CSE221.2

Algorithms

What is Algorithms:

An algorithm is a sequence of unambiguous instructions for solving a computational problem.

no
① unnessenary Lines
② no repeated "

Algorithm is a building blocks of programming.

কোনো Software Development এর জন্য SDLC way use হয়,

⑦ Algorithm Vs Program.

Algorithm	Program
① algorithm is a systematic approach to solving a specific problem.	① set of instraction for a computer.
② written at Design time.	② written at implement time.
③ person with domain knowledge	③ Programmer.
④ Any language (Eng, Math problem)	④ Programming language (C, Java)
⑤ HW and OS Independent.	⑤ HW and dependent.
⑥ Prior Analysis	⑥ Posterial Testing.
⑦ Time and Space function	⑦ Watch Time and Bytes.

Properties of Algorithms:

④ Finiteness → Finite number of steps. Unnessary Line বা code বাদ হবে Finiteness.

⑤ Definiteness → Algorithm এ ব্যবহৃত প্রতিটি meaning থাকবে, অস্পষ্ট থাকবে না যেন বিভিন্ন meaning বোঝায়,

⑥ Input → Valid input clearly বলা থাকতে হবে,

⑦ Output → " output "

⑧ Effectiveness → Algorithm effective হবে যখন Algorithm use করার জন্য যেসব resource and device থাকতে, must be accessible resources.

How to write an Algorithm?

⑥ Pseudocode for swapping two numbers:

```
Algorithm swap (a, b){
    temp = a;
    a = b;
    b = temp;
}
```

Analyze an Algorithm:

① ⑦ Time → How much time is it taking. → time function

① ⑧ Space → space function

⑨ N/W → Data transfer

⑩ Power consumption

⑪ CPU Registers

↳ how many registers is it going to consume to execute as a program.

Time f^n ⟶ Time complexity.

Space f^n ⟶ Space "

(*) a, b, temp are considered as word, not byte.

(#) every simple statement of an algorithm takes 1 unit of time.

(#) O(1) means constant.

Pseudocode

Algorithm Swap(a,b)

{

temp = a; ⟶ 1

a = b; ⟶ 1

b = temp; ⟶ 1

}

T(n) = 3

Space

a → 1

b → 1

temp → 1

S(n) → 3

Constant value यदि आई, तो T(n) तथा S(n) का order भी Constant,

Time complexity → O(1) ⟶ (order of 1)

Space " → O(1)

Frequency Count Method

8	3	4	7	2
0	1	2	3	4

n = 5

Algorithm sum(A, n) → A: Array

```
Algorithm sum(A, n)                          Time
{   s = 0;                                   → 1
    for (i = 0; i < n; i++) {                → n+1
         (1)   (n+1)  (n)
        s = s + A[i];                        → n
    }
    return 0;                                → 1
}
```

Space

- A (Array) → n
- n → 1
- s → 1
- i → 1

$S(n) \rightarrow n+3$

$T(n)/f(n) = 2n+3$

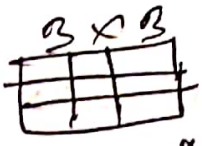
for loop এ (0 - n) পর্যন্ত চালালে (n+1 হবে), কিন্তু for loop ভিতরের statement (n) term পর্যন্ত হবে, সুতরাং for loop 1 টি term বেশি (n+1) check করে, আর for ভিতরের statement n Term হয়,

Time complexity → $O(n)$

Space " → $O(n)$

n term আসলে, n এর Highest degree তত বেশি হবে, → n^n

④ An algorithm to add two matrices A and B of size n,



Time t^n

Algorithm add (A, B, n)

{ for (i=0; i<n; i++){ ⟶ n+1

for (j=0; j<n; j++){ ⟶ n(n+1)

c[i,j] = A[i,j] + B[i,j]; ⟶ n×n

} (2D)

}

}

$f(n) = 2n^2 + 2n + 1$

2D Array = n×n = n^2

Space

A = n^2
B = n^2
C = n^2

n = 1
i = 1
j = 1

$S(n) = 3n^2 + 3$

∴ Time complexity ⇒ $O(n^2)$

∴ Space " ⇒ $O(n^2)$

⑧ for loop (n+1), for loop of statement (n) + n.

④ 1D Array = n
2D " = n^2 } Space complexity

Pseudocode:

```
Algorithm Multiply(A, B, n)
{   for(i=0; i<n; i++){                          ---> n+1
        for(j=0; j<n; j++){                      ---> n(n+1)
            c[i,j] = 0;                          ---> n x n
            for(k=0; k<n; k++){                  ---> n x n x (n+1)
                c[i,j] = c[i,j] + A[i,k] * B[k,j];
                                                 ---> n x n x n
            }
        }
}
```

$T(n) = n+1 + n(n+1) + n \times n + n^2(n+1) + n^3$

Time complexity = $O(n^3)$

Space " = $O(n^2)$

Time Complexity

Pseadocode:

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

2nd for loop 1st for loop এর i এর উপর Dependant

i	j	no of time
0	0	0
1	1	1
2	2	2
3	3	3
⋮		⋮
n		n

⊛ যোগ আকারে (সমান্তর ধারা),

$$1+2+3+\cdots+n = \frac{n(n+1)}{2}$$

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

∴ Time complexity = $O(n^2)$

Used variable = i, n, j
= 1+1+1
= 3 (constant variable)

∴ Space complexity = $O(1)$

⊛ যখন inner for loop outer যতো for loop এর উপর নির্ভর করে, তখন series Type হয়,

Pseudocode:

```
for(i=1; i<n; i=i*2){
    stmt;
}
```

i
$1 \times 2 = 2$
$2 \times 2 = 2^2$
$2^2 \times 2 = 2^3$
.
.
.
2^k

Assume $i >= n$

$\because i = 2^k$

$\because 2^k >= n$

$2^k = n$

$k = \log_2 n$

$i = 1$

$n = 1$

2

$\therefore$ Time complexity $= O(\log_2 n)$

$\therefore$ Space complexity $= O(1)$

Pseudocode:

```
p=0;
for(i=1; p<=n; i++)
{
    p = p+i;
}
```

i	p
1	0+1=1
2	1+2=3
3	1+2+3=6
4	1+2+3+4=?
.	.
.	.
k	1+2+3+4+ ---- +k

Assume, $p > n$

$\therefore p = \frac{k(k+1)}{2}$

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

$k^2 > n$

$k > \sqrt{n}$

Time Complexity = $O(\sqrt{n})$

④ loop condition p কত? true? Depend করে।

Using 'if' and 'while'

Pseudocode:

```
i=0;                    ——— 1
while(i<n){             ——— n+1
    stmt;               ——— n
    i++;                ——— n
}
```

$f(n) = 3n+2$

$i = 0 \to 1$
$n = 1$
2

$\therefore$ Time complexity = $O(n)$, Space complexity = $O(1)$

Pseudocode:

```
while(m!=n){
    if(m>n){
        m=m-n;
    }
    else{
        n=n-m;
    }
}
```

m=16	n=2
14	2
12	2
10	2
8	2
6	2
4	2
2	2

Time Complexity = $O(n)$

Space Complexity = 2

$m=1$
$n=1$
2

* $n/2$

Time function Summary

- $O(1)$ ——— Constant
- $O(\log n)$ ——— Logarithmic
- $O(n)$ ⟶ Linear
- $O(n^2)$ ⟶ Quadratic
- $O(n^3)$ ⟶ Cubic
- $O(n^n)$ ⟶ Exponential

Compare time function

$$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \cdots < 2^n < 3^n < n^n$$

- $1 < \log n < \sqrt{n}$ — Lower bound
- n — Average bound
- $n\log n < n^2 < n^3 < \cdots < 2^n < 3^n < n^n$ — Upper bound

Asymptotic Notations:

O-notation: When we have only an asymptotic upper bound.

Ω-notation: When asymptotic lower bound.

θ-notation: When asymtotic average bound.

Ch-3

Divided and Conquer

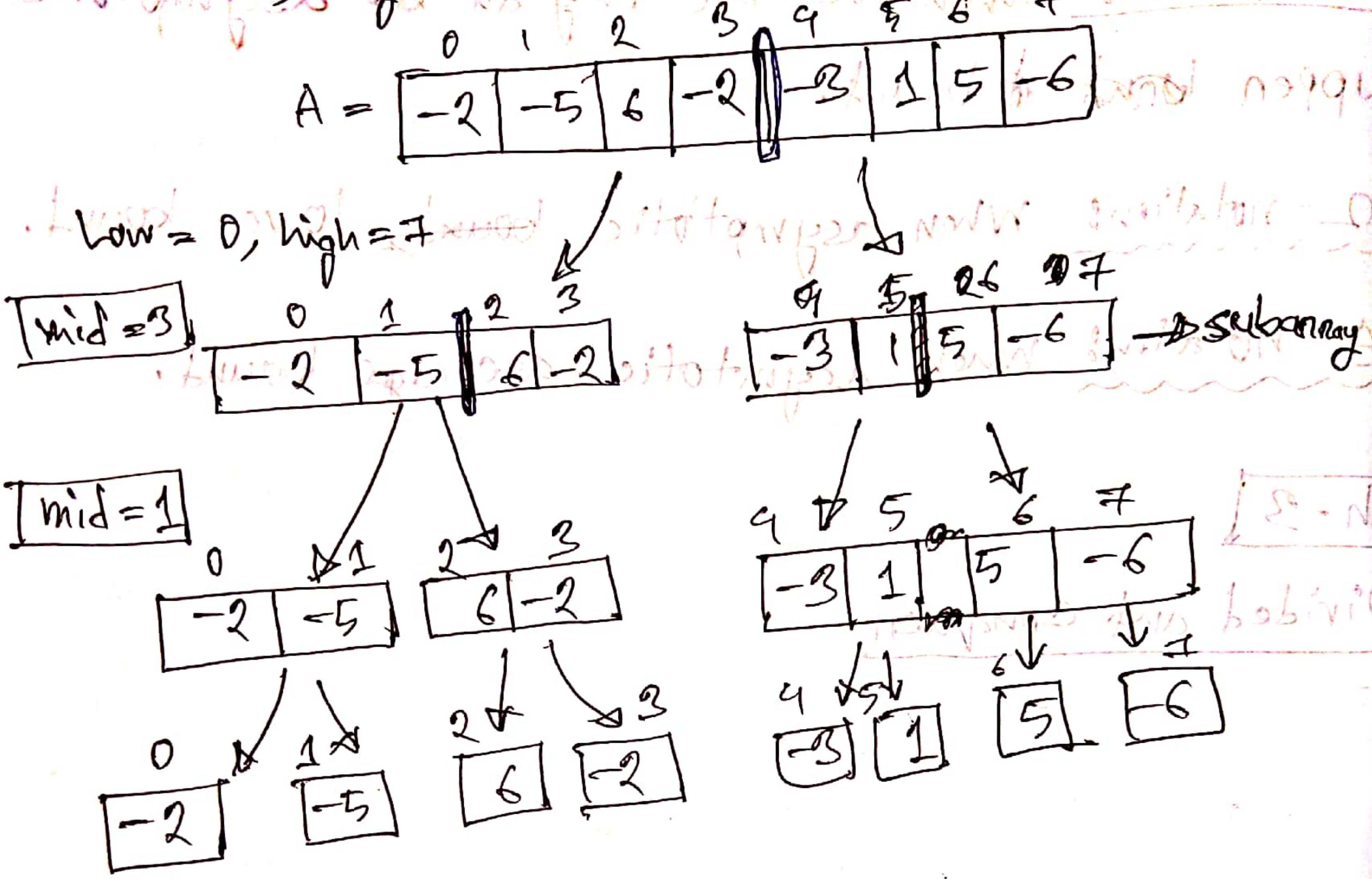
Maximum Sum Subarray (MSS)

