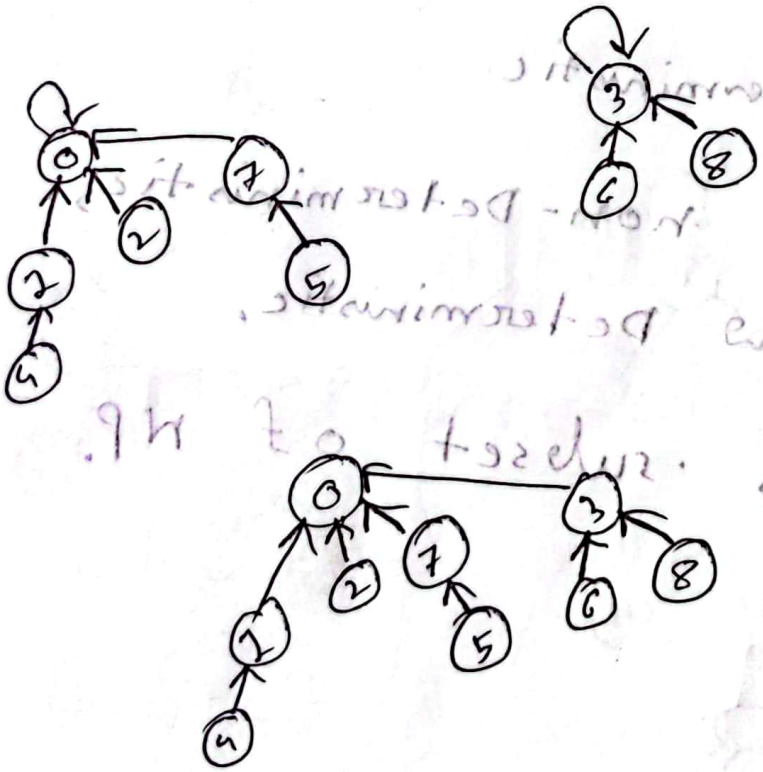


union by rank

path compression

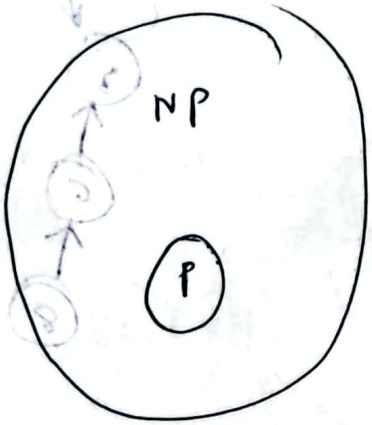


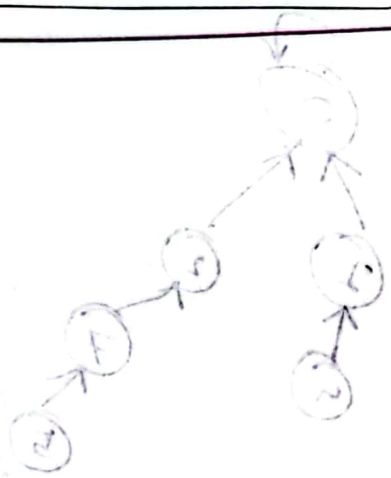
union(7,8)



