Algorithm
Sequence of unambigious instructions for solving a problem.

→ Eli's prime no.

```
for p ← 2 to n do
    A[p] ← p

for p ← 2 to n do
    if A[p] ≠ 0
        j = p * p
        while j <= n
            A[j] = 0
```

→ Sorting → selection

```
for i ← 0 to n-2
    min ← i
    for j ← i+1 to n-1
        if A[j] < A[min]
            min ← j
    swap A[i] to A[min]
```

→ search → binary

```
l = 0
h = n-1
while (l <= h)
    mid = (l+h)/2
    if x < a[mid]
        mid = h = m-1
    if x > a[mid]
        l = m+1
```

Time efficiency

$T(n) \approx C_{op}\, C(n)$

- $T(n)$ ↑ running time
- C_{op} ↑ Execution time
- $C(n)$ ↑ no. of tyms basic operation

Best, Worst, Average Cases

Example

10 20 30 40

If Key = 10

Best - 1
Worst - n
Average - n/2

In order to find time efficiency we have to take upper bound ignoring lower bound & constants

$n^3 + n^2 + 1$ (highest degree is upper bound)

Asymptotic notations

$O\, g(n)$ → Worst Case
$\Theta\, g(n)$ → Average Case
$\Omega\, g(n)$ → Best Case $f(n) > g(n)$

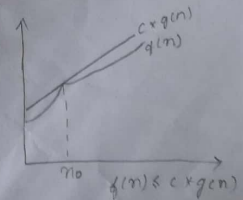
Average Case is calculated when worst & best case time efficiency is same

Types

1) Big-oh O notation

to find upper bound.

$f(n) = \{T(n)\}$ - running time
$g(n) = (C(n))$ - basic operation



then $f(n) \in O\, g(n)$

Example

(1) $f(n) = 3n+2$

$g(n) = n$, find $f(n) \in O\,g(n)$

$\Rightarrow f(n) \leq C * g(n)$.

C = 4 as LHS 3n. C will be 3+1

$3n+2 \leq C * n$

$3n+2 \leq 4 * n$ $(\because C = 4)$

$n = 1$ $\quad 5 \leq 4$ $\quad \times$

$n = 2$ $\quad 8 \leq 8$ $\quad \checkmark$

$n = n_0 \geq 2$

$\therefore C = 4$, $n_0 \geq 2$

then $3n+2 \in O(n)$

(2) $10n^2$ is in $O(n^2)$

$\Rightarrow f(n) = 10n^2$, $g(n) = n^2$.

W.K.T $f(n) \leq C * g(n)$.

$10n^2 \leq C * n^2$.

$10n^2 \leq 11n^2$ $\quad \because C = 11$

$n = 1$, $10 \leq 11$.

$n = n_0 \geq 1$

$\therefore C = 11$, $n_0 \geq 1$

then $10n^2 \in O(n^2)$

(3) $5n+20$ is in $O(n)$.

$f(n) = 5n+20$, $g(n) = n$.

$f(n) \leq c * g(n)$.

$5n+20 \leq c * n$.

$5n+20 \leq 6 * n$.

$n = 20$

$c = 6$, $n \geq 20$

$5n+20 \in O(n)$.

4) $100n+5$ g $\underline{n}$.

$f(n) = 100n+5$

$g(n) = n$

W.K.T $f(n) \leq c * g(n)$

$100n+5 \leq c * n$.

$100n+5 \leq 101n$.

$n=5$, $505 \leq 505$,

5) $10n^3+8 \leq c * n^3$.

$f(n) = 10n^3+8$.

$g(n) = n^3$

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

$10n^3+8 \leq c n^3$.

$10n^3+8 \leq 11n^3$.

$n=8$ $88 \leq 88$.

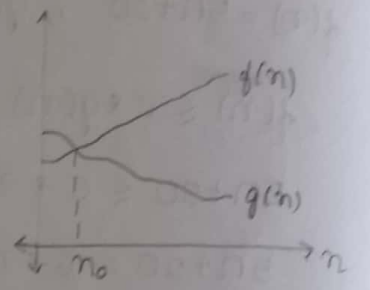
2) Big - Omega Ω

Lower bound - Best Case

$f(n) \geq c * g(n)$

$c > 0$

n



1) $3n+2 \in \Omega(n)$

$f(n) = 3n+2$, $g(n) = n$

$f(n) \geq c * g(n)$.

$3n+2 \geq c * n$

$3n+2 \geq 1 * 1$

$3(1)+2 \geq 1$

$5 \geq 1$

$3(2)+2 \geq 1*2$

$8 \geq 2$.

$n_0 \geq 2$

$2 \in \Omega(n)$

$c = 1$

$n_0 \geq 1$

2) $10n^2 \in \Omega(n^2)$

$f(n) \geq c * g(n)$.

$10n^2 \geq 1 * n^2$

$10 \geq 1$

$\therefore c = 1$ & $n \geq 1$

$10n^2 \in \Omega(n^2)$

3). $0.3n^2 - 2n \in \Omega(n^2)$

$f(n) \geq c * g(n)$ $\quad c = 0.1$

$0.3n^2 - 2n \geq c * n^2$ $\quad n_0 = 10$

$0.3(10)^2 - 2*10 \geq 0.1 * (10)^2$

$10 \geq 10$

$c = 0.1 , n_0 \geq 10$

4) $0.1n^3 \in \Omega(n^2)$

$f(n) \geq c * g(n)$ $\quad c = 0.1$

$n \geq 1$

$0.1n^3 \geq 0.1 * n^2$

$0.1 * 1 \geq 0.1 * 1$

$0.1 \geq 0.1$

$c = 0.1 , n_0 \geq 1$

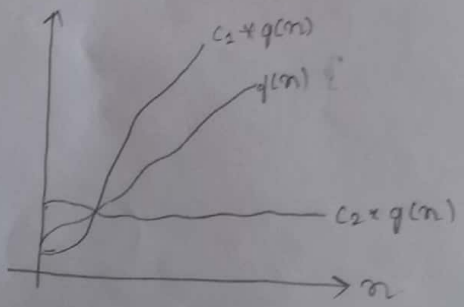
3) Big-theta Θ

$c_2 * g(n) \leq f(n) \leq c_1 * g(n)$

$c_1 > 0$

$c_2 > 0$

$n \geq 1$



Example

$3n+2 \in \Theta(n)$

$c_2 * n \leq 3n+2 \leq c_1 * n$ $\quad c_1 = 4$

$1*1 \leq 3(2)+2 \leq 4*2$ $\quad n_0 \geq 2$

$2 \leq 8 < 8$ $\quad c_2 = 1$

$n_0 \geq 1$

$\therefore c_1 = 4 , c_2 = 1 , n_0 \geq 2$

Order of growth

$\lim f(n)/g(n)$, $n \to \infty$

(LH rule

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

1) $O = f(n)$, O Worst

2) $C = f(n)$, θ Average

3) $\infty = f(n)$, Ω Best

Example

1) $10n$, n^2.

$f(n) = 10n$

$g(n) = n^2$.

$\lim_{n \to \infty} f(n)/g(n)$.

$\lim_{n \to \infty} \frac{10n}{n^2}$.

$\lim_{n \to \infty} 10/n$.

$10 \lim_{n \to \infty} (1/n)$

$10 (1/\infty) = 0$, worst Case.

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

```
#include <stdio.h>
void main()
{
  int fact = 1, i, n;
  clock_t start, end, total
  printf("enter value of n \n");
  scanf("%d", &n);
  start = clock();
 for(i=1; i<=n; i++)
    fact = fact * i;
    delay(100);
    end = clock();
 printf("factorial of a no. is %d", fact);
  total = (end - start) / (double) CLOCKS_PER_SEC
 printf("total time taken is %g", total);
 }
```

2) $n(n+1)/2$ vs n^2

$f(n) = n(n+1)/2$.

$g(n) = n^2$

$\lim_{n\to\infty} f(n)/g(n)$.

$$\lim_{n\to\infty} \frac{\frac{n(n+1)}{2}}{n^2}$$

$$\lim_{n\to\infty} \frac{\frac{n+1}{2}}{n}$$

$$\frac{1}{2} \lim_{n\to\infty} \frac{n+1}{n}$$

$$\frac{1}{2} \lim_{n\to\infty} \frac{n}{n} + \frac{1}{n}$$

$$\frac{1}{2} \lim_{n\to\infty} \left(1 + \frac{1}{n}\right)$$

$$\frac{1}{2} [1 + 1/\infty]$$

$$\frac{1}{2} [1 + 0]$$

$\frac{1}{2}$ = Constant

$\therefore f(n) \in \Theta g(n)$.

$\frac{n(n+1)}{2} \in \Theta(n^2)$.

3) $\log_2 n$ & $\sqrt{n}$

$f(n) = \log_2 n$

$g(n) = \sqrt{n}$

$\lim_{n \to \infty} f(n)/g(n)$

$\lim_{n \to \infty} \frac{\log_2 n}{\sqrt{n}}$ $\qquad \log_a b = \frac{\log_e b}{\log_e a}$

$\lim_{n \to \infty} \frac{\frac{\log_e n}{\log_e 2}}{\sqrt{n}}$

$\frac{1}{\log_e 2} \lim_{n \to \infty} \frac{\log_e n}{\sqrt{n}}$ $\qquad \frac{1}{\log_e a} = \log_a e$

$\log_2 e \lim_{n \to \infty} \frac{\log_e n}{\sqrt{n}}$

$\log_2 e \lim_{n \to \infty} \frac{\frac{1}{n}}{\frac{1}{2\sqrt{n}}}$

$\log_2 e \lim_{n \to \infty} \frac{1}{n} \times \frac{2\sqrt{n}}{1}$

$\log_2 e \lim_{n \to \infty} \frac{1}{\sqrt{n} \cdot \sqrt{n}} \times \frac{2\sqrt{n}}{1}$ $\qquad n = \sqrt{n} \times \sqrt{n}$

$\log_2 e \lim_{n \to \infty} \frac{1}{\sqrt{n}} \times \frac{2}{1}$

$2 \log_2 e \, (1/\sqrt{\infty}) = 0$. Worst Case

$f(n) = O g(n)$

Analysis of recursive algorithms

Steps to find

(1) Factorial

$t(n) = 0$ if $n = 0$

$t(n) = 1 + t(n-1)$ if $n \geq 1$

$t(n) = 1 + t(n-1) \longrightarrow (1)$

replace n by $n-1$

$= 1 + [1 + t(n-1-1)]$

$= 1 + [1 + t(n-2)]$

$= 2 + t(n-2) \longrightarrow (2)$

$= 2 + [1 + t(n-1-2)]$

$= 3 + t(n-3) \longrightarrow (3)$

$= 3 + [1 + t(n-1-3)]$

$= 4 + t(n-4) \longrightarrow (4)$

$= n + t(n-n)$

$= n + t(0)$

$= n + 0$

$t(n) = n$

Algorithm : TH (n,s,t,d)

purpose : move disc from source to destination only 1 disc is moved. small disc is placed above the large

Input : n (no. of discs)

output : all the discs should be places on the destination tower basic operation

Step 1 : if (n==1) move disc from s to d
else if (n>=1) move disc from s to d t
{n-1, s, d, t}

move disc from s to d
{n, s, d}

move disc from t to d
{n-1, t, s, d}

end.