Two approaches to solve program:

1. DAC
2. DP (Kadanes Algorithm)

DAC Approach :

Low = 0, high = n-1

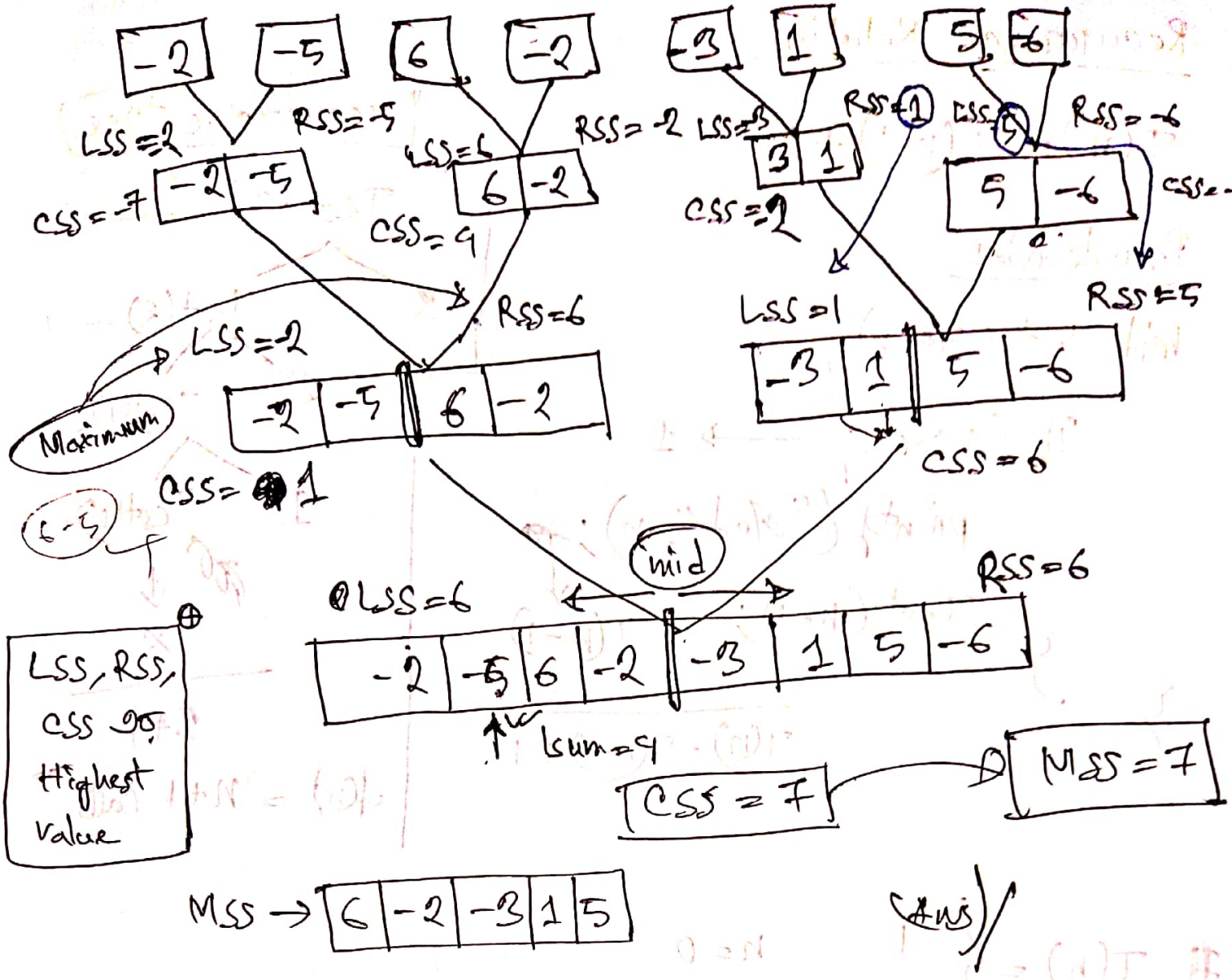


MSS

RSS

CSS = LSS + RSS

* compare CSS, LSS and RSS, যেটি বেশি (Max) হবে তাই হবে MSS এর output.



Time Complexity: maxSubArraySum() function is a recursive method and time complexity can expressed,

$$T(n) = 2T(n/2) + \theta(n)$$

maxSumarraySum tree is binary tree. So,

Time complexity = $O(n \log n)$

Recurrence Relation

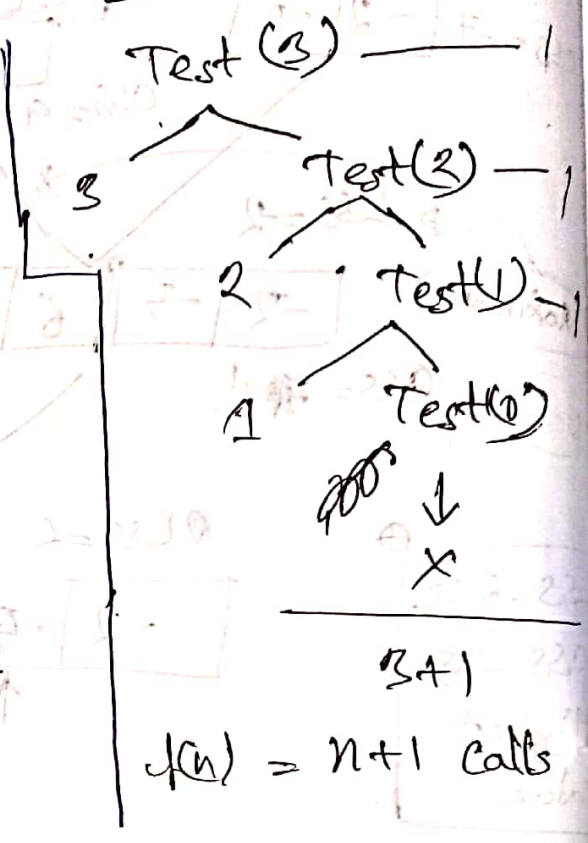
$T(n) = T(n-1) + 1$

Pesudocode:

```
void Test (int n){
    if (n>0){ ——→ 1
        printf ("%d", n);
        Test (n-1);   T(n-1)
    }
}
```

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

Recurrsive tree



$$T(n) = \begin{cases} 1 & n=0 \\ T(n-1)+1 & n>0 \end{cases}$$

Back substitute,

Here, $T(n) = T(n-1)+1$

Subsitute $T(n-1)$,

$\Rightarrow T(n) = [T(n-2)+1]+1$

$\Rightarrow T(n) = T(n-2)+2$

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

$\Rightarrow T(n-1) = T(n-1-1)+1$

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

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

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

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

⋮ continues for k times

$T(n) = T(n-k) + k$

Now, $T(n) = T(n-k) + k$

Assume, $n - k = 0$

$\therefore n = k$

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

$T(n) = T(0) + n$

$T(n) = 1 + n$

$\Theta(n)$ / $O(n)$

Kadane's Algorithm

Dynamic Programming

A[4] = | -2 | 3 | 2 | -1 |
0 1 2 3

A = [-2, 3, 2, -1]

subarray

c → current

g → global

Pesudocode:

```
max_current = max_global = A[0];

for(i=1; i<n-1; i++){

    max_current = max(A[i], max_current+A[i])   // which is maximum

    if (max_current > max_global){

        maxglobal = max_current                 // MSS

    }

}

return max_global
```

~~Hence,~~

~~max(2,3)~~

i	0	1	2	3
c	-2	3	5	4
g	-2	3	5	5

i=1, c=3, g=3

i=2, c=5, g=5

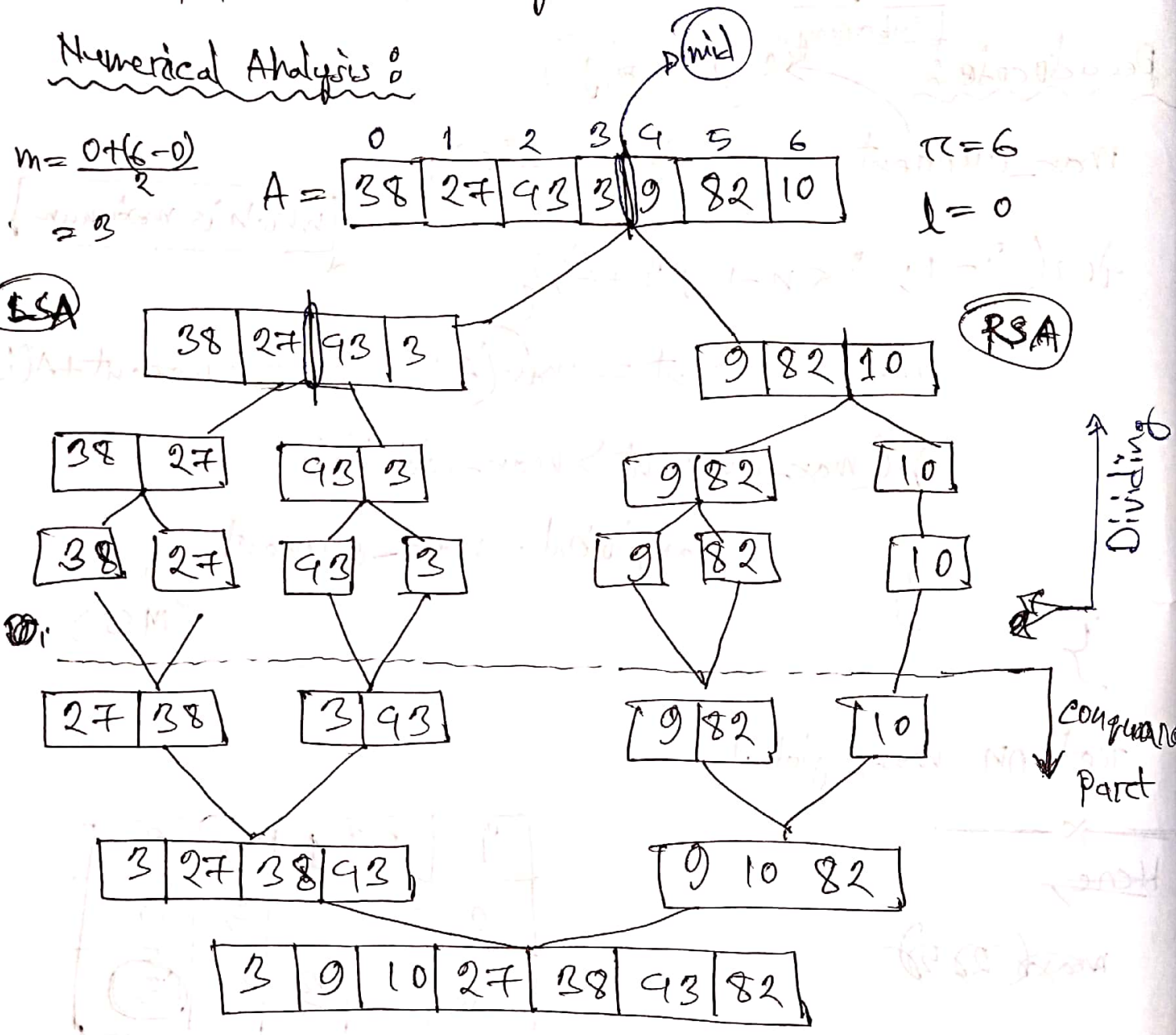
i=3, c=4, g=5 → MSS

Time Complexity: O(n)

Merge Sort

Most popular ~~sotin~~ sorting.