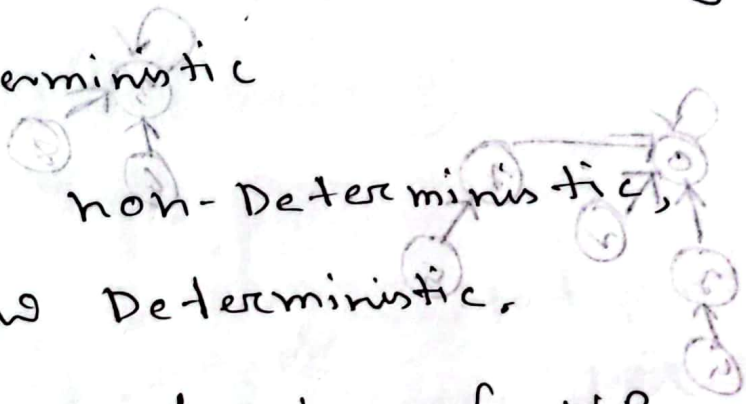
NP- Hard & NP complete

#



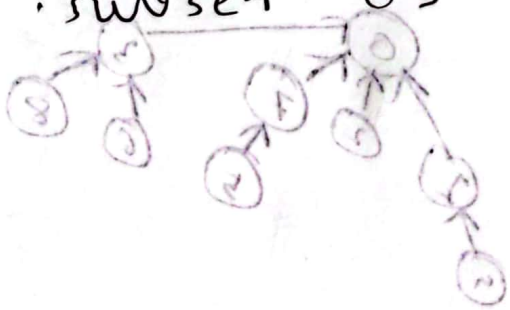


* Today Deterministic, yesterday it was how Deterministic



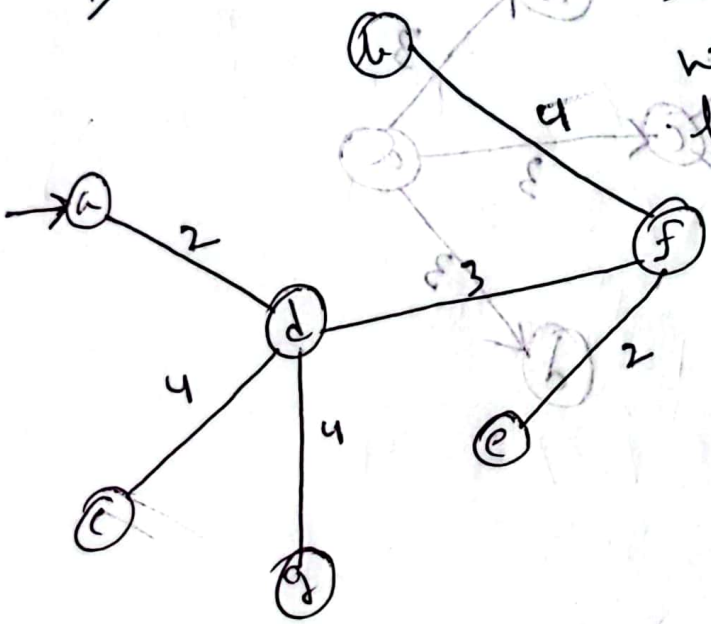
* today non-Deterministic, tomorrow Deterministic.

* P is subset of NP.



Sum-23

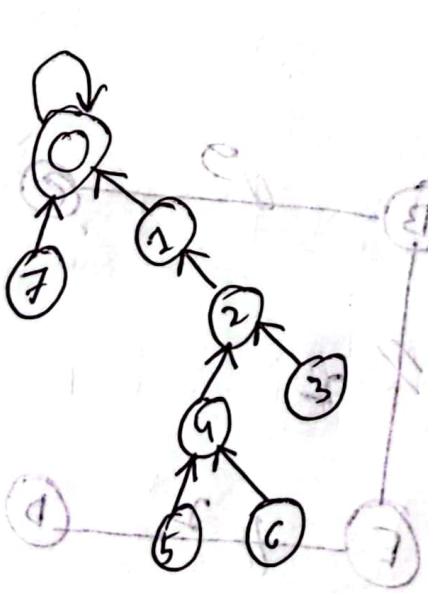
1(a) (4)

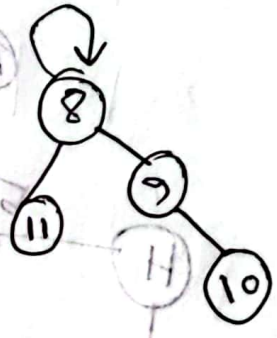


if we check the sum of the all weight than it will be 19 which is minimum.

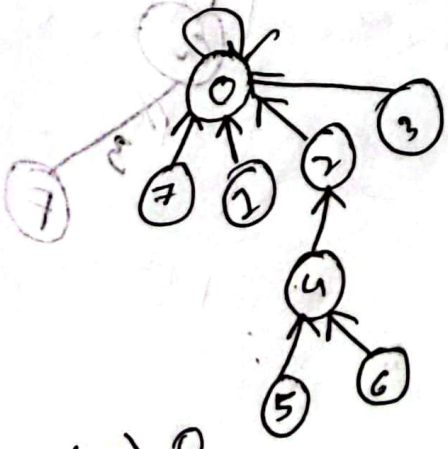
4(a)

(I)

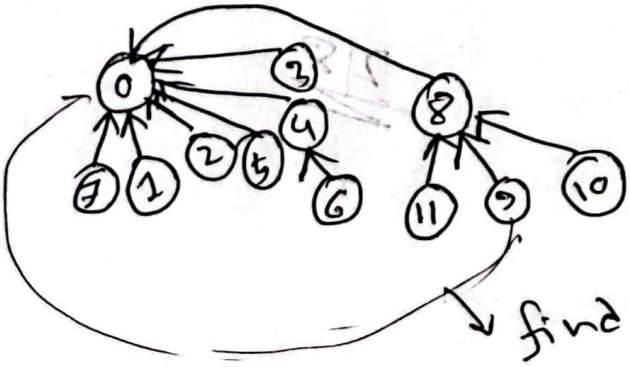




(II) Union (3,7)



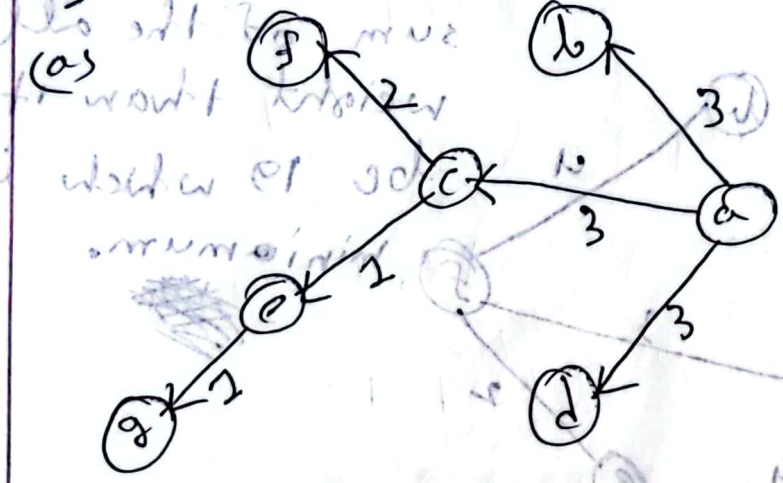
(III) Union (5,10)