Analysis of tower of hanoi

(1) Read the input → n

(2) identify basic operation → moving disc from source to destination

(3) solve using recurrence eqn

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

t(n-1) → s to t, 1 → s to d, t(n-1) → t to d

Time effeciency

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

$= 2t(n-1) + 1 \longrightarrow ①$

replace n by $n-1$.

$= 2[2t(n-2) + 1] + 1$

$= 2^2[2t(n-1-2)+1] + 2 + 1$

$= 2^3 t(n-3) + 2^2 + 2 + 1$

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

$= 2^4 t(n-4) + 2^3 + 2^2 + 2 + 1$

$= 2^i t(n-i) + 2^{i-1} + 2^{i-2} + \ldots + 1$

Let $n-i = 1 \longrightarrow ②$

$2^i t(1) + 2^{i-1} + 2^{i-2} + \ldots + 1$

↓base case

$2^i * 1 + 2^{i-1} + 2^{i-2} + \ldots + 1$

$2^i + 2^{i-1} + 2^{i-2} + \ldots + 1$.

W.K.T $n - i = 1$.

$i = n - 1$.

$2^{n-1} + 2^{n-1-1} + 2^{n-1-2} + \ldots + 1$

$2^{n-1} + 2^{n-2} + 2^{n-3} + \ldots + 1$.

$\Rightarrow 2^n$

$\therefore T(n) \in \Theta(2^n)$.

Time efficiency of nonrecursive algorithms

input size
basic operation
worst, best, average case.
sum of number of times
substitution method

$$\sum_{i=1}^{n} 1 = n - 1 + 1 = n$$ ← formula. (upper lim - lower lim + 1)

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \dots n = \frac{n(n+1)}{2} = \frac{n^2+n}{2} \approx \frac{n^2}{2}$$

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{n^3}{3}$$

$$\sum_{i=1}^{n} a^i = (a^{n+1} - 1) / (a-1).$$

$$\sum_{i=1}^{n} 2^i = 2^{n+1} - 1 / 2 - 1 = 2^{n+1} - 1$$

Example 1 : Max element

. input - n

. basic operation - comparison

. time efficiency

. ALGORITHM MaxElement(A[0..n-1])

//Determines the value of largest element in the given array

//input : An array A[0..n-1] of real numbers

//output : the value of largest element in A

maxval ← A[0]

for i ← 1 to n-1 do

if A[i] > maxval

maxval ← A[i]

return maxval.

. substitution method.

$\sum_{i=1}^{n-1} 1$ → comparison

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

$\Rightarrow n - 1$.

$\Rightarrow n$

best, average, worst case are same for maximum element.

$T(n) \in \Theta(n)$.

$T(n) \in O(n)$.

Example 2 Element uniqueness problem.

Input size

basic operation - comparison

Algorithm

ALGORITM Unique elements (A[0...n-1])

//Determines whether all elements in given array

//input : an array A[0...n-1].

//output : returns true if all elements in A // are distinct & false otherwise

for i ← 0 to n-2 do

for j ← i+1 to n-1 do

if A[i] = A[j] return false

return true.

Substitution method

$$T(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 \quad \text{→ 1 if statement so 1}$$

$$= \sum_{i=0}^{n-2} n-1-(i+1)+1.$$

$$= \sum_{i=0}^{n-2} n-1-i-1+1$$

$$= \sum_{i=0}^{n-2} n-1-i$$

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

$$= n-1+n-2+n-3+1.$$

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

$T(n) = n^2$

$\therefore T(n) = O(n^2)$ worst & average case same

$= \Omega(n^2)$ best case is 1

Example 3 Sequential search / Linear

Input size - array of n elements.
basic operation - comparison.
Algorithm

ALGORITHM Sequential search(A[..n-1], k)

// searches for a given value in a given array by sequential search.
// input : An array A[0..n-1] & a search key K.
// output : The index of the first element of A that matches K or -1 if there are no matching elem

```
i ← 0
while i<n and A[i] ≠ K do
    i ← i+1
if i<n return i
else return -1
```

substitution method.

$$T(n) = \sum_{i=0}^{n-1} 1 \quad \rightarrow \text{only 1 if statement}$$

$$= n-1-0+1$$

$$T(n) = n$$

$T(n) \in O(n)$ - $\Theta(n)$:
worst — Average → n or (n+1)/2

$\Omega(n)$ (Best Case) — 1

Example 4 : Factorial problem.

input size

basic operation - multiplication

Algorithm

ALGORITHM

Step 1 : start

Step 2 : declare variables n, factorial & i

Step 3 : initialize variables.

factorial ← 1

i ← 1

Step 4 : read value of n

Step 5 : repeat steps until i = n

factorial ← factorial * i

$T(n) = \sum_{i=1}^{n-1} 1$

$= n - 1 + 1$

$= n.$

$T(n) \in O(n)$

$T(n) \in \Theta(n)$

$T(n) \in \Omega(n)$

Sieve of Eratosthenes.

algorithm

input : integer $n \geq 2$.

starts with 2 because 0 & 1 are composite no.

the is done to get prime no.

$p \leftarrow 2$ to $\sqrt{n}$

2 to $\sqrt{20}$ 2 to 5

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	2	3	4	5														
2	3	0	5	0	7	0	9	0	11	0	13	0	15	0	17	0	19	0
							0			0			0			0		

$\sqrt{20} = 4 \ldots = 5$

5

→ 2 multiples 4, 6 removing multiple of 2 & making 0

Time complexity : $O(n)$.

Module 02

1) Selection sort

Algorithm

```
for i ← 0 to n-2
    min ← a[i]
for j ← i+1 to n-1
    if [a[j] < a[min])
        min ← j
        swap a[i], a[min]
```

Analysis

$$T(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1.$$

$$= \sum_{i=0}^{n-2} (n-1) - (i+1) + 1$$

$$= \sum_{i=0}^{n-2} n - 1 - i - 1 + 1$$

$$= \sum_{i=0}^{n-2} n - 1 - i$$

$$= (n-1-0) + (n-1-1) + (n-1-2) + \cdots (n-1-(n-2))$$

$$= (n-1) + (n-2) + (n-3) + \cdots (n-1-n+2)$$

$$= (n-1) + (n-2) + (n-3) + \cdots + 1$$

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

$$= \frac{n^2+n}{2} \Rightarrow \frac{n^2}{2} + n/2.$$

$$= n^2.$$

$\therefore T(n) \in \Theta(n^2)$

$T(n) \in O(n^2)$.

1) Sequential search/Linear search

eg- 10 | 20 | 35 | 40 |
i → a[0] a[1] a[2] a[3]

key = 40

40 = = a[0] = 10
i increment
i=1 40 = = a[1] = 20 x
i=2 40 = = a[2] = 35 x
i=3 40 = = a[3] = 40 ✓

Algorithm

S1 : // Input → array of n element & key element

S2 : // output → successful search (key found).
→ Unsuccessful search (key nt found)

S3 : basic operation

```
for i ← 0 to n-1
    if (key == a[i])
        return 1;
end for
return -1
```

```
for(i=0 ; i<=n-1 ; i++)
{
    if(key = = a[i])
        Pl ("found")
        break;
Pl ("nt found"),
```

Analysis : Time complexity

i) Best Case : $T(n) \in \Omega(1)$

ii) Worst Case :

$$t(n) = \sum_{i=0}^{n-1} 1$$

$$= n-1-0+1$$

$$t(n) = n$$

$$t(n) \in O(n)$$

iii) Average case : (Probability)

$$p + q = 1$$

$$q = 1-p$$

Let P → probability of success
1-P → " failures

Probability for successful search

P/n

$= [1 * P/n + 2 * P/n + 3 * P/n + \dots n * P/n]$

$= \frac{P}{n} [1 + 2 + 3 + \dots + n]$.

$= \frac{P}{n} \left[\frac{n(n+1)}{2}\right]$

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

Probability for unsuccessful search

$n * (1-P)$

Probability for successful & unsuccessful

$\text{Prob} = \frac{P(n+1)}{2} + n * (1-P)$

If P=1, successful

$t(n) = \frac{P(n+1)}{2} + n(1-P)$

$t(n) = \frac{n+1}{2} + 0$

$t(n) = \frac{n+1}{2}$, $\therefore t(n) \in \Theta(n+1)/2$.

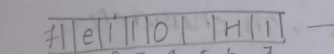
If P=0, failure

$t(n) = \frac{P(n+1)}{2} + n(1-P)$

$t(n) = n$

$\therefore t(n) \in \Theta(n)$

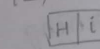
String Matching



text → n

pattern → m

Hi → m



H = H

P[j] = t[i+j]

P[i] = t[0+1]

i = 0 < j = 0, j = 1 ; i++

e = i X

outer loop

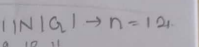
i = 1

e = H X

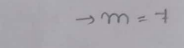
i = 2 . . . . . i = 5 H = H ; increment j

j =

Algorithm



n = 12



m = 7

i → 0 to 5

j → 0 to 6

G O O d _ m o r n i n g

M o r n i n g G ≠ M

M o r n i n g

unit G = G

after matching increment j value

0=0 r=r n=n i=n n=n j=j

	text i	i=5	pattern j
M	t[5]	= t[5+0]	P[0]
O	t[6]	= t[5+1]	P[1]
R	t[7]	= t[5+2]	P[2]
N	t[8]	= t[5+3]	P[3]
I	t[9]	= t[5+4]	P[4]
N	t[10]	= t[5+5]	P[5]
G	t[11]	= t[5+6]	P[6]

Algorithm

//input : Read text → n characters
Read pattern → m characters

//output : if j=m, return i //returning position of 1st character the pattern.
else return -1 // pattern not present

Basic operation :

```
for i←0 to n-m
    j←0
    while (j<m) & P[j] = = t[i+j]
        j←j+1
    end while
        j=m, return i
end for
    j≠m, return -1
```

```
for(i=0; i<n-m; i++)
    for(j=0; j<m; j++)
        P[j] = t[i+j]
```

{ i is used shift the position of pattern
j is used to compare character of pattern & text }

i) i=0, 0<5
j=0, 0<7
M=G ✗

ii) i=1, 1<5
j=0, 0<7
O=M ✗

iii) i=2, 2<5
j=0 0<7
O=M

iv) i=3, 3<5
i=0, 0<7
O=M ✗

v) i=4, 4<5
i=0, 0<7
O=M ✗

vi) i=5, 5<5
i=0, 0<7
M=M ✓
i, j=1 1<7
O=O ✓

j=2, 2<7
R<7
7=S, 3<7
n=n
7=U, 4<7
i=1
7=E, 5<7
A=A
7=G, 6<7
9=9
7=1, 7<7

Time effeciency

$$t(n) = \sum_{i=0}^{n-m} \sum_{j=0}^{m-1} 1$$

$$= \sum_{i=0}^{n-m} m-1-0+1$$

$$= \sum_{i=0}^{n-m} m-1$$

$$= m-1 \sum_{i=0}^{n-m} 1.$$

$$= (m-1)\,[(n-m)-0]$$

$$= (m-1)\,(n-m).$$

$$= mn - m^2 - m + m.$$

$$= mn.$$

$t(n) \in O(mn)$

$t(n) \in \Theta(mn)$.