Numerical Analysis :



Time Complexity :

Best Case → O(n log n)

worst " → O(n log n)

Average " → O(n log n)

Quick sort (DAC)

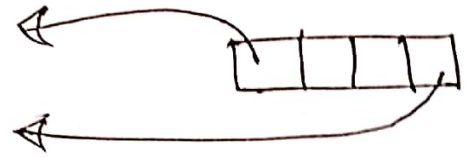
Quick sort algorithm based on the divided and conquer approach.

Where use → Search information

Array is divided into subarrays by selecting a pivot element.

Pivot element select 3 ways:

1. left most element.
2. Right most "
3. Random



Of course a right most element নিয়ে Quicksort এর কাজ করা হবে,

For numerical analysis: $i = low - 1$

$j = low$

Pivot = A[high]

* i start হয় (-1) থেকে, যদি index 0 থেকে start হয়,

Numerical

	0	1	2	3	4	5	6
A =	10	80	30	90	40	50	70
	Low					high	Pivot

Low = 0
high = 6
Pivot = A[high]
= 70

① $i = 0 - 1 = -1$

for (j = 0 to 5)

② j = 0, A[j] = 10, Pivot = 70, i++

i = 0

No Swap A[0] & A[0]

j++

Right side का element को Pivot कहते हैं।

③ j = 1, False, no movement, j++

④ j = 2, True, i++, i = 1, swap A[1] & A[2], j++

A =	10	30	80	90	40	50	70

⑤ j = 3, False, no movement, j++

⑥ j = 4, True, i++, i = 2;

Swap A[2] & A[4]

j++

A =	10	30	40	90	80	50	70

⑦ j = 5, True, i++, i = 3;

swap A[5] & A[3], j++

	0	1	2	3	4	5	6
A =	10	30	40	50	80	90	70

(end for loop)

Sorted

~~⑦ j=5, True, i++, i=3,~~

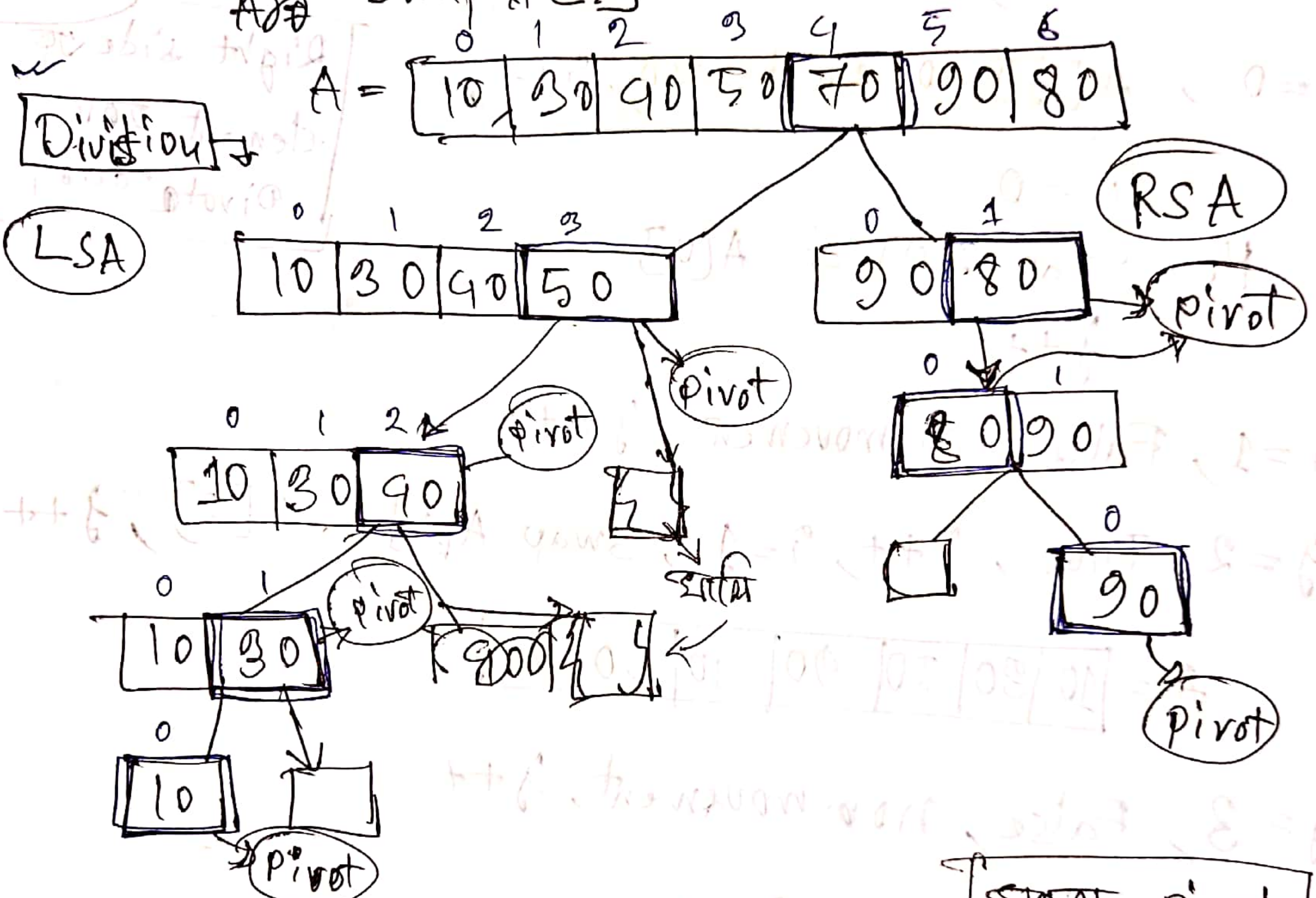
~~Swap A[5].~~

* Pivot এর মধ্যে Swapping হয়।

⑧ i+1 = 9, swap A[9] & A[High]

A[9] → 80, A[High] → 70

~~Swap A[5]~~



A = 10 | 30 | 40 | 50 | 80 | 90

সকল pivot সাজিয়ে আছি

Binary tree এর মত দেখাচ্ছে আছি তাই,

Time complexity = O(nlogn)

For left sub array,

0	1	2	3
10	30	40	50

← Small < Pivote < big

Pivot = 50, Low = 0, high = 3

for(j = 0 to 2), i = -1

① j = 0, True, i = 0, no swap, j++

② j = 1, True, i = 1, " " , j++

③ j = 2, True, i = 2, " " , j++

④ j = 3, True,

(end for loop) A[3] = Pivot, no swap (because same)

For right sub array,

0	1
90	80

i = -1, Pivot = 80

j = 0 to 0

① j = 0, False, No swap,

(for loop end) i + 1 = 0

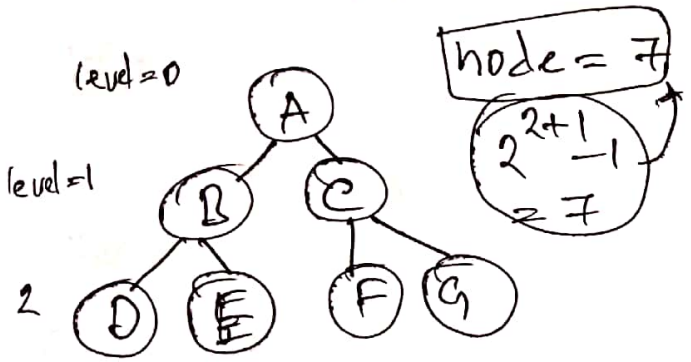
Pivote = A[1] = 80

swap A[1] & A[0]

80	90

Binary Binary Tree (Heap Sort)

→ Full Binary Tree.
↳ Complete Binary Tree.



node = 7
$2^{2+1} - 1 = 7$

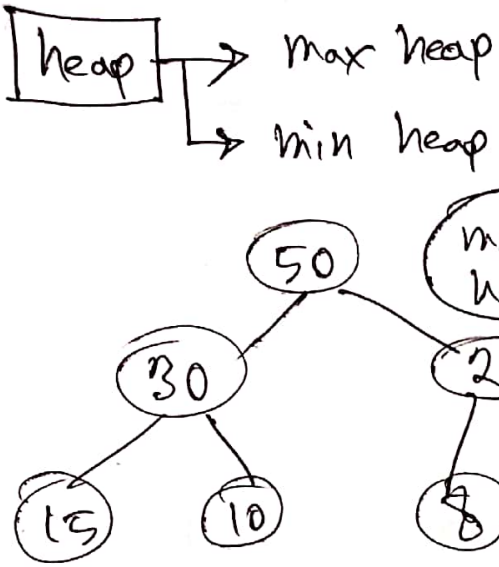
T = | A | B | C | D | E | F | G |

- Array representation of binary tree,

A node is at index → i
its left child is at → 2 * i
its Right " " " → 2 * i + 1
its parent is at → $\left\lfloor \frac{i}{2} \right\rfloor$ → Floor value

- Full Binary tree has maximum $2^{h+1} - 1$ nodes

Level - 2^h h



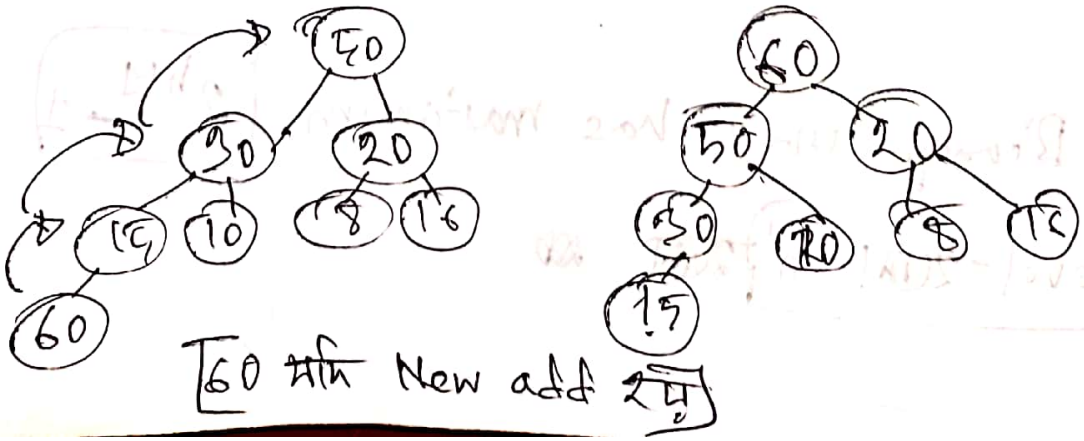
H	50	30	20	15	10	8	16

* Heap sort 2 step ও কাজ করে,

1. Max heap, addition করে (অর্থাৎ Array থেকে Max heap তৈরি করতে হবে,
2. Max heap deletion (Max Heap তৈরী করার পর তার অর্থাৎ root delete করতে হবে)