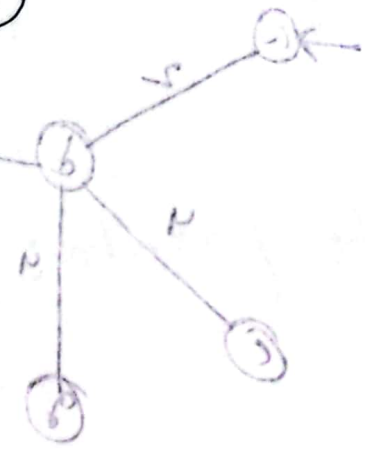


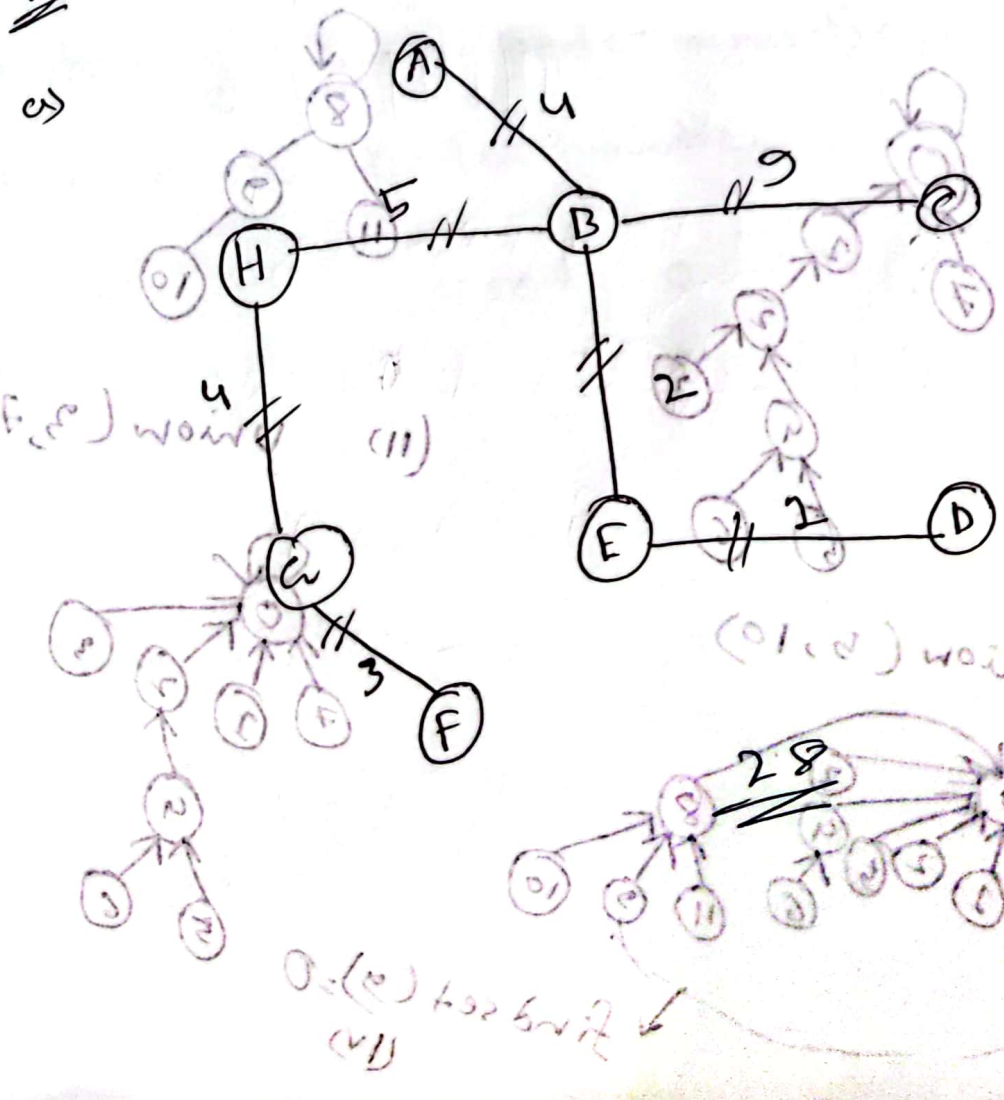
find set (9) = 0

(IV)

Sp-23







E - D - 2 ✓

B - E - 2 ✓

G - F - 3 ✓

H - G - 4 ✓

A - B - 4 ✓

H - B - 5 ✓

F - E - 8 X

B - C - 9 ✓

H - A - 12 X

2

(c) A = 1, C = 2, G = 3, T = 4

AGCTAGCGCAGCTAG P = AGCTAG

d = 4

P = AGCTAG

= (2×4^5) + (3×4^4) + (2×4^3) + (4×4^2) + (1×4^1) + (3×4^0)

= 1991.7 = 3

~~AWCTAWQQAWCTAW~~ AWCTAW

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

d = 4

mod = 7

t_0 = AWCTAW

= 1991 %. 7 = 3 [match found]

h = 6

t_1 = WCTAWC

= {(1991 − 1~~x45~~ 1024) × 4} + 2

= ~~3961~~

= 3870 %. 7 = 6

t_2 = CTAWCW

= ~~(3870 − 3 × 4^5) × 4} + 3~~

= {(3870 − 3 × 4^5) × 4} + 3

= 3195 %. 7 = ~~3~~ 3 [Spurious hit]

t_3 = TAWCWA (4 1 3 2 3 1)