Strengthes & weakness of Brute force.

strengthes
- simplicity
- widely applicable
- matrix multiplication, factorial, bubble, selection sort,

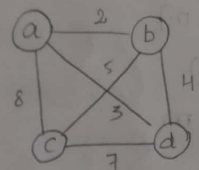
weakness.
- time efficiency less.
- space efficiency less.

Exhaustive search

for a given problem, searching best solution (optimal solution) in a given n no. of solution for a given problem is called exhaustive search.

example

1. travelling sales person problem. (minimum)



Visiting all the nodes from the source to neighbourhood nodes coming back to source.

1) $a \xrightarrow{2} b \xrightarrow{4} d \xrightarrow{7} c \xrightarrow{8} a = 21$

2) $a \xrightarrow{2} b \xrightarrow{5} c \xrightarrow{7} d \xrightarrow{3} a = 17$

3) $a \xrightarrow{8} c \xrightarrow{7} d \xrightarrow{4} b \xrightarrow{2} a = 21$

4) $a \xrightarrow{8} c \xrightarrow{5} b \xrightarrow{4} d \xrightarrow{3} a = 20$

5) $a \xrightarrow{3} d \xrightarrow{7} c \xrightarrow{5} b \xrightarrow{2} a = 17$

6) $a \xrightarrow{3} d \xrightarrow{4} b \xrightarrow{5} c \xrightarrow{8} a = 20$

feasible solution

Optimal solution

2) $a \rightarrow b \rightarrow c \rightarrow d \rightarrow a$

5) $a \rightarrow d \rightarrow c \rightarrow b \rightarrow a$

(4-1)! = 6.

$T(n) \in \Theta(n-1)!$

2) Knapsack problem.

capacity $w = 16$.

item	weight	value
1	2	$20
2	5	$30
3	10	$50
4	5	$10

→

item	weight	profit
{1}	2	$20
{2}	5	$30
{3}	10	$50
{4}	5	$10
{1,2}	7 (2+5)	$50
{1,3}	12	$70
{1,4}	7	$30
{2,3}	15	$80
{2,4}	10	$40
{3,4}	15	$60
{1,2,3}	17	not feasible
{1,2,4}	12	$60
{2,3,4}	20	nt feasible
{1,3,4}	17	not feasible
{1,2,3,4}	22	not feasible.

best soln.
max. - 80
{2,3}
15

$T(n) \in \Theta(2^n - 1). = \in \Theta 2^n.$

MODULE 03

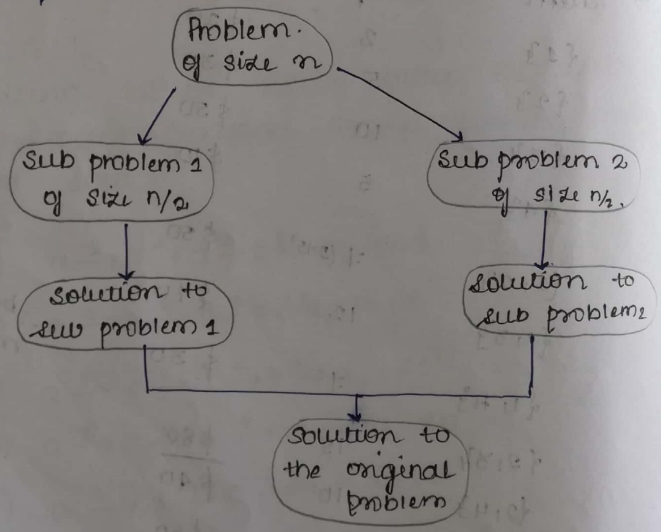
DIVIDE AND CONQUER

(1) divide instance of problem into two or more smaller instances

(2) solve smaller instances recursively

(3) Obtain solution to original instance by combining these solutions

Technique



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

Divide & conquer - examples

Sorting : mergesort & quicksort
Binary tree traversals
Binary search(?)
multiplication of large integers.
matrix multiplication : strassen's algorithm

General divide & conquer Recurrence

$$T(n) = a\,T(n/b) + f(n)$$

- a: number of substances.
- $T(n/b)$: time for size (n/b)
- $f(n)$: time required to divide the problem into subproblem

where $f(n) \in \Theta(n^d)$, $d \geq 0$

Masters theorem

$$T(n) = a\,T(n/b) + \Theta(n^k \log^p n)$$

$a \geq 1$, $b > 1$, $k \geq 0$, p any real no.

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

(ii) if $a = b^k$
- (a) if $p > -1$ then $T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$
- (b) if $p = -1$ then $T(n) = \Theta(n^{\log_b a} \log \log n)$
- (c) if $p < -1$ then $T(n) = \Theta(n^{\log_b a})$

(iii) if $a < b^k$
- (a) if $p \geq 0$, then $T(n) = \Theta(n^k \log^p n)$
- (b) if $p < 0$, then $T(n) = \Theta(n^k)$

Solve given problem using masters theorem

a) $T(n) = 3T(n/2) + n^2$

$T(n) = aT(n/b) + \theta(n^k \log^p n)$

$a = 3 \quad a \geq 1$

$b = 2 \quad b > 1$

$k = 2 \quad k \geq 0$

$p = 0$ (by default)

$a \not\geq b$

$a = 3, \quad b^k = 2^2 = 4.$

$\rightarrow a < b^k.$

$p = 0$

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

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

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

b) $T(n) = 4T(n/2) + n^2$

$a = 4$

$b = 2,$

$k = 2,$

$p = 0$

$a = 4, \quad b^k = 2^2 = 4.$

$a = b^k.$

$p > -1, \quad T(n) = \theta(n^{\log_b a} \log^{p+1} n).$

$T(n) = \Theta(n^{\log_2 3} \log^2 n)$

$= \Theta(n^{\log_2 4} \log n)$

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

c) $T(n) = 4T(n/2) + n$

$T(n) = a\,T(n/b) + \Theta(n^k \log^p n)$.

$a = 4$

$b = 2$

$k = 1$

$p = 0$

$a = 4$, $b^k = 2^1 = 2$

$a > b^k$

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

$= \Theta(n^{\log_2 4})$

$T(n) = \Theta(n^2)$

d) $T(n) = 4T(n/2) + n^3$

$T(n) = a\,T(n/b) + \Theta(n^k \log^p n)$

$a = 4$

$b = 2$

$k = 3$

$p = 0$

$a = 4$, $b^k = 2^3 = 8$

$a < b^k$.

$T(n) = \Theta(n^{\log_b^a} \log^{p+1} n)$.

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

$= \Theta(n^2 \log n)$ //.

$T(n) = \Theta(n^k \log^p n)$.

$= \Theta(n^3 \log^0 n)$.

$= \Theta(n^3)$.

e). $T(n) = T(n/2) + n^2$.

$T(n) = a\,T(n/b) + \Theta(n^k \log^p n)$.

$a = 1$

$b = 2$.

$k = 2$

$p = 0$

$a = 1 \quad b^k = 4$.

$a < b^k$.

$T(n) = \Theta(n^k \log^p n)$.

$= \Theta(n^2 \log^0 n)$

$= \Theta(n^2)$ //.

2)

Selection Sort – defn → sorting by taking minimum

Algorithm

time - analysis – using substitution method

3) same as 2

defn searching key element from array of n elements

trauns → 10 20 30 40

key - 30 -

Analysis frame work

time effi.

Space →

best, worst, average.

Order of growth

Part A - mcq's 1x10 = 10m

Part B thought provoking question 3→6, 2x7=14 =)20 m

Part C

non-recursive, recursive, asymptotic notation

defn, algo, efficiency.

Merge Sort

Example

low mid mid+1 high

8 3 2 9 7 1 6 4

0 1 2 3 4 5 6 7

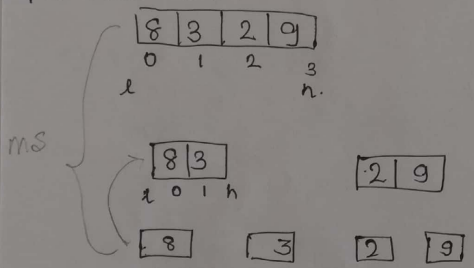
if (low < high) → mid = low + high/2

Call mergesort (A, low to mid)

mergesort (A, mid to high)

merge (A, low, mid, high)

post order - L R ; R



Algorithm merg(A[0..n-1])

I/p: A[0..n-1] recursive mergesort

A[0...n-1] Orderable elements.

A[0..n-1] Sorted in non decreasing

if n > 1

copy A[0... (n/2)-1] to B[0..

copy A[(n/2)... n-1] to C[

mergesort B[0..(n/2)-1]

mergesort C[0..(n/2)-1]

merge (B, C, A)

```
merge (A, low, mid, high)
{
  i ← low , j ← mid+1 , k ← low
  while (i <= m) && (j <= n)
  {
    if (a[i] <= b[j])
    {
      c[k] ← a[i]
      k++
      i++
    }
    else
    {
      c[k] ← b[j]
      k = k+1
      j = j+1
    }
  }

  while (i <= m)
  {
    c[k] = a[i]
    k = k+1
    i = i+1
  }
  while (j <= n)
  {
    c[k] = b[j]
    k = k+1
    j = j+1
  }
```

m ⇒ mid.
n ⇒ high

8	3
0	1

A

2	9
2	3

B

⇒ resultant C.

0	

K.

while (i < m) && (j < n)
i < 2
j = mid + 1

a[0] ≤ b[2]
if (8 ≤ 2) X
{
}
else
C[K] ← b[j].
C[0] ← b[2].
C[0] ← 2.
j++
K++.