→ 2 step এর পর আমরা completly অর্থাৎ sorted Array পাব,

Leaf towards Root: যে কোনো Tree তে নতুন কোনো Node add করতে হলে lowest stage এ add করতে হবে,



60 টি New add হয়েছে

For heap sort,

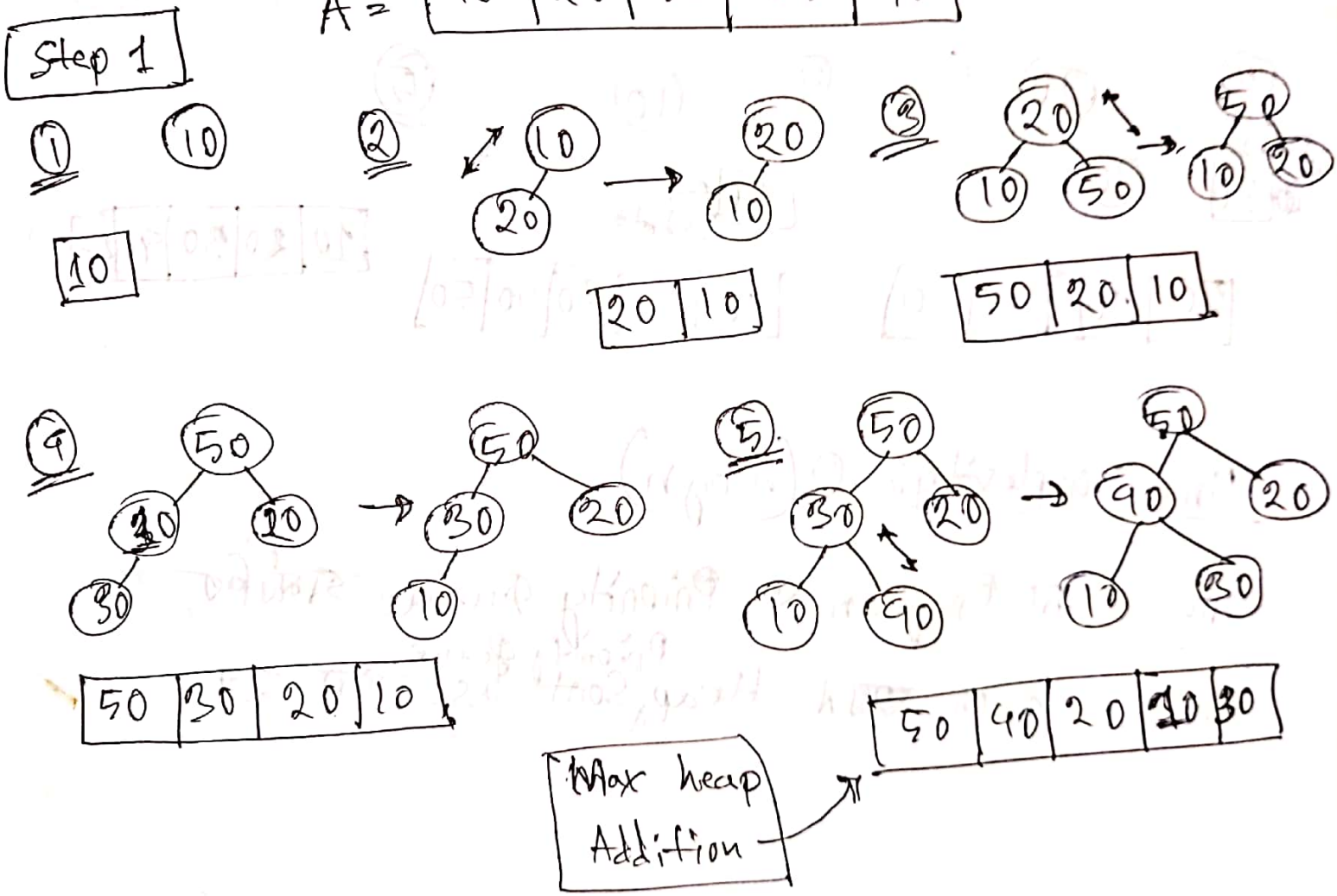
(1) Create the heap (max) from given array.

(2) Delete one by one element from heap to form a sorted array.

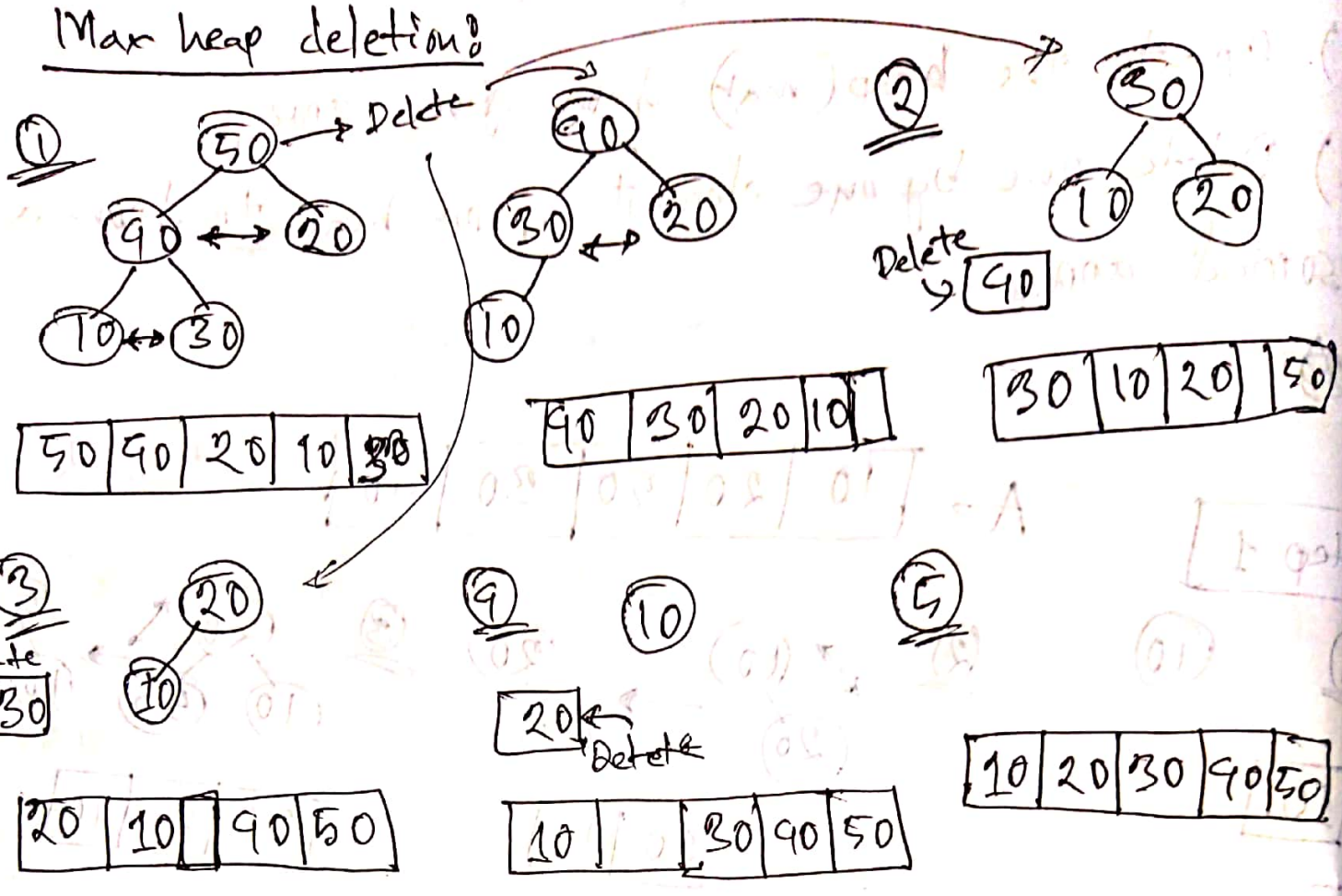
A =

10	20	50	30	40

Step 1



Max heap deletion:



Time complexity: $O(n \log n)$

যদি কোনো Program এ Priority Queue ব্যবহার Problem আসে তখন Heap Sort (Priority Queue) use করা হয়।

Final Term

Chapter-4

6 Mark

Greedy Method

Always makes the choice that seems to be the best at that moment. This mean a locally optimal choice in the hope that this choice will lead to a globally-optimal solution.

optimal → optimization

* optimization মানে minimize or maximize করা হোল Problem টে, এই জন্য আমরা Backtrack or reverse করতে না,

Greedy algorithms have five components:

Knapsack Problem (0-1)

⊛ Fractional problem (0-1), 0 or 1 represent.

⊛ ~~ইচ্ছে~~ কোনো unit দিয়ে কাজ হবে না,

① Bag ও 15 weight পর্যন্ত ধরা,

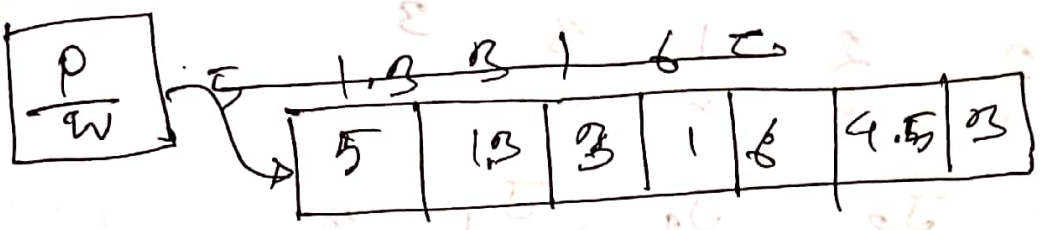
আমাদের Highest মান নিতে হবে P/W ও ~~ইচ্ছে~~ যেতে,

$n = 7$ (total Objects)

$m = 15$ (total weights যা যত পারে Bag ও)

(2) same type,

object	0	1	2	3	4	5	6	7
Profit p	0	10	5	15	7	6	18	3
weight w	2	3	5	7	1	4	1	



Highest to low

$x \rightarrow (1 \quad 2/3 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1)$
$x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7$

$0 \le x_i \le 1$

$\sum x_i w_i = 1\times2 + 2/3\times3 + 1\times5 + 0\times7 + 1\times1 + 1\times4$

$+ 1\times1 = 2+2+5+0+1+4+1 = 15$

Total weight

$\sum x_i p_i = 1\times10 + 2/3\times5 + 1\times15 + 1\times6 + 1\times18 + 1\times3$