= {(3195 − 2 × 4^5) × 4} + 1

= 4589 %. 7 = 4

$t_4 = AGCGA$

$= \{(4589 - 4 \times 4^5) \times 4\} + 3 \% 7$

$= 1975 \% 7 = 2$

$t_5 = GCAG$

$= \{(1975 - 1 \times 4^5) \times 4\} + 2 \% 7$

$= 3806 \% 7 = 5$

$t_6 = CAGT$

$= \{(3806 - 3 \times 4^5) \times 4\} + 4 \% 7$

$= 2940 \% 7 = 0$

$t_7 = AGTA$

$= \{(2940 - 2 \times 4^5) \times 4\} + 1 \% 7$

$= 3569 \% 7 = 6$

t_8 %. ALGORITHM

$= \{(3569 - 3 \times 4^5) \times 4\} + 3$ %. 7

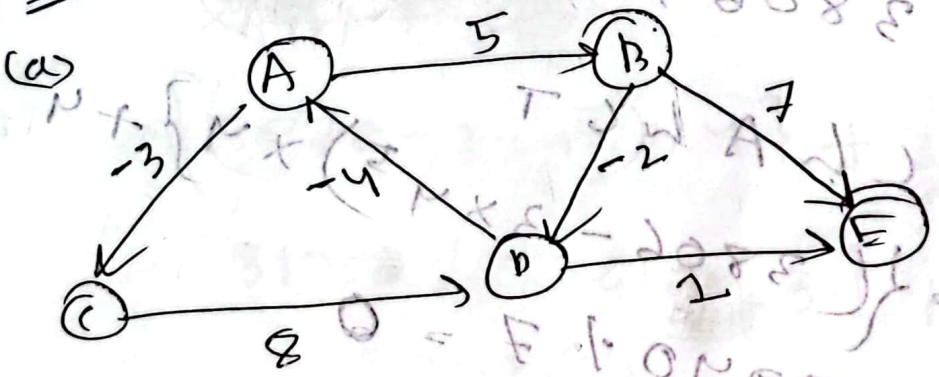
$= 1991$ %. 7 %. 2 3 [match found

number of occurrence = 2

spurious hit = 7

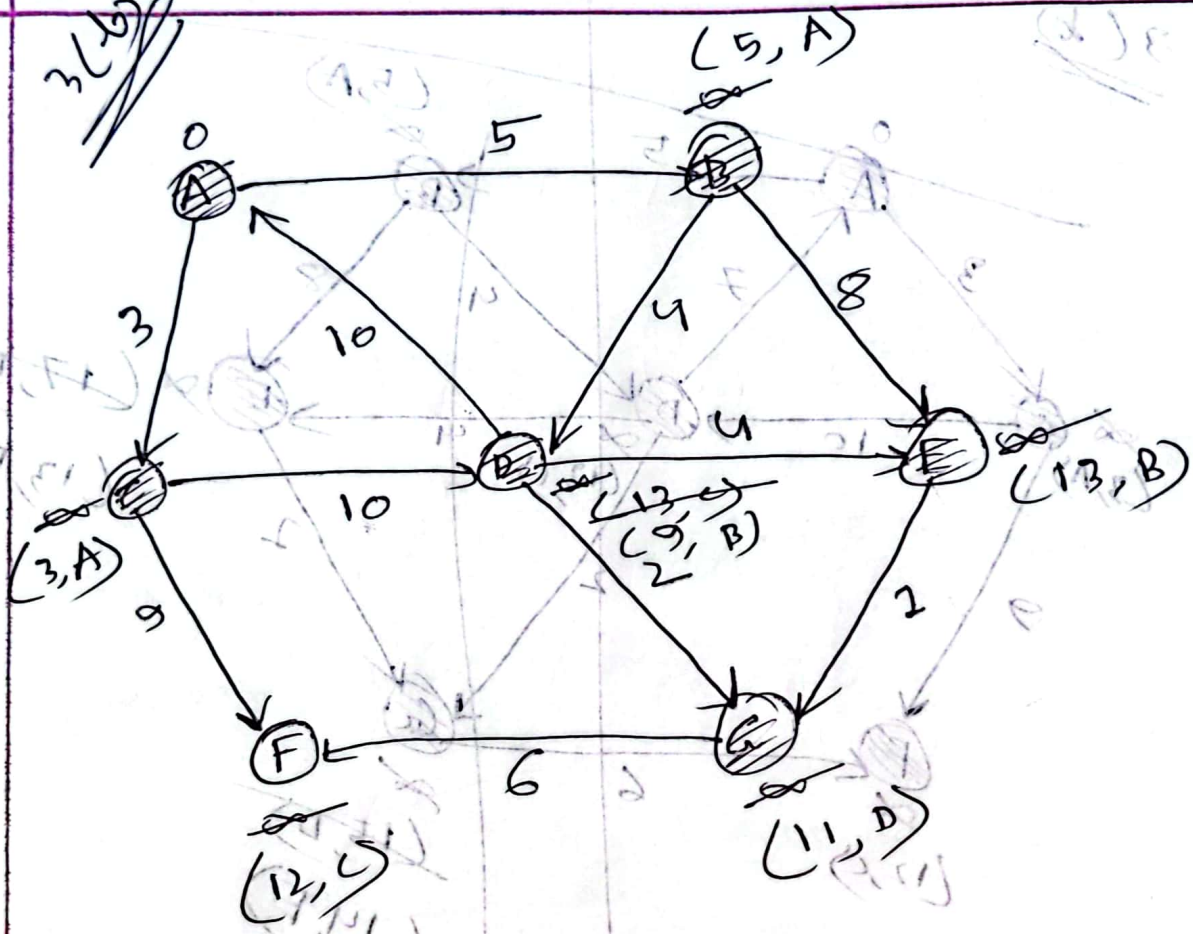
3

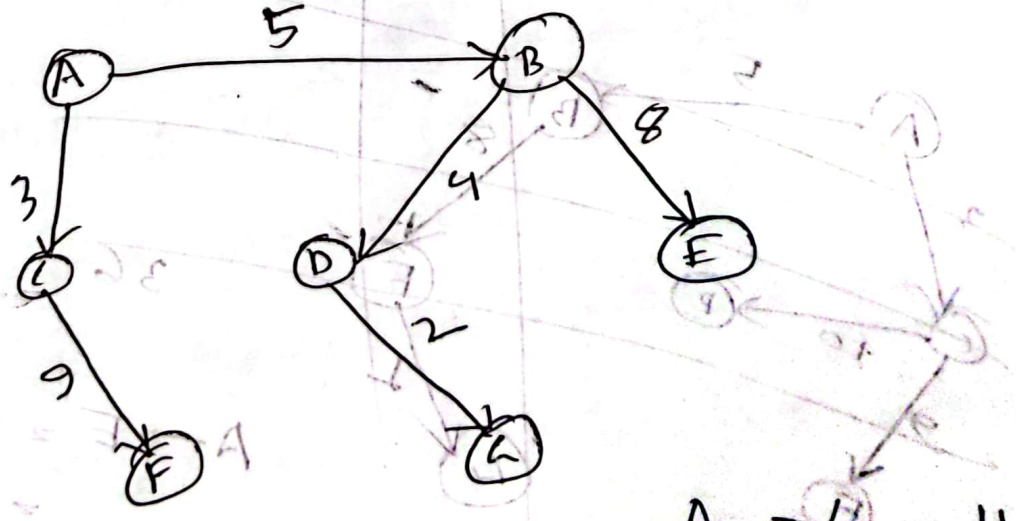
(a)



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

3(b)