C	2	
K.	0	

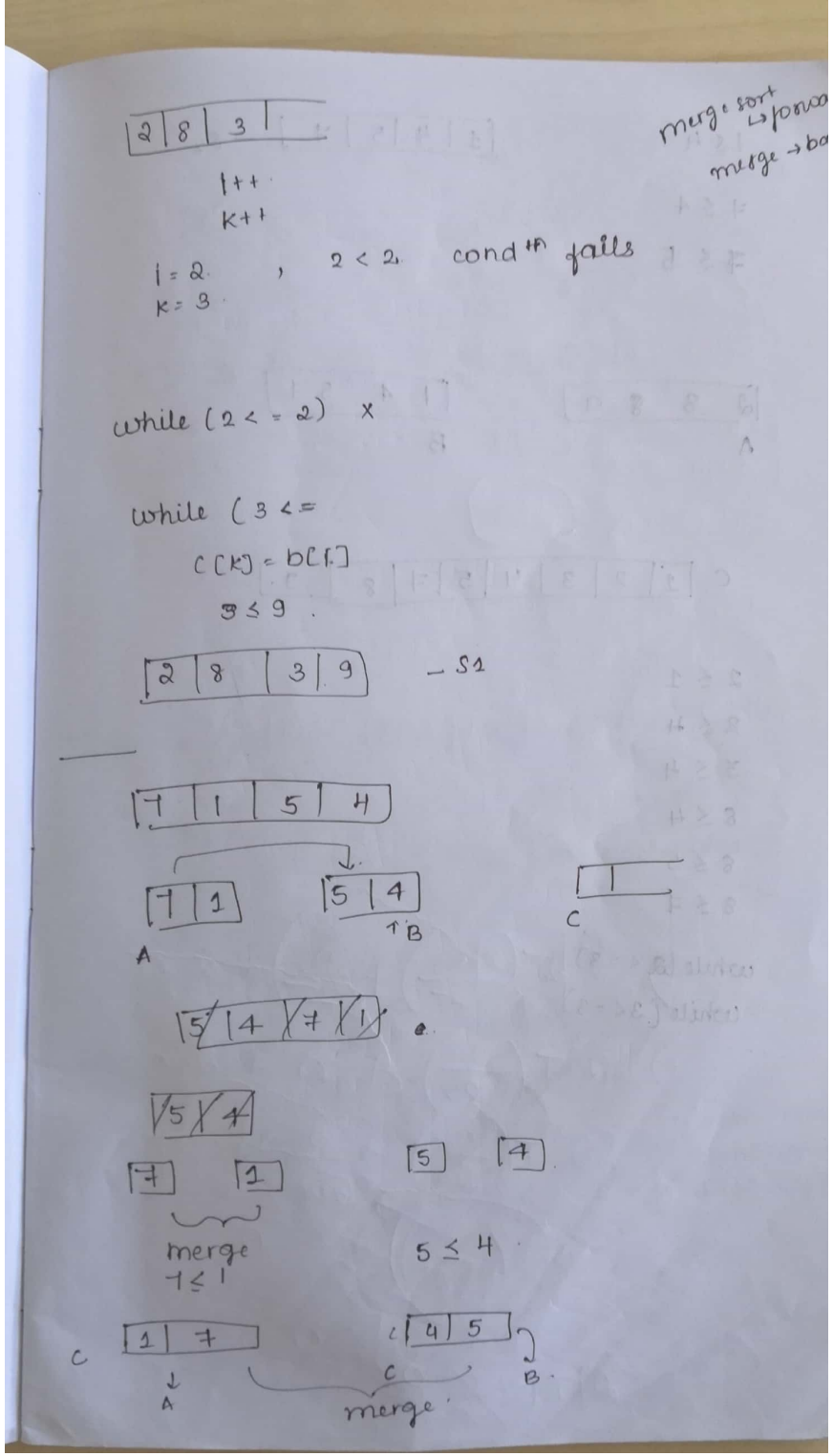
i = 3
K = 1

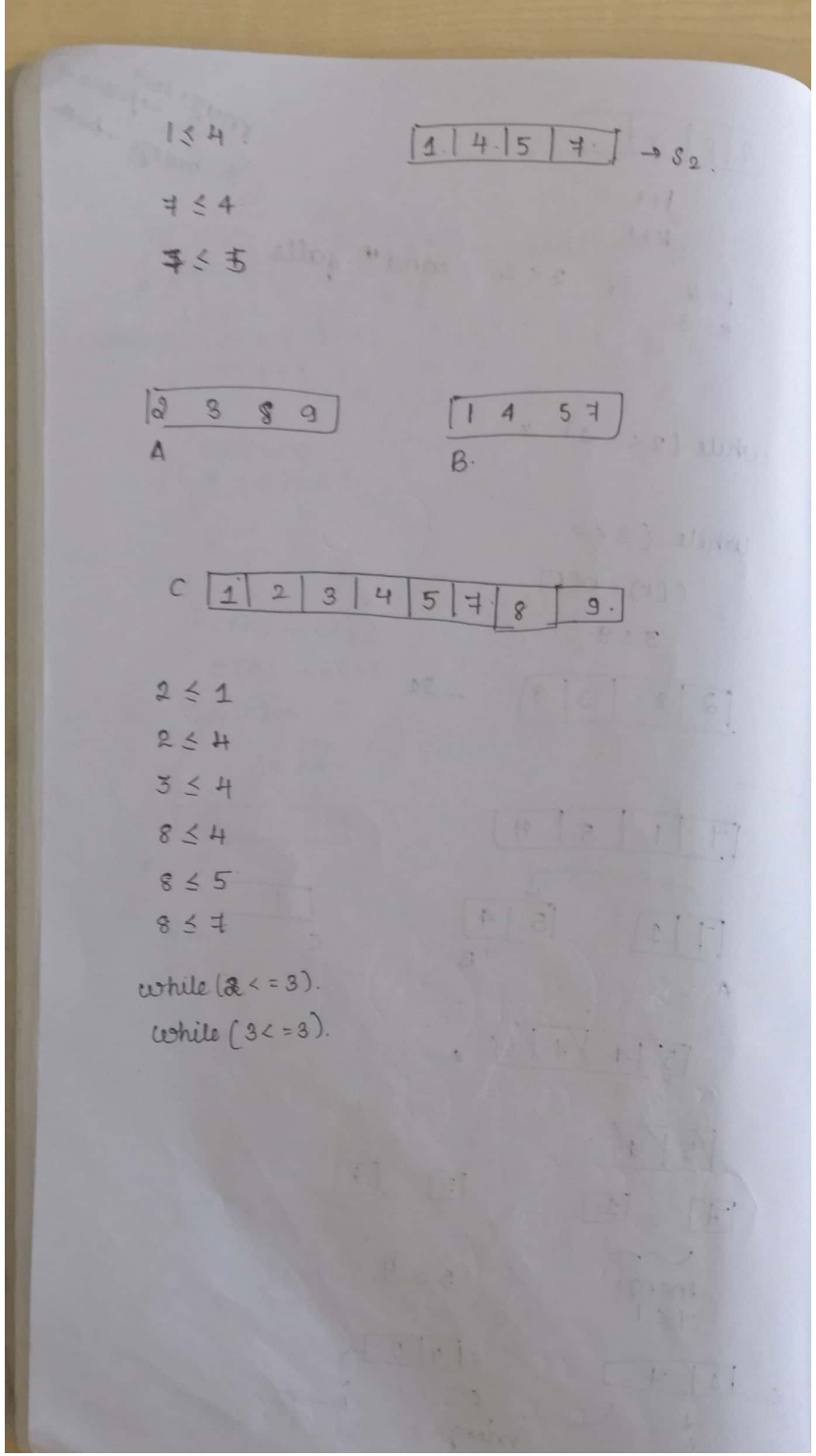
8 ≤ 9 ✓
{
C[1] ← a[0].
C[1] ← 8

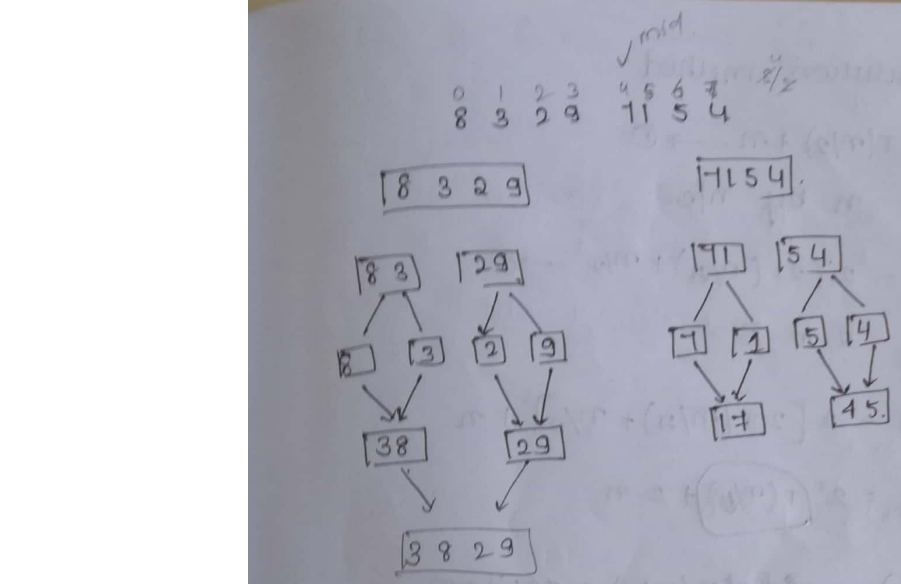
2	8	

i++
K++.

i = 1
K = 2.
3 ≤ 9
C[2] ← a[1]







Time efficiency for merge sort

$$T(n) = \begin{cases} T(1) = 0 & \text{if } n = 1 \\ T(n/2) + T(n/2) + n & \text{otherwise} \end{cases}$$

$$T(n) = 2T(n/2) + n \longrightarrow (1)$$

by masters method.

$$T(n) = a\,T(n/b) + \theta(n^k)$$

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

$a = 2$, $b^k = 2^1$

$\therefore T(n) = \theta(n \log_2 n)$

$$\begin{cases} \text{if } a > b^k = \theta(n) \\ a = b^k \Rightarrow \theta(n \log n) \\ a < b^k \Rightarrow \log n \end{cases}$$

by substitution method

$T(n) = 2T(n/2) + n \longrightarrow ①$

Replace n by $n/2$.

$T(n/2) = 2[2T(n/4) + n/2 \longrightarrow ②$

② in ①

$T(n) = 2[2T(n/4) + n/2] + n$

$= 2^2 T(n/2^2) + 2 \cdot n$

~~$T(n/4) = 2^2 \cdot 2[2^2 T(n/2^3) + 2n/2^2] + n$~~

~~$T(n/4) = 2^3 T(n/2^3) + 2n + 2n$~~

$= 2^3[2T(n/2^3) + n/4] + 2n$

~~$= 2^3 T(n/2^3) + 3n$~~

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

$\Rightarrow 2^2\left[2T\left(\frac{n/2}{2^2}\right) + \frac{n}{2^2}\right] + 2 \cdot n$

$\Rightarrow 2^3 T\left(\frac{n}{2^3}\right) + n + 2 \cdot n$

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

$= 2^i T\left(\frac{n}{2^i}\right) + i \cdot n$ → Base case.

Put $2^i = n$.