$= 10 + 2\times1.3 + 15 + 6 + 18 + 3$

$= 59.6$

Max Profit

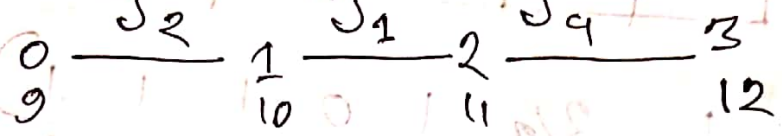
Total Profit

15 − 1 = 14
14 − 2 = 12
12 − 4 = 8
8 − 5 = 3
3 − 1 = 2
2 − 2 = 0

Job sequencing with deadlines

$n = 5$

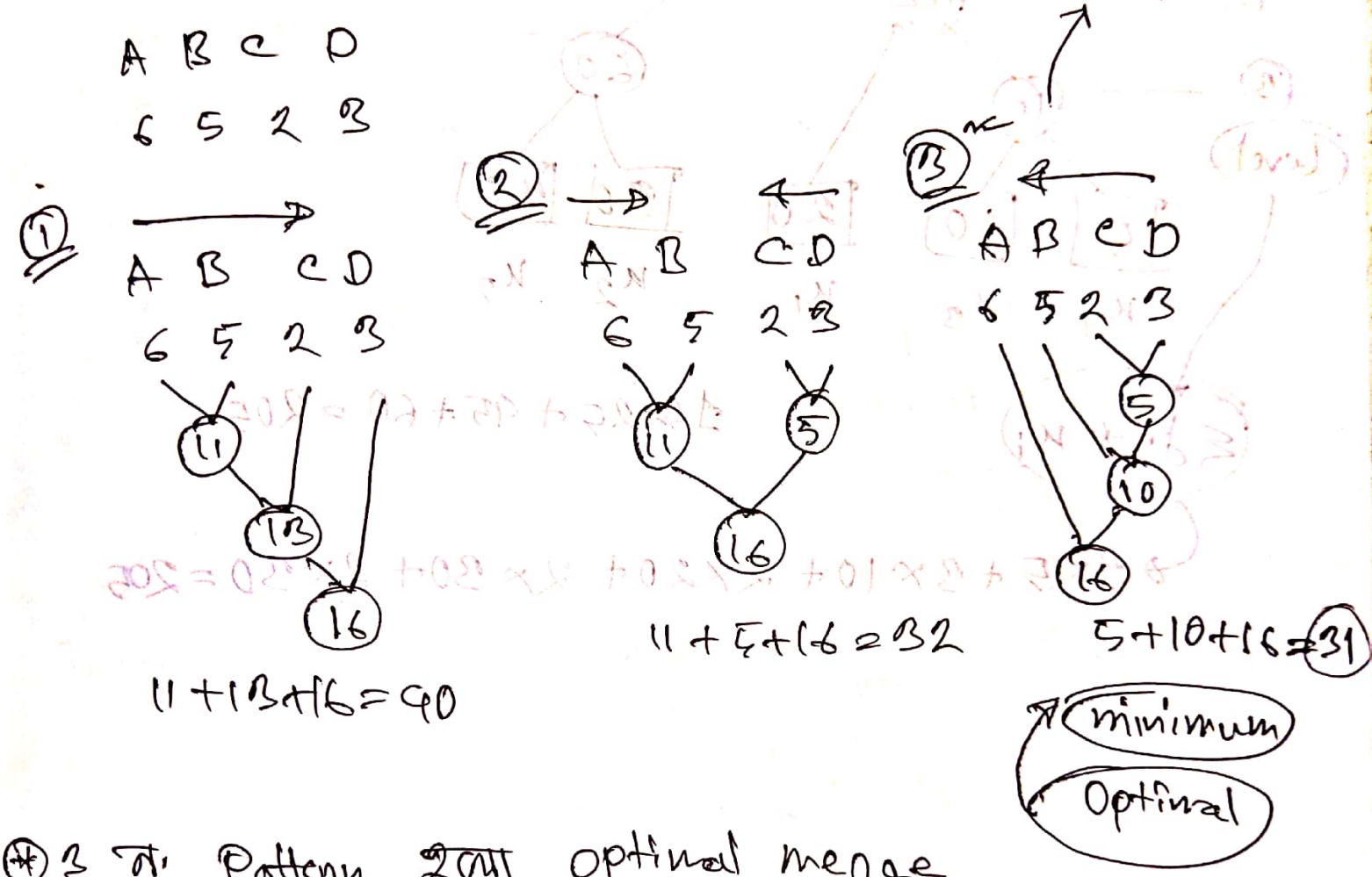
Jobs	J_1	J_2	J_3	J_4	J_5
Profits	20	15	10	5	1
deadline	2	2	1	3	3



Job	slot	solution	profit
—	—	—	0
J_1	[1, 2]	J_1	20
J_2	[0, 1] [1, 2]	J_1, J_2	20+15
J_3	[0, 1] [1, 2]	J_1, J_2	20+15
J_4	[0, 1] [1, 2] [2, 3]	J_1, J_2, J_4	20+15+5

Optimal Merge Pattern

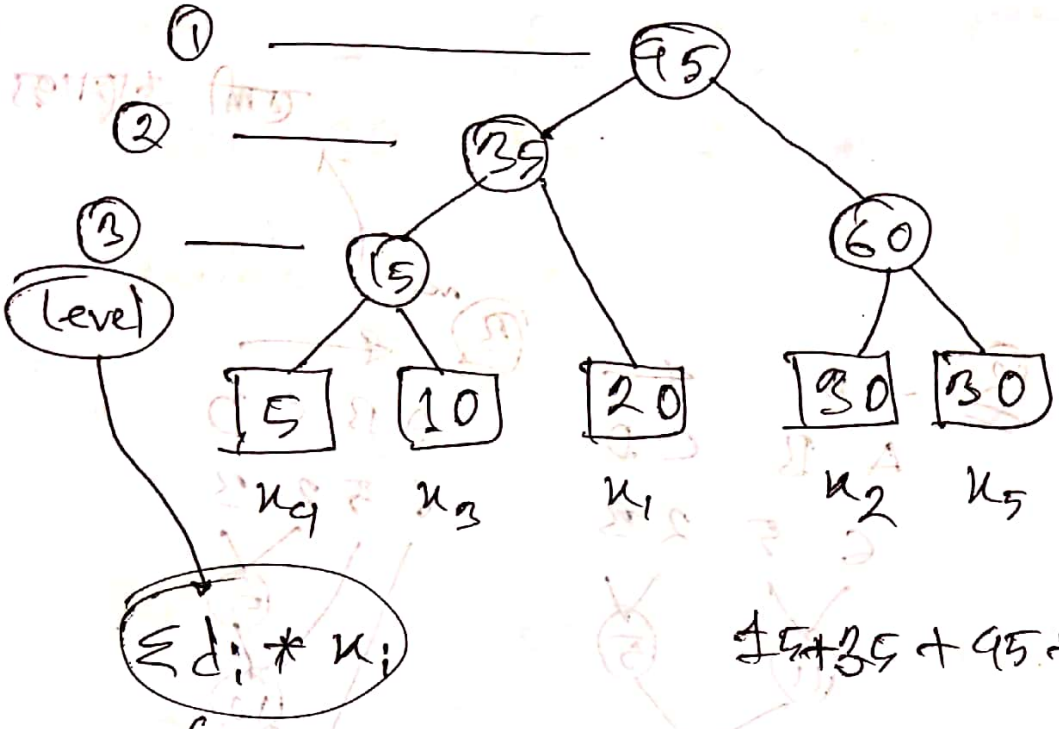
এটি কোনো Algorithm নয়, এটি concept যা বিভিন্ন Algorithm এ use হয়।



⊛ 3 নং Pattern হলো optimal merge pattern.

x_1 x_2 x_3 x_4 x_5

20 30 10 5 30



$\sum d_i * x_i$

$5+35+95+60 = 205$

$3\times5 + 3\times10 + 2\times20 + 2\times30 + 2\times30 = 205$

Huffman coding:

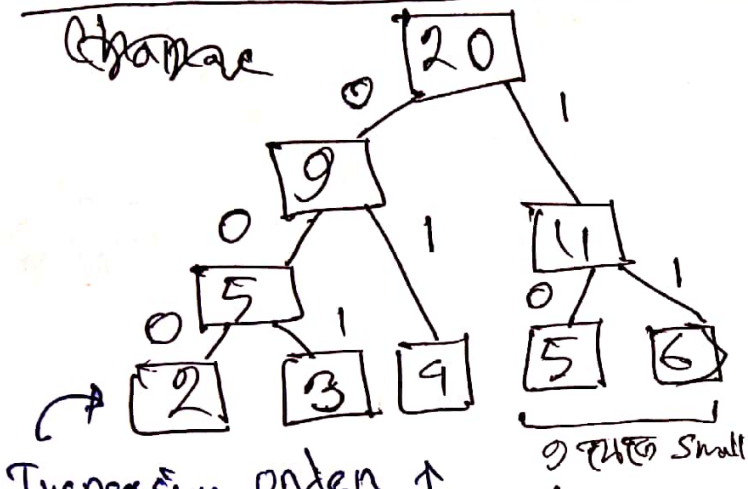
Example: message → B C C A B B D D A E C C B B A E D D C C

length = 20

for ASCII = 8 bit

Total bit = 8 × 20 = 160 bits

Variable sized code: (Method)



Left side = 0

right " = 1

Ascending order

Increasing order ↑

variable sized code → root to leaf

Char	Count	Code	Total
A	3	001	3 × 3 = 9
B	5	10	5 × 2 = 10
C	6	11	6 × 2 = 12
D	4	01	4 × 2 = 8
E	2	000	2 × 3 = 6
5 × 8 bit = 40 bit	20	12 bit	45 bits

$\Sigma di * ti = 3\times2 + 3\times3 + 2\times4 + 2\times5 + 2\times6$

$= 45$ bit

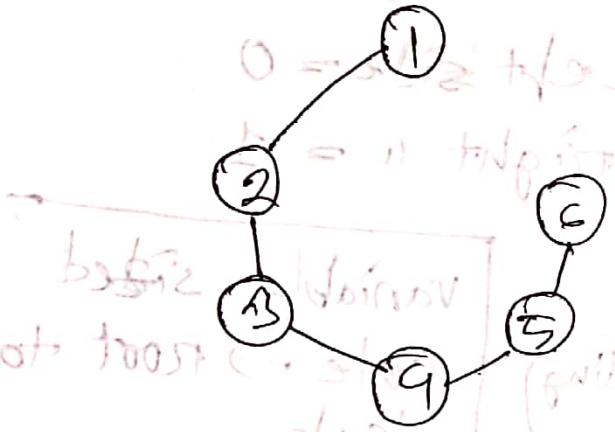
Here,

Message = 45 bit

Table / Tree = 40 + 102 bit

97 bit

Minimum Spanning Tree



(*) Spanning tree তে কোনো loop বা cycle থাকে না।

(*) Edge = vertex - 1

$S = (V, E)$

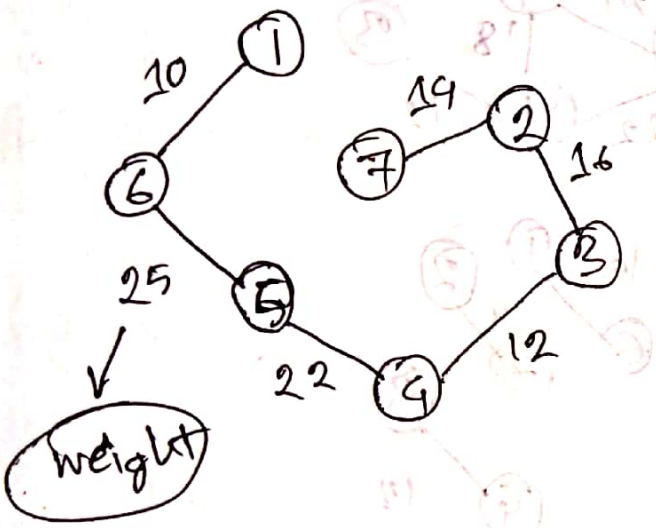
(*) কত গুলো spanning tree বানানো সম্ভব,

$|E|C_{|V|-1}$ = number of cycles

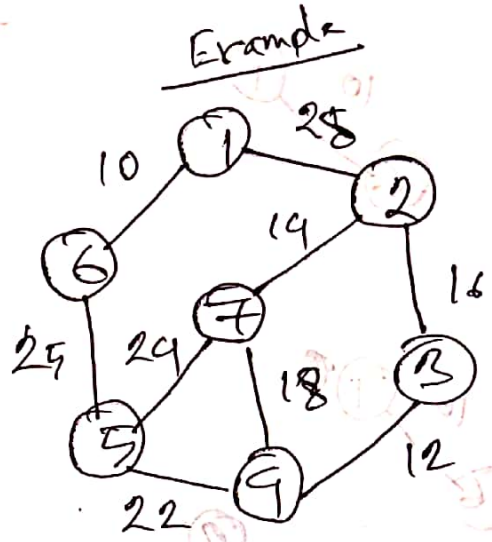
Two method to find minimum cost :