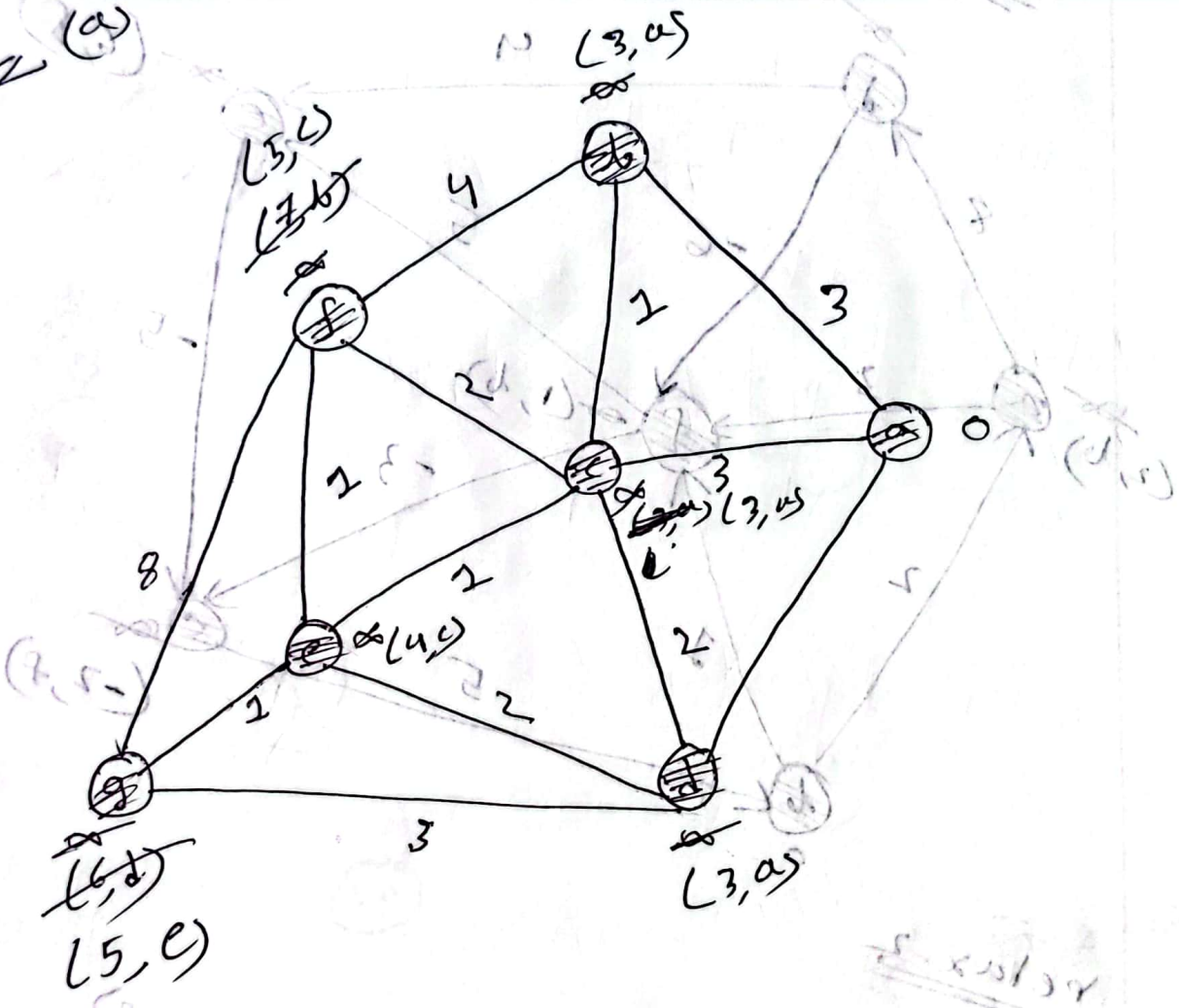


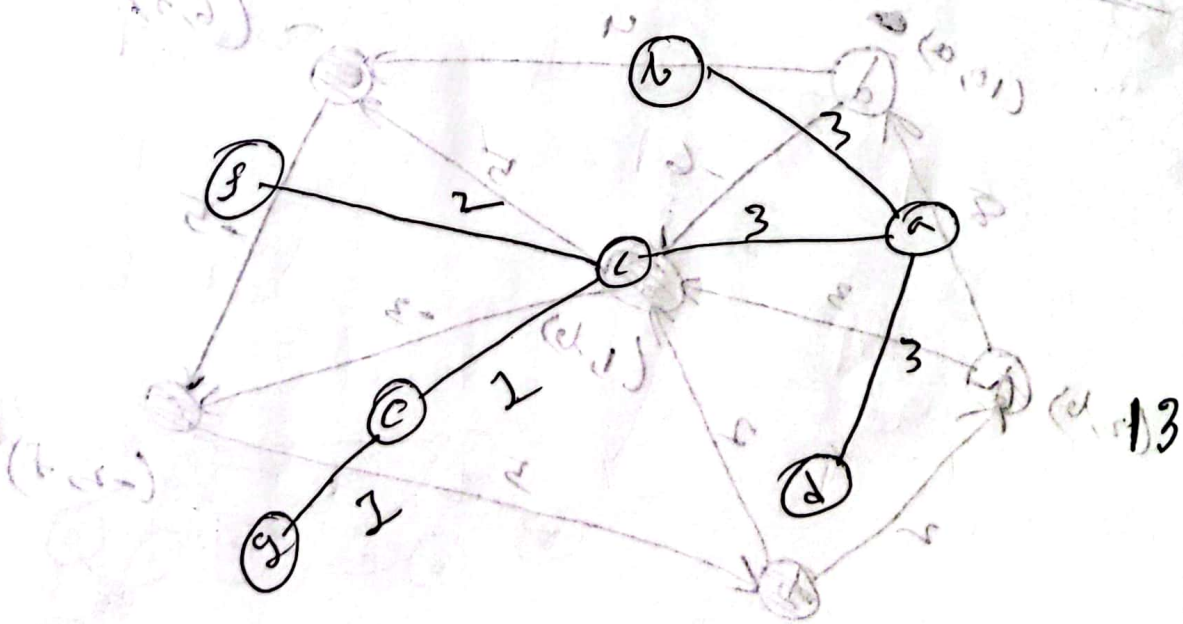