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

$T(n) \Rightarrow i \cdot n \longrightarrow (4)$

W.K.T $2^i = n$

log o.b.s

$\log 2^i = \log n$

$i \log 2 = \log n$

$i = \frac{\log n}{\log 2}$

$i = \log_2 n \rightarrow (5)$

(5) in (4)

~~$T(n) = \log_2 n * n$~~

$T(n) = O(n * \log_2 n)$

Quick Sort

low							high
50	30	10	90	80	20	40	70
0	1	2	3	4	5	6	7

pivot

↳ left side of pivot element shd be less
right side of —u— shd be greater

i = low
j = high = n-1

while (a[i] ≤ a[pivot])
    i++

while (a[j] ≥ a[pivot])
    j--

if (i < j)
    swap a[i] & a[j]

else
    swap a[j] & a[pivot].

1st case.

while (a[i] ≤ a[pivot].
(while (50 ≤ 50) ↓
    i++
    (1).

~~while (a[i] ≥ a[pivot])~~
~~(while (30 ≤ 50)~~
~~10~~

30 ≤ 50
10 ≤ 50
90 ≤ 50 condn fails goes next while

while a[i] >= a[pivot].

10 >= 50. ✓

40 >= 50 ✗ condtn fails

if (3 < 6)

swap a[i] & a[j].

90 & 40

50	30	10	40	80	20	90	70
0	1	2	3	↑4	5↑	↑j	

while (40 ≤ 50)

i++.

80 ≤ 50 ✗ condn

while (90 >= 50)

~~i++~~ j--

20 >= 50 condn.

if (4 < 5).

swap 80 & 20.

50	30	10	40	20	80	90	70
0	1	2	3	i 4	↑ j 5	6	7

while (20 ≤ 50).

i++

80 ≤ 50 ✗

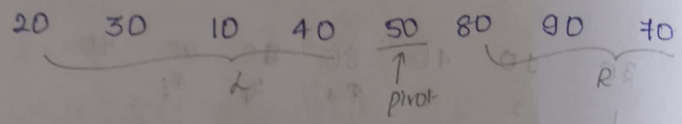
while (80 >= 50)

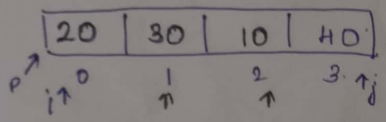
j--

20 >= 50 ✗

if (5 < 4)
x

else
swap a[i] & a[pivot]
20 & 50





while (20 ≤ 20)
i++
30 ≤ 20 x

while (40 ≥ 20)
j--
10 ≥ 20 x

if (1 < 2)
swap 30 & 20

20 10 30 40
i j

while (20 ≤ 20)
i++
30 ≤ 20 x

while (30 ≥ 20)
j--

10 > 20 x.

if (2 < 1) x.

10	20	30	40

80	90	70
0	1	2

while (80 ≤ 80).

i++.

90 ≤ 80 x i = 1

while (70 > 80) x. j = 2.

if (1 < 2).

swap a[1] & a[2].

80	70	90
0	1 ↑i	2. ↑j

while (70 ≤ 80) ✓

i++.

90 ≤ 80 x i = 2.

while (90 > 80) ✓

j--

(70 > 80) x

j = 1.

if (2 < 1) x

else swap a[j] & a[pivot] => 70 & 80.

| 70 | 80 | 90 |

10 20 30 40 50 70 80 90

→ QUESTION

based on ASCII value

Quick sort efficiency

best & average case => T(n).

$$T(n) = \begin{cases} T(1) = 0 & \text{if } n = 1 \\ T(n/2) + T(n/2) + n & \text{otherwise} \end{cases}$$

↑ time taken to sort left part of pivot

↓ time taken to sort right of pivot

↓ time taken to partition given array

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

$aT(n/b) + n^k$.

$a = 2, \quad b^k = (2)^1$

$a = b^k$

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

$= \theta(n^1 \log n)$.

$\therefore T(n) \in \theta(n \log n)$.

$T(n) \in \Omega(n \log n)$.

Worst case.

$T(n) = T(0) + T(n-1) + n$

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

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

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

n by $n-1$.

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

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

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

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

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

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

$= T(n-n) +$

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

n by $n-1$

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

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

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

$\vdots$

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

$T(0) + n-n+1 + n-n+2 + n$

$0 + 1 + 2 + \cdots + n$

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

$\in O(n^2).$

Binary search.

10	20	30	40	50
0	1	2	3	4
↑ low				↑ high = n-1.

while (low ≤ high)
{ low = 0, high = n-1.
mid ← (low + high)/2.

if (key == a[mid])
return mid+1
else key < a[mid].
high = mid - 1.

if key > a[mid]
low = mid+1

}
key 20
while (0 ≤ 4).
{
mid ← 2.

if (20 == 30) ✗

else (20 < 30) ✓
high = 2-1. = 1.

10	20	30	40	50
0	1	2	3	4
↑ low	↑ high			

while (0 ≤ 1)
if (20 · mid = 1.
if (20 == 20) ✓

if (20 < 10) ×

else (20 > 10)

low = mid + 1
= 1

10	20	30	40	50
0	1	2	3	4
	↑ low	↖ high		

while (1 ≤ 1)

mid = 1.

if (20 == 20) ✓

return 1

best case 1

worst case & Average $T(n/2) + 1$

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + 1 & \text{else} \end{cases}$$

→ comparing key element wrt mid elem.

time take.

→ if search on key left or right so 1

$T(n) = T(n/2) + 1$.

replace n by $n/2$

$T(n) = T(n/4) + 2$.

$= T(n/8) + 3$.

$= T(n/2^3) + 3$.

$2^i = n$.

$= T(n/2^i) + i$

$= T(n/n) + i$

$= T(1) + i$

$= 1 + i$

$= 2^i = n$

$\log_2$

$i =$

$T(n) = 1$

$T(n) \in \Theta$

$= 1 + i$

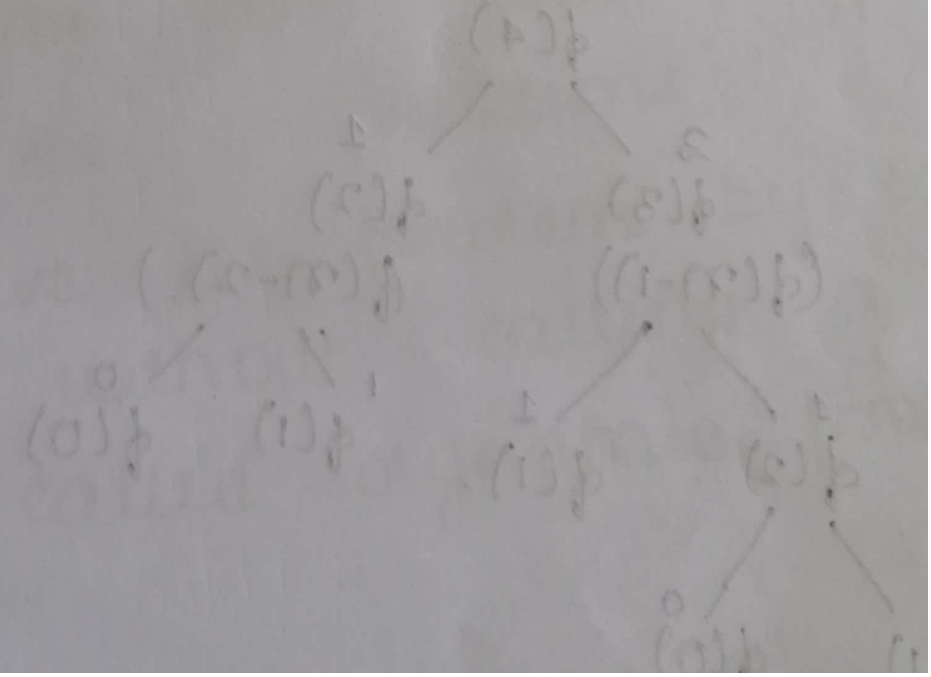
$= 2^i = n.$

$\log 2 = \log n$

$i = \log_2 n.$

$T(n) = 1 + \log_2 n.$

$T(n) \in \Theta \log_2 n$ //.

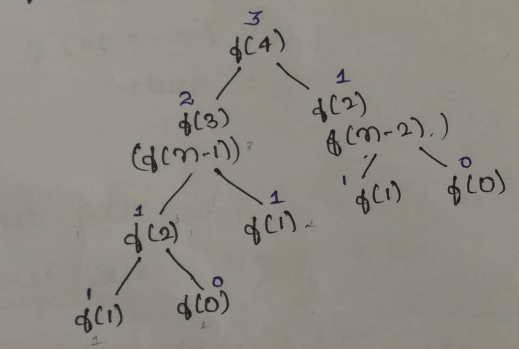


MODULE 04.

Dynamic programming & greedy technique

Fibonacci

n	0	1	2	3	4
f(n)	0	1	1	2	3



Left Right ROOT

```
if (n==0)
    ret 0

if (n==1)
    ret 1

else
  f(n) = f(n-1) + f(n-2).
```

f(0)	f(1)	f(2)
0	1	1

using recursion. $O(2^n)$.

using dynamic $O(n)$.

Dynamic programming

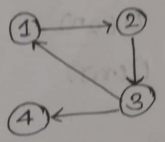
Solving the given problem by overlapping the subproblems.

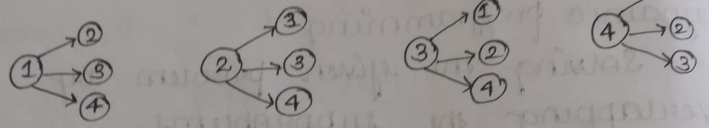
You can solve the problem by taking the values without calling a function repiatedly called dynamic programming

examples

computing a binomial coefficient,

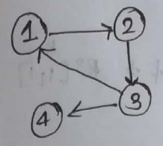
Warshall's algorithm - Transitive closure.
↕
all pairs shortage path.





Floyd's algorithm is with weighted graph.
warshalls — u — not — u —

ALGORITHM warrs A[[1...n, 1..n])
//
// i/p
// o/p.



$R^0 = A =$

	1	2	3	4
1	0	1	0	0
2	0	0	1	0
3	1	0	0	1
4	0	0	0	0

no. of rows → i , columns → j , compare → K.

$R^0 \leftarrow A$

for K ← 1 to 4.

for i ← 1 to 4.

for j ← 1 to 4 do

$R^1[1,1] \leftarrow R^0[1,1]$ || ~~or~~ ||

~~return~~ $R^{(1)}$. $R^{(K-1)}[1,1]$ && $R^0[1,1]$.

= 0 || 0 && 0

= 0

$R^1[1,2] = R^0[1,2]$ || $R^0[1,1]$ && $R^0[1,2]$.

= 1 || 0 && 1.

= 1.

$R^1[1,3]$ = $R^0[1,3]$ || $R^0[1,1]$ && $R^0[1,3]$.