① Prim's ② Kruskal's

Prim's Algorithm



Example



* Always select minimum edge which is connected.

$$\text{Cost} = 99$$

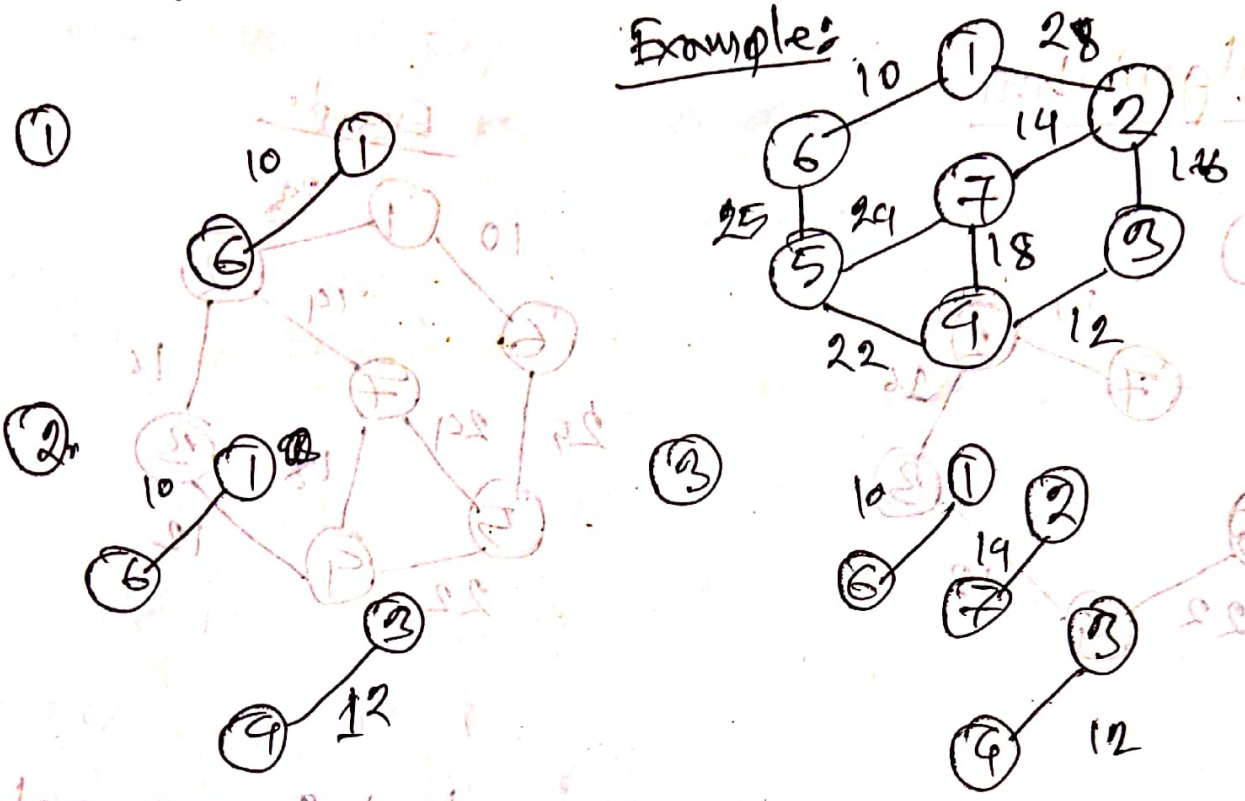
* Non connected Algorithm ~~এতে~~ ~~হয়~~ না,
* ~~সকল~~ Node completely connected ~~হয়, comple~~ ~~হয় হয়~~,
* Weight ~~এ~~ ~~নির্ণয়~~ shortest path find ~~করে~~,
* No loop.

Prim's algorithm is helpful when dealing with dense graphs that have lots of edges.

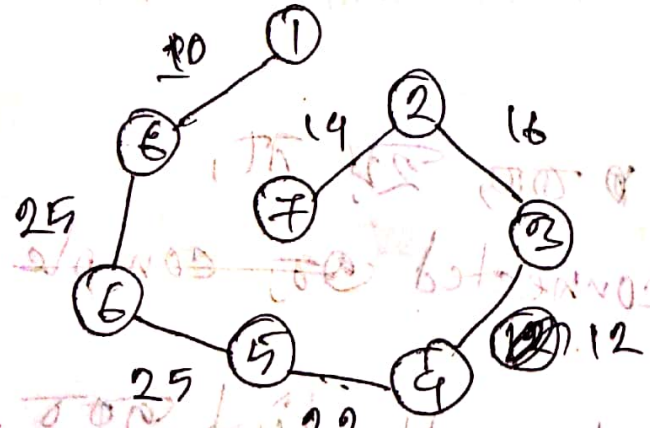
Kruskal's Algorithm

① Always select minimum ~~edge~~ weigh edge.

Example:



② Final चित्र हमारा:



cost = 99

|E| = 6

|V| = 7

Time complexity : $O(n^2)$

By using Min Heap in Kruskal's algorithm

Time complexity = $O(n \log n)$

* The optimal cost will be same for different spanning tree.

Dijkstra Algorithm:

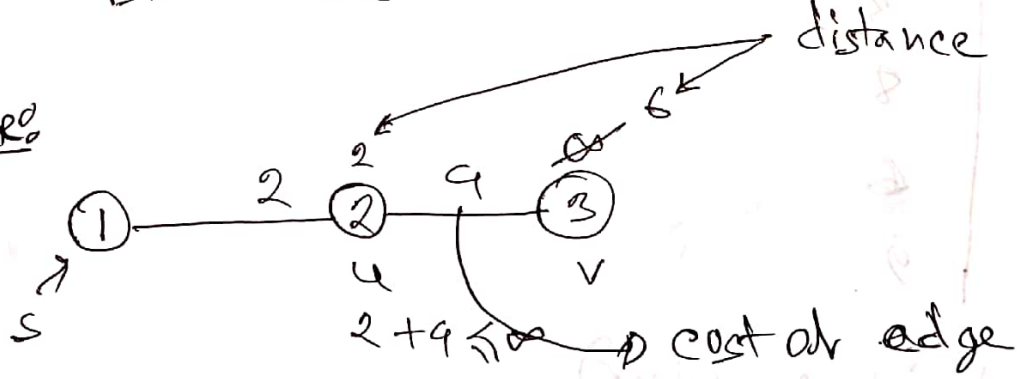
Minimization, Optimization.

* It works Directed or Non-directed graph.

Relaxation

```
if (d[u] + c(u,v) < d[v]){
    d[v] = d[u] + c(u,v) }
```

Example:

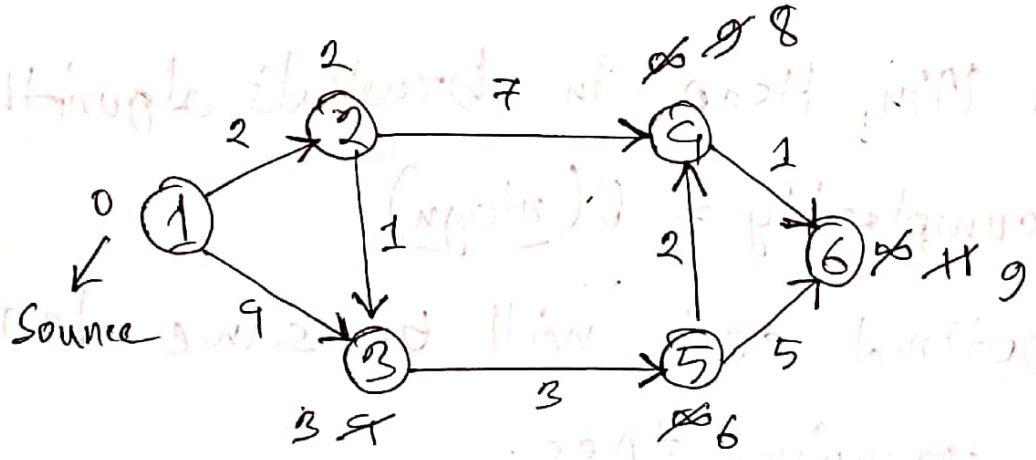


Here,

$d[u] = 2$

$d[v] = \infty$

$c(u,v) = 4$



⊛ Source থেকে বাকো Node ও edge না থাকলে ∞ চিহ্ন দিতে হবে।

এখন, Source থেকে shortest path node, ও select করতে হবে, তারপর ঐই node থেকে shortest weight Find করে Relaxation Method use হবে,

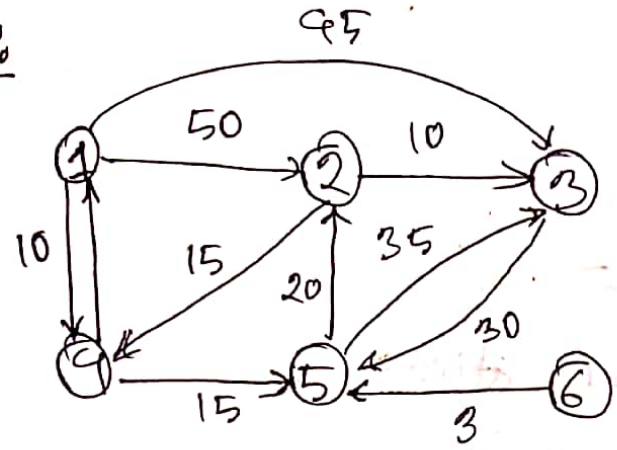
⊛ ∞ থেকে সরাসরি Number চেঞ্জ,

Total shortest path, vertex,

Vertex	d[v]
2	2
3	3
4	8
5	6
6	9

Time complexity: $\theta(n^2)$

Example:



Starting Vertex 1

minimum between (50, 95, 10)

selected vertex	2	3	4	5	6	Smallest d[v]
4	50	95	(10)	∞	∞	
5	50	95	(10)	(25)	∞	
2	(95)	95	(10)	(25)	∞	→ no change
3	(95)	(95)	(10)	(25)	∞	
6	(95)	(95)	(10)	(25)	(∞)	→ 6 doesn't have edge

Disadvantage:

→ negative distance থাকলে Dijkstra algorithm কাজ করে না.