A → H = 11

∴ sum = 32 A

Spring 23

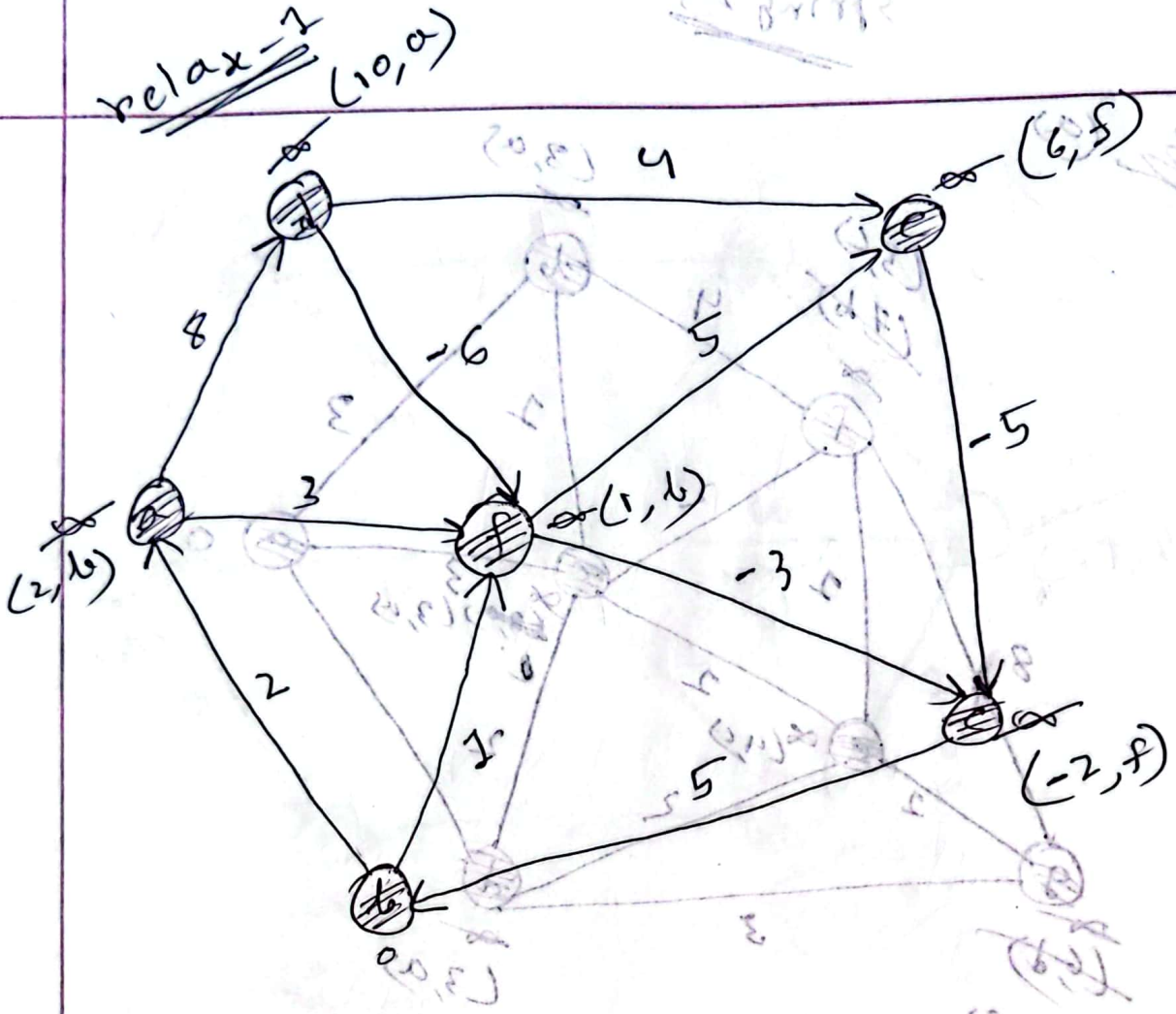
3 (a)