0 || 0 && 0

= 0

$R^1[1,4]$ = $R^0[1,4]$ || $R^0[1,1]$ && $R^0[1,4]$;

= 0 || 0 && 0

= 0

i ↑ 1.

i = 2

$R^1[2,1] = R^0[2,1] \,||\, R^0[2,1] \,\&\&\, R^0[1,1]$

$= 0 \,||\, 0 \,\&\&\, 0$

$= 0$

$R^1[2,2] = R^0[2,2] \,||\, R^0[2,1] \,\&\&\, R^0[1,2]$

$= 0 \,||\, 0 \,\&\&\, 1$

$= 0$

$R^1[2,3] = R^0\,1.$

$R^1[2,4] = R^0[2,4] \,||\, R^0[2,1] \,\&\&\, R^0[1,1]$

$= 0 \,||\, 0 \,\&\&\, 0$

$= 0$

i = 3

$R^1[3,1]$

$R^1[3,2] = R^0[3,2] \,||\, R^0[3,1] \,\&\&\, R[1,2]$

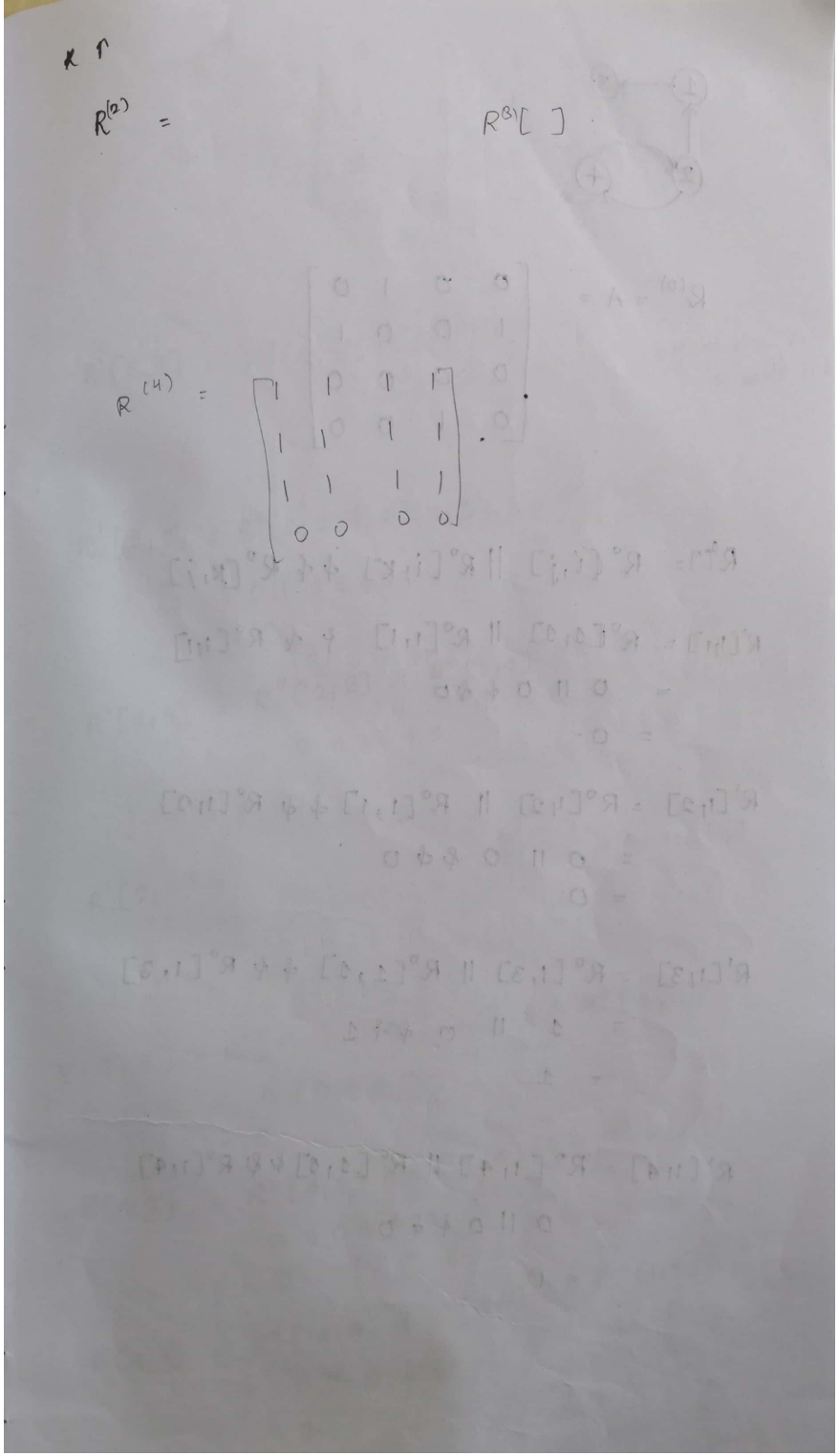
$= 0 \,||\, 1 \,\&\&\, 1$

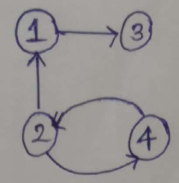
$= 1$

$R^1[3,3]$

$R^1[3,4]$

$$R^{(1)} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$





$$R^{(0)} = A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$R^1[\,] = R^0[i,j] \;||\; R^0[i,k] \;\&\&\; R^0[k,j]$$

$$R^1[1,1] = R^0[1,1] \;||\; R^0[1,1] \;\&\&\; R^0[1,1]$$
$$= 0 \;||\; 0 \;\&\&\; 0$$
$$= 0$$

$$R^1[1,2] = R^0[1,2] \;||\; R^0[1,1] \;\&\&\; R^0[1,2]$$
$$= 0 \;||\; 0 \;\&\&\; 0$$
$$= 0$$

$$R^1[1,3] = R^0[1,3] \;||\; R^0[1,1] \;\&\&\; R^0[1,3]$$
$$= 1 \;||\; 0 \;\&\&\; 1$$
$$= 1$$

$$R^1[1,4] = R^0[1,4] \;||\; R^0[1,1] \;\&\&\; R^0[1,4]$$
$$= 0 \;||\; 0 \;\&\&\; 0$$
$$= 0$$

$R^{(1)} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$.

$R'[2,1] = R^0[2,1] \,||\, R^0[$ where there is 1 with it

$R'[2,2] = R^0[2,2] \,||\, R^0[2,1] \,\&\&\, R^0[1,2]$

$= 0 \,||\, 1 \,\&\&\, 0$

$= 0$

$R'[2,3] = R^0[2,3] \,||\, R^0[2,1] \,\&\&\, R^0[1,3]$

$= 0 \,||\, 1 \,\&\&\, 1$

$= 1$.

$R'[2,4] =$

$R'[4,1] = R^0[4,1] \,||\, R^0[4,1] \,\&\&\, R^0[1,1]$

$= 0 \,||\, 0 \,\&\&\, 0$

$= 0$

$R'[4,3] = R^0[4,3] \,||\, R^0[4,1] \,\&\&\, R^0[1,3]$

$= 0 \,||\, 0 \,\&\&\, 1$

$= 0$

$R'[4,4] = R^0[4,4] \,||\, R^0[4,1] \,\&\&\, R^0[1,4]$

$= 0 \,||\, 0 \,\&\&\, 0$

$= 0$

$$R^1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

~~$R^2[1,1]$~~ $R^2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$

$R^2[1,1] = R^1[1,1] \,||\, R^1[1,2] \,\&\&\, R^1[2,1]$
$= 0 \,||\, 0 \,\&\&\, 0$
$= 0$

$R^2[1,2] = R^1[1,2] \,||\, R^1[1,1] \,\&\&\, R^1[2,2]$
$= 0 \,||\, 0 \,\&\&\, 0$
$= 0$

$R^2[1,4] = R^1[1,4] \,||\, R^1[1,2] \,\&\&\, R^1[2,4]$
$= 0 \,||\, 0 \,\&\&\, 1$
$= 0$

$R^2[2,1] = R^1[2,1] \,||\, R^1[2,2] \,\&\&\, R^1[2,1]$
$= 1$

$R^2[2,2] = R^2[2,2] \,||\, R^2[2,2] \,\&\&\, R^2[2,2]$
$= 0 \,||\, 0 \,\&\&\, 0$
$= 0$

$$T(n) = \sum_{k=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} 1$$

$$= \sum_{k=1}^{n} \sum_{i=1}^{n} n - 1 + 1$$

$$= n \sum_{k=1}^{n} n - 1 + 1$$

$$= n \sum_{k=1}^{n} n.$$

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

$$= n^3 \quad //$$

Binomial coefficient using Dynamic programming

$n C_r$. $5C_3$

$$\frac{n!}{(n-r)!\,r!} = \frac{5!}{(5-3)!\,2!}$$

5!

$\frac{5 \times 4!}{2! \times 3!} = \frac{5 \times 4 \times 3!}{5 \times 2!}$

i ↑ \ r → / n−1 ↓	0	1	2	3
0	1	X	X	X
1	1	1	X	X
2	1	2	1	X
3	1	3	3	1
4	1	4	6	4
5	1	5	10	10

$C[i,j] = C[i-1, j-1] + C[i-1, j]$.

$C[0,0] = C[0-1, 0-1] + C[0-1, 0]$.

$i = j$

$C[0,0] = 1$.

$C[0,1] = C[0-1, 1-1] + C[0-1, 1]$

$C[0,2]$

$C[0,3]$

$C[1,0] = C[0, 0-1] + C[0, 0]$

$= 0 + 1 = 1$.

$C[2,1] = C[1,0] + C[1,1]$.

$= 1+1 = 2$.

$C[3,1] = C[2,0] + C[2,1]$.

$= 1+2 = 3$.

$C[3,2] = C[2,1] + C[2,2]$

$= 2+1 = 3$.

Algorithm: Read n & r.

for $i \leftarrow 0$ to n..

for $j \leftarrow 0$ to min(i, r).

$C(i,j) = C(i-1, j-1) + C(i-1, j)$.

$1,0)$

$^7C_5$.

$\frac{7\times6\times5\times4\times3\times2\times1}{5\times4\times3\times2\times1}$

	0	1	2	3	4	5
0	1	x	x	x	x	x
1	1	1	x	x	x	x
2	1	2	1	x	x	x
3	1	3	3	1	x	x
4	1	4	6	4	1	x
5	1	5	10	10	5	1
6	1	6	15	20	15	6
7.	1	7	21.	35	35	21

$C[2,1] = C[1,0] + C[1,0]$.
$= 1 + 1 = 2$.

$C[3,1] = C[2,0] + C[2,1]$
$= 1 + 2 = 3$.

$C[4,1] = C[3,0] + C[3,1]$
$= 1 + 3 = 4$.

$C[5,1] = C[4,0] + C[4,1]$.
$= 1 + 4 = 5$.