Dynamic Programming

DP have two appnochi:-

1. Top-down with Memoization.
2. Bottom-up with Tabulation.

LCS (longest common subsequence)

String compare করা হয়, কোনো ধরণের consecutive connection ছাড়া,

used in LCS is -

→ LCS using recursion.

→ LCS ও Memoization.

→ LCS ও Dynamic Programming.

Example:

String 1: a b c d e f g h i j

String 2: c d g i

LCS = cdgi

* Don't intersect too line.

Example: String 1: a b c d e f g h i j

String 2: e c d g i

LCS = cdgi

we get,
① egi
② cdgi

longest common subsequence নিতে হবে।

Solving LCS using dynamic programming (Bottom up).

A | b | d |
1 | 2 |

B | a | b | c | d |
1 | 2 | 3 | 4 |

Condition

if (A[i] = B[j])

$$LCS[i,j] = 1 + LCS[i-1, j-1]$$

else

$$LCS[i,j] = \max(LCS[i-1, j], LCS[i, j-1])$$

Here,

* আমরা 1 index থেকে start করে থাকি।

i \ j		a	b	c	d
		0	1	2	3	4
	0	0	0	0	0	0
b	1	0	0	1	1	1
d	2	0	0	1	1	2

→ Size of the LCS

'0' index fill up করে থাকি, '0' index ও।

চিত্রের এর steps অনুযায়ী,

① i = 1, j = 1, No match, max(LCS[0, 1], LCS[1, 0])
, LCS [1, 1] = 0,

~~② i = 2, j = 2, 1'~~

② i = 1, j = 2, Match,

LCS[1, 2] = 1 + LCS[0, 1]

= 1 + 0

= 1

③ i = 1, j = 3, No Match,

LCS[1, 3] = ~~1 + LCS[0, 2]~~ = max(LCS[0, 3], LCS[1, 2])

= ~~1 + 0~~

= ~~1~~ = max(0, 1)

= 1

④ i = 1, j = 4, No match,

LCS[1, 4] = ~~1 + LCS[0, 3]~~ = max(LCS[0, 4], LCS[1, 3])

= ~~1 + 0~~

= ~~1~~ = max(0, 1)

= 1

⑤ ~~start~~ skip --- steps ---

i = 2, j = 4, Match,

LCS[2, 4] = ~~max(LCS[1, 4], LCS[2, 3])~~

= ~~max(~~

= 1 + LCS[1, 3]

= 1 + 1

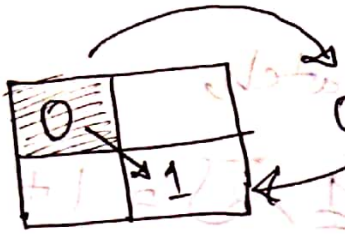
= 2

(*) Technic to solve LCS table,

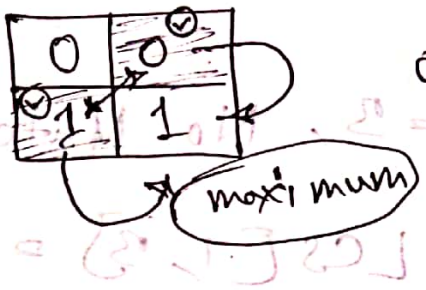
यदि match हुआ 1 + LCS[i-1, j-1],

और यदि match ना हुआ maximum (LCS[i-1, j], LCS[i, j-1]).

Match हुआ →



Match ना हुआ →



0/1 Knapsack Problem : 2 Methods (DP)

Object নিলে 1 হবে, object না নিলে 0 হবে, অর্থাৎ Fractional বা Point থাকতে না,

(অর্থাৎ complete item use করতে হবে, না হয় পুরো ফেলে দিতে হবে, Dynamic Knapsack Problem এর ধরন)

2 টি Methods আছে,

① Bottom Up tabulation → এটি use হবে,

② Set Method.

#Bottom Up tabulation, Method using,

table এর Dimention = object × weight

Example

$m = 8$ (Capacity) $\quad P = \{1, 2, 5, 6\}$

$n = 4$ $\quad W = \{2, 3, 4, 5\}$

Object, Weight

P_i	W_i		0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	1	1	1	1	1
2	3	2	0	0	1	(2)	2	3	3	3	3
5	4	3	0	0	1	2	5	5	6	7	7
6	5	4	0	0	1	2	5	6	6	7	(8)

Highest Profit

row 0 থেকে start হবে, column 0 থেকে start হবে, m এর মান column এ বসবে।

তারপর 0 row এবং 0 column এ 0 দিয়ে ফিল-আপ করব,

m=8 হলো capacity of bag এর weight.

(4) Table এর ভিতর আমরা Profit গুলো বসাব, weight দিয়ে profit table এর মান বসাব,

Formula for the profit,

$$V[i,w] = \max\{V[i-1,w],\ W[i-1,\ w-w[i]] + p[i]\}$$

V এর মান (–) আসলে অবশ্যই বিয়োগ হবে,

Example solution: Highest Profit 8,

$x_1 \quad x_2 \quad x_3 \quad x_4$ → object গুলো,

$\{0 \quad 1 \quad 0 \quad 1\}$

Ans ⇒ $\{x_2, x_4\}$

Profit
8 − 6 = 2 remain
2 − 2 = 0
must see

(*) 3 নং object এ 2 থাকলেও, যেহেতু 2 নং object থেকে এসেছে,

(*) আর Total Profit টা বিয়োগ করে চেক করতে হয়।

Set Method,

Example

$n = 8$ $\quad P = \{1, 2, 5, 6\}$

$n = 4$ $\quad W = \{2, 3, 4, 5\}$

(P, W) আকারে হবে,

Highest weight 8 অর্থাৎ হবে,

$S^0 = \{(0, 0)\}$ ⟶ (First start)

$S^0_1 = \{(1, 2)\}$

$S^1 = \{(0, 0), (1, 2)\}$

$S^1_1 = \{(2, 3), (3, 5)\}$

$S^2 = \{(0, 0), (1, 2), (2, 3), (3, 5)\}$ → weight 9 হবে না

$S^2_1 = \{(5, 9), (6, 6), (7, 7), (8, 9)\}$

$S^3 = \{(0, 0), (1, 2), (2, 3), (3, 5), (5, 9), (6, 6), (7, 7)\}$

& $S^3_1 = \{(6, 5), (7, 7), (8, 8), (11, 9), (12, 11), (13, 12)\}$

$S^4 = \{(0, 0), (1, 2), (2, 3), (5, 9), (6, 6), (6, 5), (7, 7), (8, 8)\}$

↓ weight reduce করছি

Take it

This method generating all the possibilities.

Now, ① $(8,8) \in S^q$ ← qth object

But $(8,8) \notin S^3$ $\therefore x_q = 1$

$(8-6, 8-5) = (2,3)$

② $(2,3) \in S^3$

But $(2,3) \in S^2$ $\therefore x_3 = 0$

③ $(2,3) \in S^2$

but $(2,3) \notin S^1$ $\therefore x_2 = 1$

$(2-2, 3-1) = (0,0)$

④ $(0,0) \in S^1$

and $(0,0) \in S^0$ $\therefore x_1 = 0$

Solution: $\{0, 1, 0, 1\}$

বিদ্যমান আনুক্রমে S^1, S^2, S^3, S^4 কি সেট check হবে।

Graph:

A graph is a non-linear data structure consisting of vertices and edges.

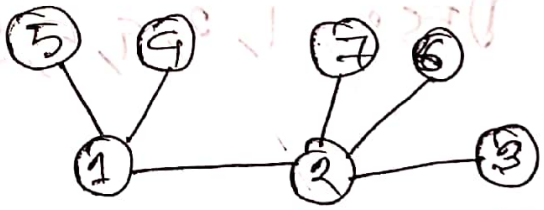
components of a graph → Vertices, Edges.

BFS and DFS

We have to traverse,

① visiting a vertex

② Exploration of vertex.



BFS: 1, 2, 4, 5, 7, 3, 6

DFS: 1, 2, 3, 6, 7, 4, 5

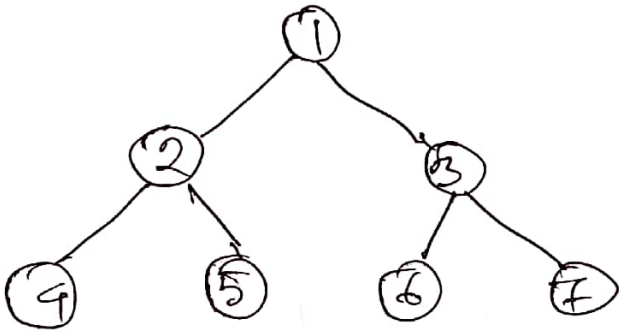
BFS can do any order.

BFS ~~BFS~~ first search vertex. Then we go to the next vertex.

DFS ~~DFS~~ new vertex visite ~~visite~~ suspend ~~suspend~~ start ~~start~~ explonation.

(*) DFS ~~DFS~~ vertex ~~vertex~~ connected না থাকলে Back ~~back~~ আসতে হয়।

Example:



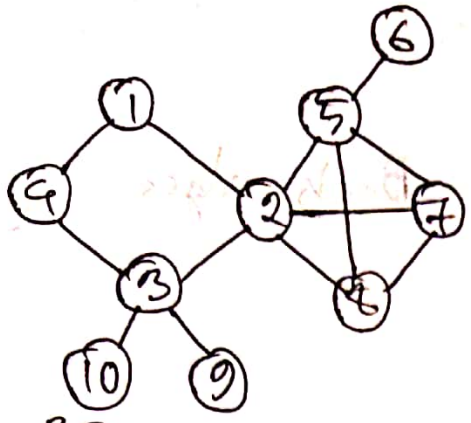
Binary tree

BFS: 1, 2, 3, 4, 5, 6, 7 (Level Order)

DFS: 1, 2, 4, 5, 3, 6, 7 (Pre Order)

(*) BFS and DFS time complexity: O(V+E)

Example:



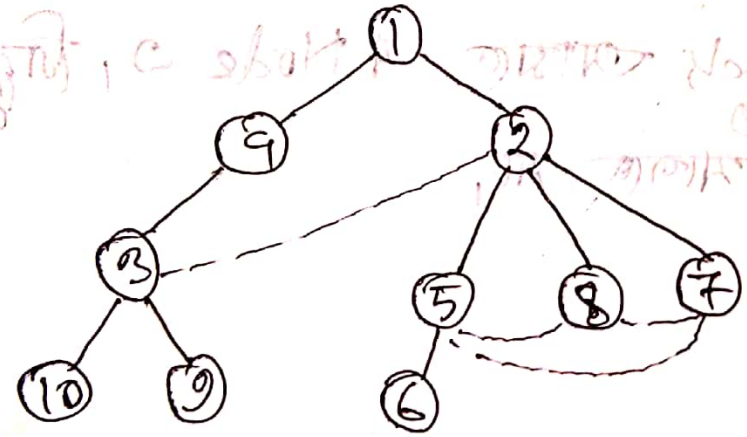
Here, Using BFS,

BFS = 1, 4, 2, 3, 5, 8, 7, 10, 9, 6

Queue Q =

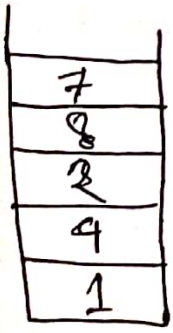
1	4	2	3	5	8	7	10	9	6

BFS spanning tree:



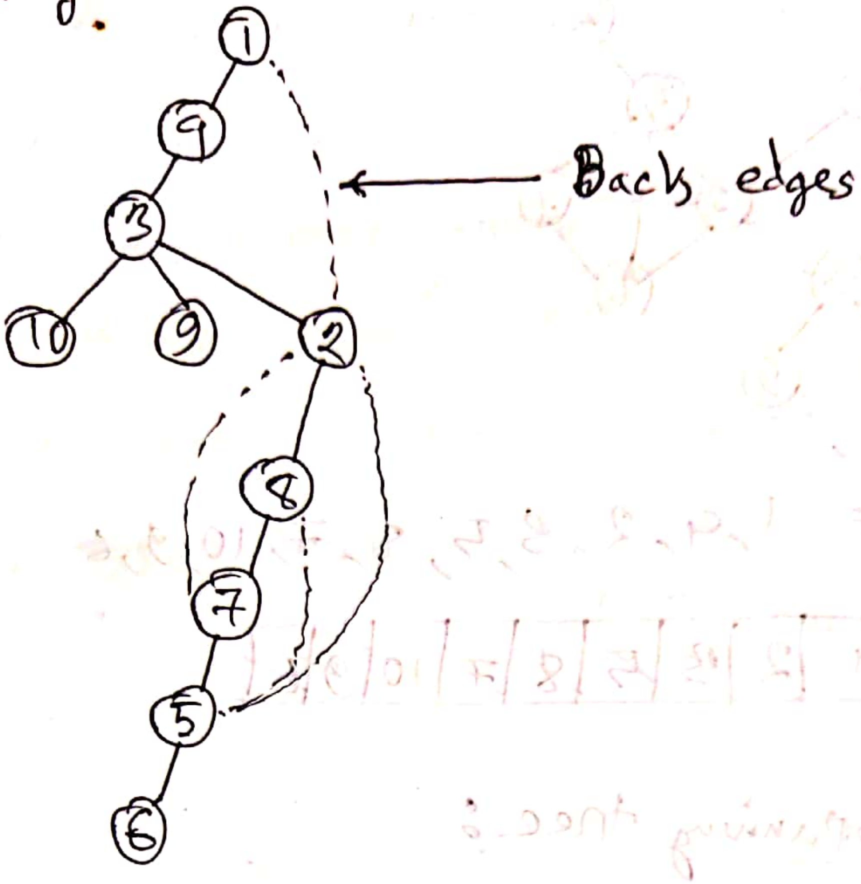
Using DFS,

DFS = 1, 4, 3, 10, 9, 2, 8, 7, 5, 6



Stack

DFS spanning tree,



⑤ DFS এ হলে Back আসতে Node ও, কিন্তু repeat value Any ও থাকতে না।

Chap-5

Matrix Chain Multiplication:

$A_1 \cdot A_2 \cdot A_3 \cdot A_4$

A matrix B matrix

row col row col

same multiply যাবে।

④ দুই ও তার অধিক Matrix multiply হবে,

$A \times B = C$

$5 \times 4 \quad 4 \times 3 \quad 5 \times 3$

$5 \times 3 \times 4 = 60$ multiplication

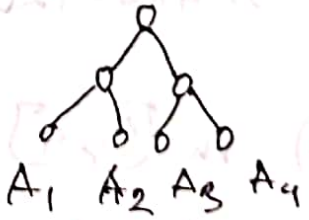
Example: $A_1 \cdot A_2 \cdot A_3 \cdot A_4$

$5 \times 4 \quad 4 \times 6 \quad 6 \times 2 \quad 2 \times 7$

we can multiply like this,

$((A_1 \cdot A_2) \cdot A_3) \cdot A_4$

$(A_1 \cdot A_2) \cdot (A_3 \cdot A_4)$



This problem now solved by dynamic problem method, (tabulation method):

m	1	2	3	4
1	0	120	88	158
2		0	48	104
3			0	84
4				0

s	1	2	3	4
1		1	1	3
2			2	3
3				3
4				

left

① $m[1,1]$

A_1

② $m[2,2]$

A_2

③ $m[3,3]$ ④

A_3

⑤ $m[1,2]$

$A_1 \cdot A_2$

$5 \times 4 \quad 4 \times 6$

$5 \times 4 \times 6 = 120$

⑥ $m[2,3]$

$A_2 \cdot A_3$

$4 \times 6 \quad 6 \times 2$

$4 \times 6 \times 2 = 48$

⑦ $m[3,4]$

$A_3 \cdot A_4$

$6 \times 2 \quad 2 \times 7 = 84$

⑧ $m[1,3]$

$A_1 \cdot (A_2 \cdot A_3)$
5×4 4×6 6×2

$= m[1,1] + m[2,3] + 5\times4\times2$

$= 0 + 48 + 40$

$= 88$ ✓

| $(A_1 \cdot A_2) \cdot A_3$
| 5×4 4×6 6×2
| $= m[1,2] + m[3,3] + 5\times6\times2$
| $= 120 + 0 + 60$
| $= 180$

④ min is 88

④ bracket এর inside যেটা থাকবে সেটা আগে করতে হয়, তাই $5\times4\times2$ এখানে 6 আগে বসান হয়েছে।

⑨ $m[2,4]$

$A_2 \cdot (A_3 \cdot A_4)$
4×6 6×2 2×7

$= m[2,2] + m[3,4] + 4\times6\times7$

$= 0 + 84 + 168$

$= 252$

| $(A_2 \cdot A_3) \cdot A_4$
| 4×6 6×2 2×7
| $= m[2,3] + m[4,4] + 4\times2\times7$
| $= 48 + 0 + 56$
| $= 104$

∴ minimum is 104

(10) $m[1,4] = \min\{ m[1,1] + m[2,4] + 5\times4\times7,$

$m[1,2] + m[3,4] + 5\times6\times7,$

$m[1,3] + m[4,4] + 5\times2\times7 \}$

$A_1 \cdot (A_2 \cdot A_3 \cdot A_4)$

$(A_1 \cdot A_2)(A_3 \cdot A_4)$

$(A_1 \cdot A_2 \cdot A_3) A_4$ ✓

$= \min\{ 0 + 104 + 140,\ 120 + 84 + 210,$

$88 + 0 + 70 \}$

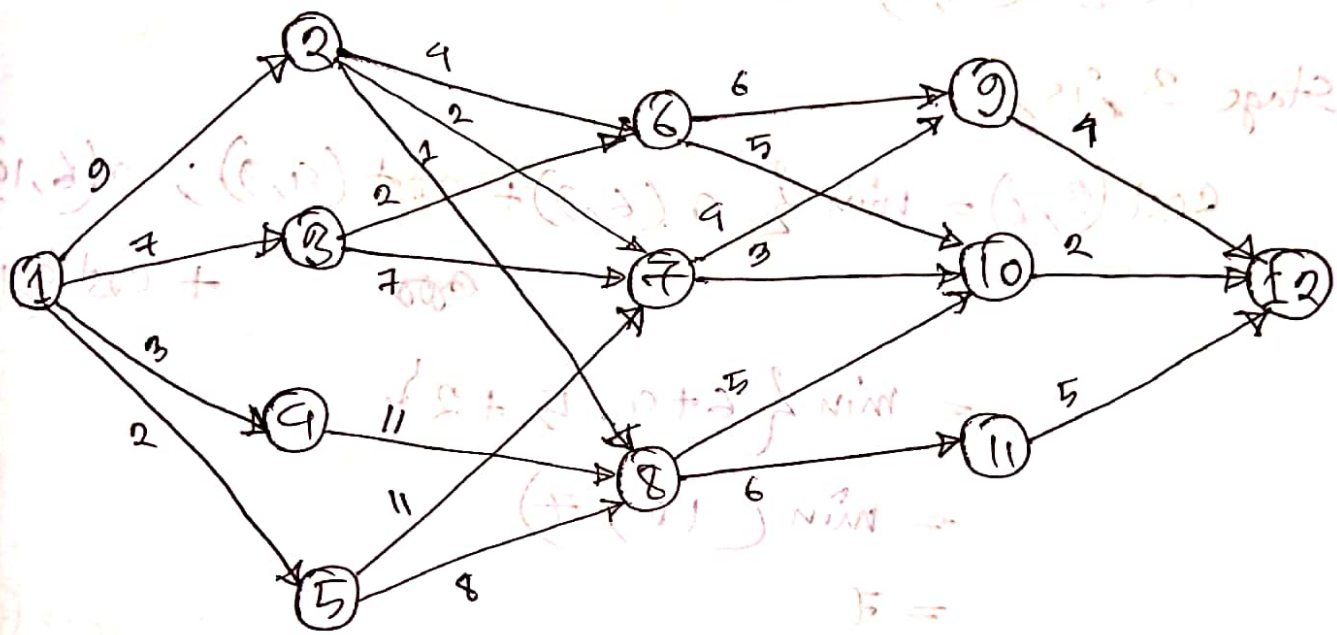
$= \min\{ 244, 414, \underline{158} \}$

$= 158$

∴ Minimum is 158.

Multistage Graph

Example: DP using tabulation Method,



V	1	2	3	4	5	6	7	8	9	10	11	12
cost	16	7	9	18	15	7	5	7	4	2	5	0
d	2/3	7	6	8	8	10	10	10	12	12	12	12

④ Graph এর সব বাম থেকে ডান দিকে আসতে হবে।

Here, $cost(5, 12) = 0$

(5 → stage Number, 12 → Vertex)

① Stage 4 is,

$cost(4,9) = 4$

$cost(4,10) = 2$

$cost(4,11) = 5$

② Stage 3 is,

$cost(3,6) = \min\{ c(6,9) + cost(4,9);\ c(6,10) + cost(4,10) \}$

$= \min\{6+4, 5+2\}$

$= \min(10, 7)$

$= 7$

$cost(3,7) = \min\{ c(7,9) + cost(4,9),\ c(7,10) + cost(4,10) \}$

$= \min\{4+4, 3+2\} = 5$

$cost(3,8) = \min\{ c(8,10) + cost(4,10),\ c(8,11) + cost(4,11) \}$

$= \min\{5+2, 6+5\} = 7$

(3) stage 2 is,

$cost(2,2) = \min\{c(2,6) + cost(3,6), c(2,7) + cost(3,7), c(2,8) + cost(3,8)\}$

$= \{4+7, 2+5, 1+7\} = 7$

$cost(2,4) = \min\{c(4,8) + cost(3,8)\}$

$= \min\{11+7\} = 18$

$cost(2,5) = \min\{c(5,8) + cost(3,8), c(5,7) + cost(3,7)\}$

$= \min\{8+7, 11+5\} = 15$

(4) stage 1 is,

$cost(1,1) = \min\{c(1,2) + cost(2,2), c(1,3) + cost(2,3), c(1,4) + cost(2,4), c(1,5) + cost(2,5)\}$

$= \min\{9+7, 7+9, 3+18, 2+15\}$

$= 16$ or 16

Using DP :

stage → $d(1,1) = 2$ ← destination, vertex → vertex

$d(2,2) = 7$

$d(3,7) = 10$

$d(9,10) = 12$

$\therefore$ cost $= 31$

vertex : $1 \to 2 \to 7 \to 10 \to 12$

shortest path