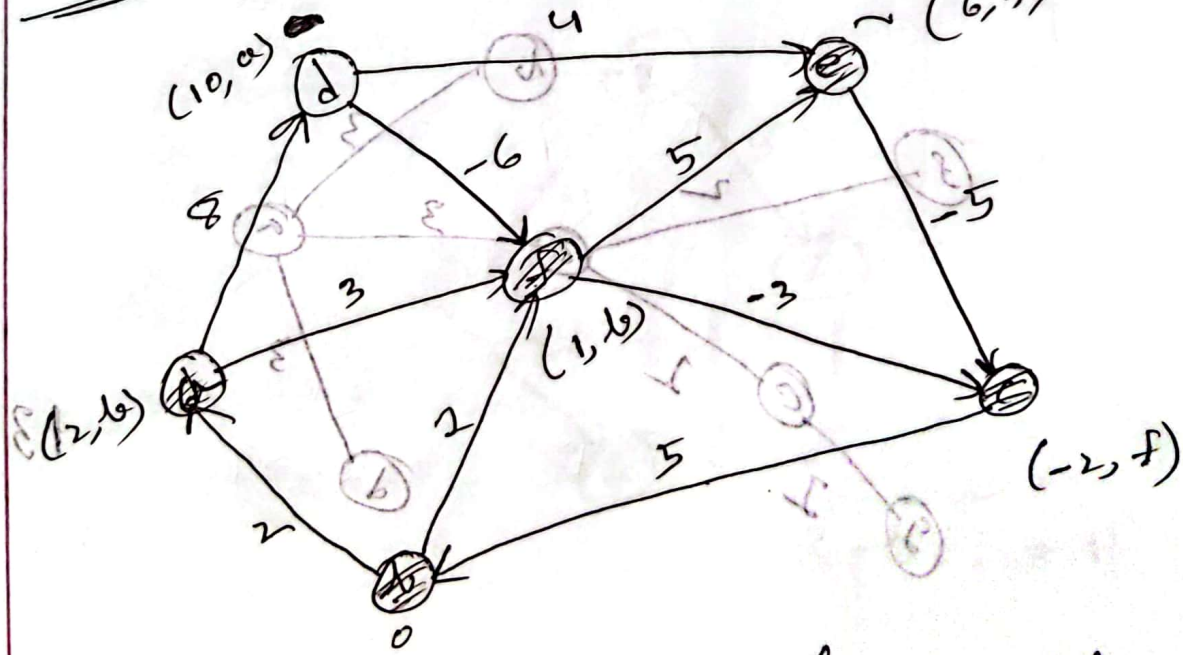


13

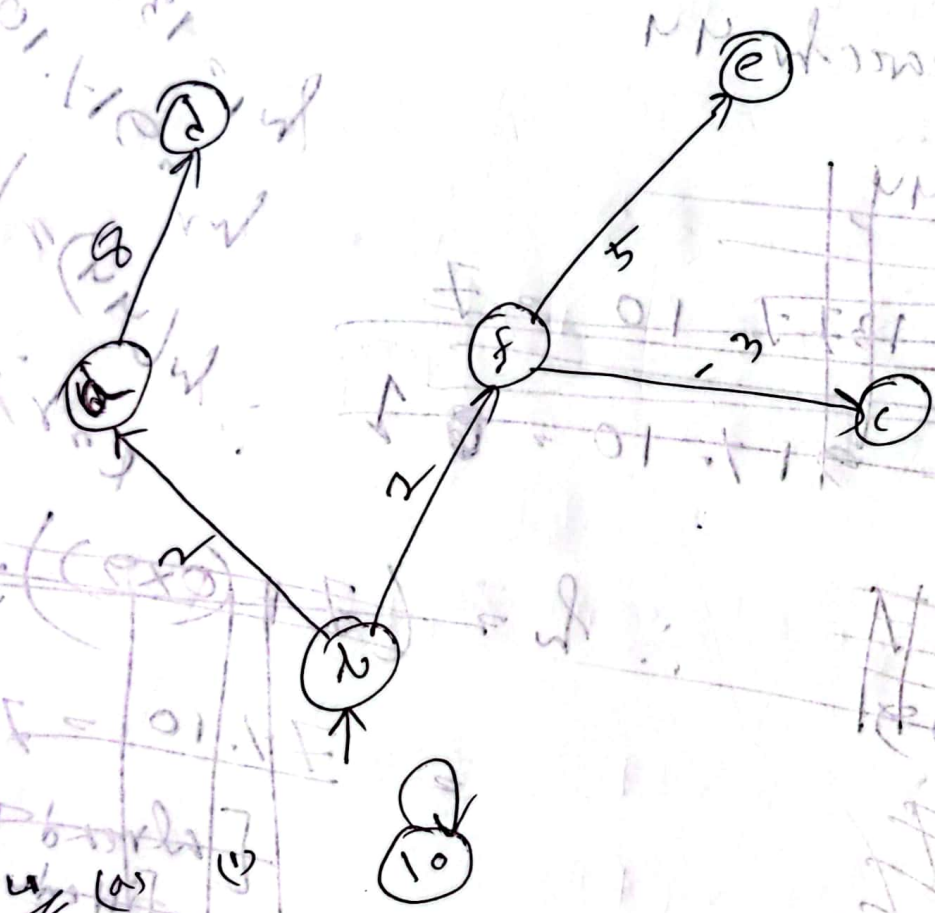
relax-1



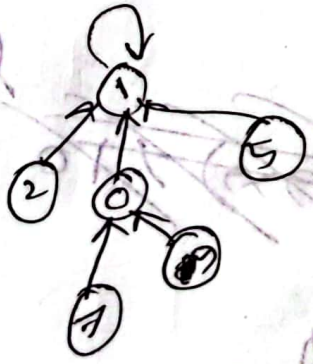
relax-2



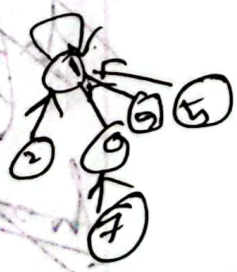
relax 3, 4, 5 will not change the value of weight.



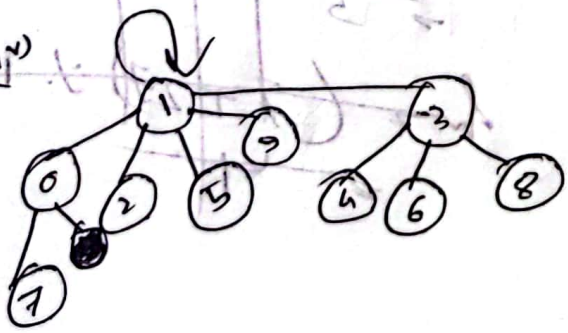
(ii) find set (0)



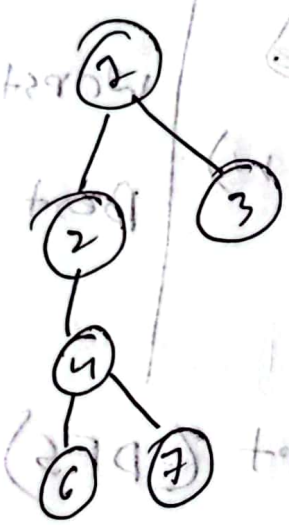
(iii) find set (9)



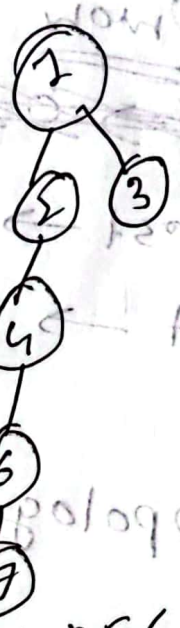
(iv)



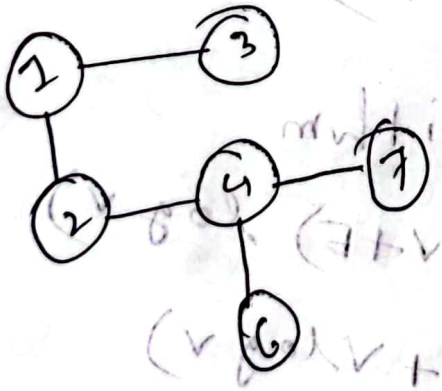
2 2 4 6



BFS



DFS



Time Complexity

Disjoint set

Union	Find
~~Best → O(log n)~~	Worst → O(log n)
Worst → O(log n)	Best → O(1)
Best → O(1)	

topological sort (DFS)

Worst → O(V+E)

Best → O(V+E)

Prims algorithm

Worst → O((V+E) log V)

Best → O(E + V log V)

Krushal algorithm

Worst → O(E log E) or O(E log V)

Best → same

Dijkstra algorithm

worst → Bin heap → $O((V+E)\log V)$

Fib heap → $O(E + V\log V)$

Best → $O(E + V\log V)$ → Fib heap

Bellmanford algorithm

Worst → $O(V \times E)$

Best → $O(V \times E)$

Rabin karp algorithm

worst → $O((n+m) \times q)$

n → text length

m → pattern n

q → mod value

Best → $O(n+m)$

P.T.O

All hash table

Worst → O(N)

Best → O(1)

BFS

Worst → O(V+E)

Best → O(1)

DFS

Worst → O(V+E)

Best → O(1)

PHP tutorials