Time efficiency

Floyd's algo

A =

1

2

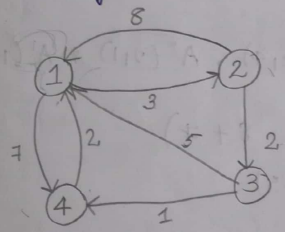
3

4

A' =

Time efficiency - $\theta(n^3)$

Floyd's algorithm



$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 2 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

In A^1 fix first row & first column values from A^0

$$A^1 = \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & \infty & 0 \end{bmatrix}$$

$A^1(2,3) = \min(A^0(2,3), A^0(2,1) + A^0(1,3))$ (intermediate node, destination node)

$= \min(2, 8 + \infty)$

$A^1(2,3) = 2.$

$A^1(2,4) = \min(A^0(2,4), A^0(2,1) + A^1(1,4))$

$= \min(\infty, 8 + 7).$

$= \min(\infty, 15)$

$A^1(2,4) = 15$

$A^1(3,2) = \min(A^0(3,2), A^0(3,1) + A^0(1,2))$

$= \min(\infty, 5 + 3)$

$A^1(3,2) = 8.$

$A^1(3,4) = \min(A^0(3,4), A^0(3,1) + A^0(1,4)).$

$= \min(1, 5 + 7)$

$= 1$

$A^1(4,2) = \min(A^0(4,2), A^0(4,1) + A^0(1,2))$

$= \min(\infty, 2 + 3)$

$= 5$

$A^1(4,3) = \min(A^0(4,3), A^0(1,4) + A^0(1,3))$

$= \min(\infty, 7 + \infty).$

$A^1(4,3) = \infty$

$A^2 =$

	1	2	3	4
1	0	3		
2	8	0		
3	5	8		
4	2			

$A^2(1,3) =$

$A^2(1,4)$

$A^2(3$

A^2

$$A^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix} \end{array}$$

$A^2(1,3) = [\min(A^1(1,3), A^1(1,2) + A^1(2,3)]$

$= \min(\infty, 3+2)$.

$= 5$.

$A^2(1,4) = \min(A^1(1,4), A^1(1,2) + A^1(2,4))$

$= \min(7, 3+15)$

$= 7$.

$A^2(3,1) = \min[A^1(3,1), A^1(3,2) + A^1(2,1)]$

$= \min(5, 8+8)$

$= 5$

$A^2(3,4) = \min(A^1(3,4), A^1(3,2) + A^1(4,2))$

$= \min(1, 8+5)$

$= 1$

$A^2(4,1) = \min(A^1(4,1), A^1(4,2) + A^1(2,1)]$

$= \min(2, 5+8)$

$= 2$

$A^2(4,3) = \min(A^1(4,3), A^1(4,2) + A^1(2,3))$

$= \min(\infty, 5+2)$

$= 7$.

$$A_3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & 6 \\ 7 & 0 & 2 & 3 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix} \end{array}$$

$A^3(1,2) = \min(A^2(1,2), A^2(1,3) + A^2(3,2))$

$= \min(3, 5+8)$

$= 3.$

$A^3(1,4) = \min(A^2(1,4), A^2(1,3) + A^2(3,4))$

$= \min(7, 5+1)$

$= 6.$

$A^3(2,1) = \min(A^2(2,1), A^2(2,3) + A^2(3,1))$

$= \min(8, 2+5)$

$= 7.$

$A^3(2,4) = \min(A^2(2,4), A^2(2,3) + A^2(3,4))$

$= \min(15, 2+1)$

$= 3.$

$A^3(4,1) = \min[A^2(4,1), A^2(4,3) + A^2(1,3)]$

$= \min(2, 7+5)$

$= 2.$

$A^3(4,2) = \min[A^2(4,2), A^2(4,3) + A^2(2,3)]$

$= \min(5, 7+2)$

$$A_4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & 6 \\ 5 & 0 & 2 & 3 \\ 3 & 6 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix} \end{array}$$

$A^4(1,2) = \min [A^3(1,2), A^3(1,4) + A^3(4,2)]$

$= \min (3, 6+5)$

$= 3.$

$A^4(1,3) = \min (A^3(1,3), A^3(1,4) + A^3(4,3))$

$= \min (5, 6+7)$

$= 5$

$A^4(2,1) = \min (A^3(2,1), A^3(2,4) + A^3(4,1))$

$= \min (7, 3+2)$

$= 5$

$A^4(2,3) = \min (A^3(2,3), A^3(2,4) + A^3(4,3))$

$= \min (2, 3+7)$

$= 2.$

$A^4(3,1) = \min (A^3(3,1), A^3(3,4) + A^3(4,1))$

$= \min (5, 1+2)$

$= 3.$

$A^4(3,2) = \min (A^3(3,2), A^3(3,4) + A^3(4,2))$

$= \min (8, 1+5)$

$= 6$

$$A^k(i,j) = \min\left(A^{k-1}(i,j),\ A^{k-1}(i,k) + A^{k-1}(k,j)\right)$$

for $k \leftarrow 1$ to n do

for $i \leftarrow 1$ to n do.

for $j \leftarrow 1$ to n do

$A^k(i,j) = \min\left(A^{k-1}(i,j),\right.$

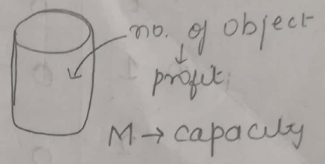
$\left.A^{k-1}(i,k) + A^{k-1}(k,j)\right)$

time effeciency.

$$T(n) = \sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n} 1$$

$$= n^3.$$

0/1 Knapsack problem using Dynamic programming.

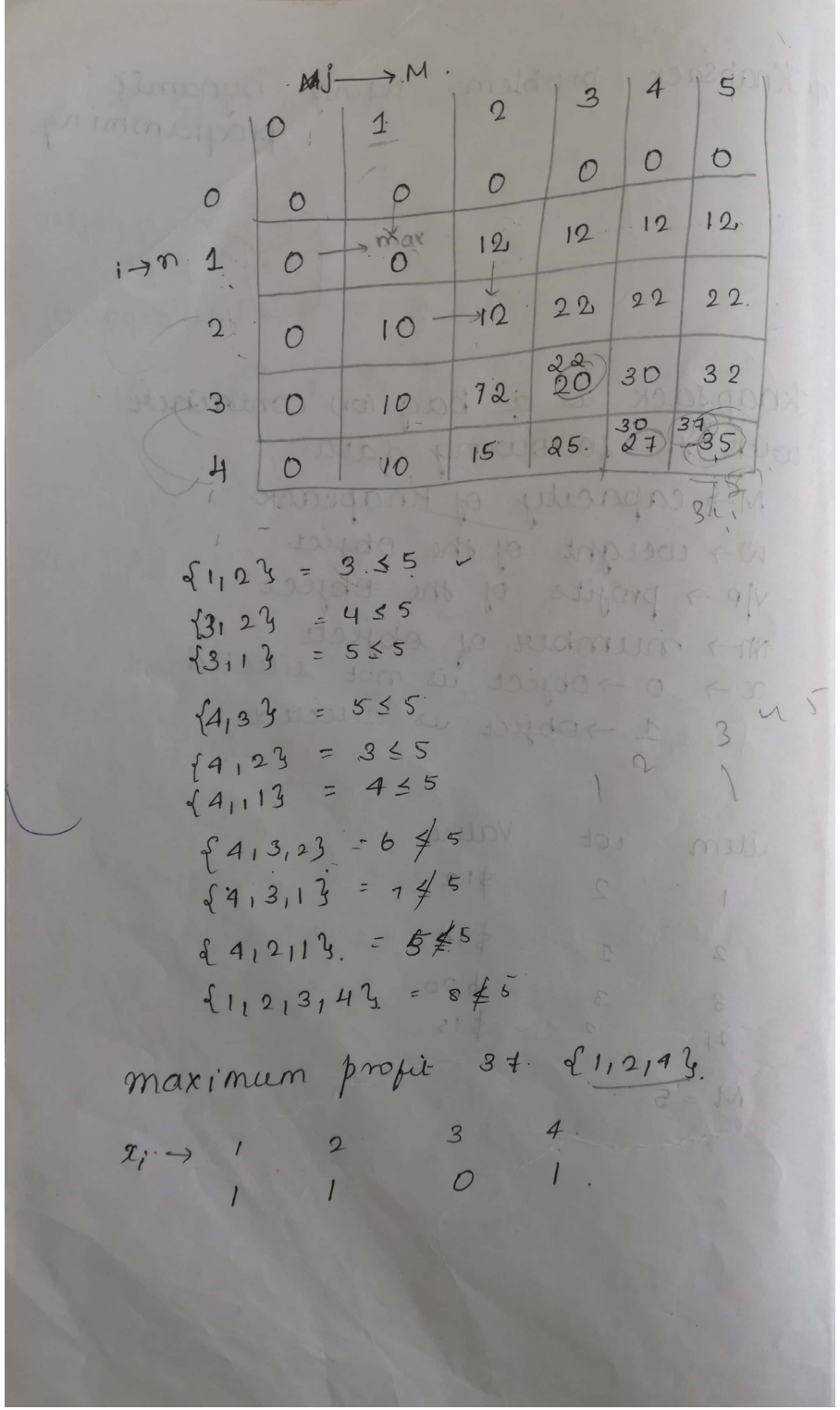


Knapsack is a bag (or) container with the following data

M → capacity of knapsack.
W → weight of the object
V/P → profits of the object
n → number of objects
x → 0 → object is not selected
1 → object is selected

item	wt	value
1	2	$12
2	1	$10
3	3	$20
4	2	$15

M = 5



$\{1,2\} = 3 \leq 5$ ✓

$\{3,2\} = 4 \leq 5$

$\{3,1\} = 5 \leq 5$

$\{4,3\} = 5 \leq 5$

$\{4,2\} = 3 \leq 5$

$\{4,1\} = 4 \leq 5$

$\{4,3,2\} = 6 \not\leq 5$

$\{4,3,1\} = 7 \not\leq 5$

$\{4,2,1\} = 5 \not\leq 5$

$\{1,2,3,4\} = 8 \not\leq 5$

maximum profit 37. $\{1,2,4\}$.

$x_i \rightarrow$	1	2	3	4
	1	1	0	1

→ formula

$V[i][j] = 0$ if $i \geq j = 0$

$V[i-1][j] =$ if $w[i] > j$

$= \max(V[i-1][j], V[i-1][j - w_i] + P[i])$

else if $(w[i] < j)$

$i = 4 \quad j = 1.$

$V[4][1] =$

$w[4] = 2 > 1.$

$= V[3][1] = 10$

$i = 4$

$V[4][2]$

$w[4] = 2 > 2 \quad x$

$\max(V[3][2], V[3][2-2] + P[4])$

$\max[12, , 0 + 15]$

$= 15.$ //

$V[4][3].$

$= 2 > 3 \quad x.$

$\max(V[3][3], V[3][1] + 15)$

$\max(22, 10 + 15)$

$\max(25)$ //

$V[4][4] = \max(V[3][4], (V[3][2] + 15))$

$= \max(30, 27)$

$= 30$

$V[4,5] = \max(V[3][5], V[3][3] + 15)$

$\max(32, 37)$

37

no. of objects	W	P
1	2	1
2	3	2
3	4	5
4	5	6

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	8

maximum profit = 8 {2,4}

x =	1	2	3	4
	0	1	0	1

V[4][1]

w[4] > 1.

5 > 1 ✗

= ~~max (V[3][1], V[3][~~

V[3][1] = 0

V[4][2].

5 > 1.

= V[3][2]

= 1

V[4][3].

5 > 3.

= V[3][3]

= 2.

= V[4][4]

= V[3][4]

= 5

V[4][5]

5 > 5 ✗

= max [V[3][5], V[3][0] + P(4).]

= max (5 , 0 + 6).

= 6.

V[4][6] = max [V[3][6], V[3][1] + 6]

= max (6 , 6)

V[4][7]

= max (V[3][7], V[3,2] + 6).

= max (7, 7) = 7

V[4][8]

= max (V[3][8], V[3,3] + 6)

= max (7 , 2 + 6)

= 8

Algorithm.

```
//input ⇒ n - no. of objects
        m - capacity of knapsack
        w - weight of object
        p → profit of object

//Output ⇒ max profit i.e V[i,j].
    for i ← 0 to n do.
        for j ← 0 to m do
            if (i==0 || j==0).
                V(i,j) = 0.

        else if (wi > j)
                V(i,j) = V(i-1,j)

        else
        V(i,j) = max (V(i-1,j), V(i-1,j-w)+
                                pi)
```

Time effeciency.

$$T(n) = \sum_{i=0}^{n} \sum_{j=0}^{m} 1$$

$$= \sum_{i=0}^{n} m - 0 + 1$$

$$= m+1 \sum_{i=0}^{n} 1$$

$$= (m+1)(n+1)$$

$$= mn + m + n + 1$$

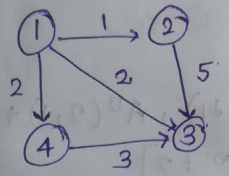
$$= mn$$

$$T(n) \in \Theta(mn)$$

n	w	p
1	5	2
2	1	1
3	3	3
4	4	6
5	2	8

M = 10

Floyd's



$A^0 =$

	1	2	3	4
1	0	1	2	2
2	∞	0	5	∞
3	∞	∞	0	∞
4	∞	∞	3	0

fix first row & first column

$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 2 & 2 \\ \infty & 0 & 5 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & 3 & 0 \end{bmatrix} \end{array}$$

$A^1(2,2) = \min[A^0(2,2), A^0(2,1) + A^0(1,2)]$
$= \min[0, \infty + 1]$
$= 0$

$A^1(2,3) = \min[A^0(2,3), A^0(2,1) + A^0(1,3)]$
$= \min[5, \infty + 2]$
$= 5$

$A^1(2,4) = \min[A^0(2,4), A^0(2,1) + A^0(1,4)]$
$= \min[\infty, \infty + 2]$
$= \infty$

$A^1(3,2) = \min[A^0(3,2), A^0(3,1) + A^0(1,2)]$
$= \min(\infty, \infty + 1)$
$= \infty$

$A^1(3,4) = \min(A^0(3,4), A^0(3,1) + A^0(1,4))$
$= \min(\infty, \infty + 2)$
$= \infty$

$A^1(4,2) = \min(A^0(4,2), A^0(4,1) + A^0(1,2))$
$= \min(\infty, \infty + 1)$
$= \infty$

$A^1(4,3) = \min(A^0(4,3), A^0(4,1) + A^0(1,3))$
$= \min(3, \infty + 2) = 3$

$A^2 =$

	1	2	3	4
1	0	1	2	2
2	∞	0	5	∞
3	∞	∞	0	∞
4	∞	∞	3	0

$A^2[1,3] = \min(A^1(1,3), A^1(1,2) + A^1(2,3))$

$= \min(2, 1+5)$

$= 2$

$A^2[1,4] = \min(A^1(1,4), A^1(1,2) + A^1(2,4))$

$= \min(2, 1+\infty)$

$= 2.$

$A^2[3,1] = \min(A^1(3,1), A^1(3,2) + A^1(2,1))$

$= \min(\infty, \infty + \infty)$

$= \infty$

$A^2[4,1] = \min(A^1(4,1), A^1(4,2) + A^1(2,4))$

$= \min(\infty, \infty + \infty)$

$= \infty.$

$A^2[4,3] = \min(A^1(4,3), A^1(4,2) + A^1(2,3))$

$= \min(3, \infty + 5)$

$= 3.$

$A^3 =$

	1	2	3	4
1	0	1	2	2
2	∞	0	5	∞
3	∞	∞	0	∞
4	∞	∞	3	0

$A^3[1,2] = \min[A^2(1,2), A^2(1,3) + A^2(3,2)]$

$= \min(1, 2+\infty)$

$= 1$

$A^3[1,4] = \min[A^2(1,4), A^2(1,3) + A^2(3,4)]$

$= \min(2, 2+\infty)$

$= 2$

$A^3[2,1] = \min[A^2(2,1), A^2(2,3) + A^2(3,1)]$

$= \min(\infty, 5+\infty)$

$= \infty$

$A^3[2,4] = \min[A^2(2,4), A^2(2,3) + A^2(3,4)]$

$= \min(\infty, 5+\infty)$

$= \infty$

$A^3[4,1] = \min[A^2(4,1), A^2(4,3) + A^2(3,1)]$

$= \min(\infty, 3+\infty)$

$= \infty$

$A^3[4,2] = \min[A^2(4,2), A^2(4,3) + A^2(3,2)]$

$= \min(\infty, 3+\infty) = \infty$

$A^4 =$

	1	2	3	4
1	0	1	2	2
2		0		∞
3	∞	∞	0	∞
4	∞	∞	3	0

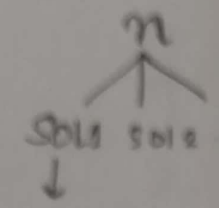
$A^4[1,2] = \min(A^3[1,2), A^3[1,4] + A^3[4,2])$
$= \min(1, 2+\infty)$
$= 1.$

$A^4[1,3] = \min(A^3[1,3], A^3[1,4] + A^3[4,3])$
$= \min(2, 2+3)$
$= 2.$

$A^4[2,1] = \min(\infty,$

$$\begin{bmatrix} 0 & 1 & 2 & 2 \\ \infty & 0 & 5 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & 3 & 0 \end{bmatrix}$$

Greedy method



feasible sol^n → optimal

A greedy method is a problem solving technique tries to find the best solution that works in stages, considering one input at a time to get an optimal solution.

The problems solving using greedy method is called optimisation problem

feasible solution

A solution that satisfies the given constraint is called feasible solution.

Optimal solution

It is a feasible solution that maximise the profit or minimise the loss.

MST : Minimum spanning tree

1) prims
2) Kushkals.

MST is a graph with all nodes are connected without forming any cycle or closed circuit.

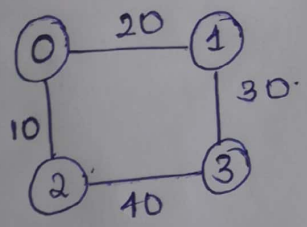
Let $G = (N, E)$

$n \rightarrow$ no. of vertices / nodes.

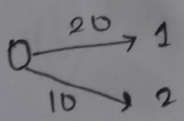
$E \rightarrow (n-1)$ edges are selected to find out the spanning tree.

Examples for MST - prims, Kushkals.

1)

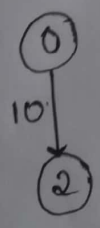


prims → first 0 shd be considered

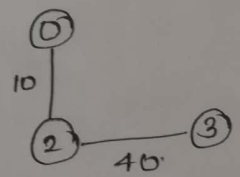


minimum is 10 so 0 to 2.

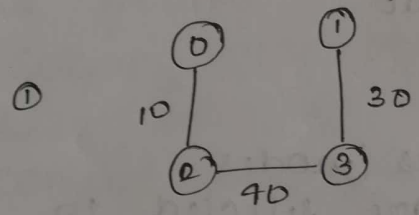
⇒



from 2 to 3.

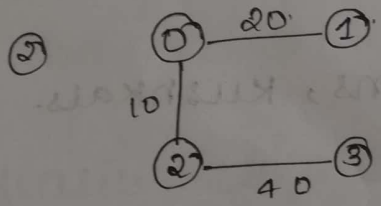


from 3 to 1

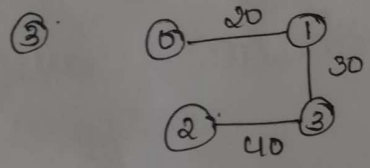


10+40+30 = 80

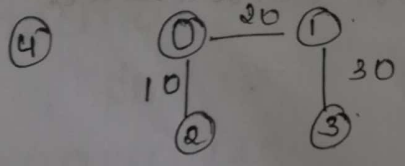
from 1 to 2



= 70



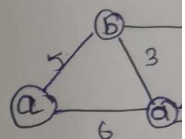
= 90



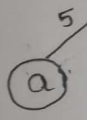
= 60 minimum

find MST

2)



1)



m=5

2)

a

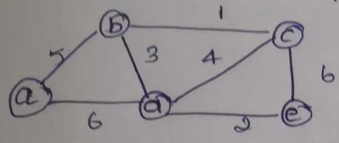
3)

a

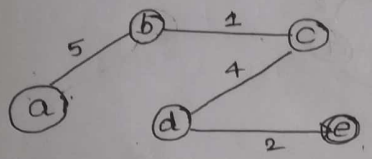
a

find MST using prims for following graph.

2)



1)

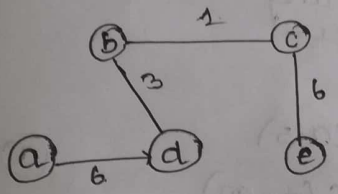


→ step by step as 1st question.

= 12.

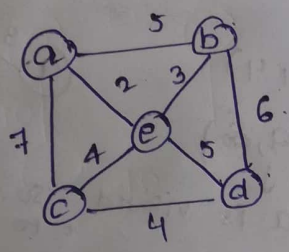
n = 5 , 5-1 = 4 edges.

2)



= 16.

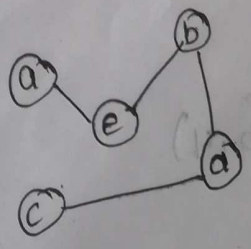
3)

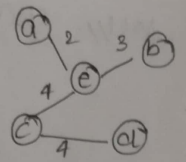


a —2— e ⇒ a —2— e ⇒ a —2→ e —3→ b

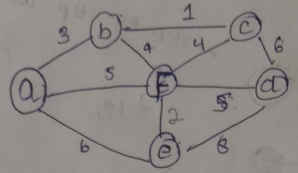
a —2— e —3— b —6— d —4— c

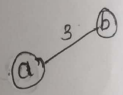
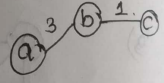
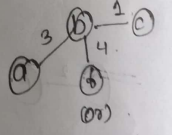
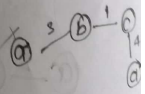
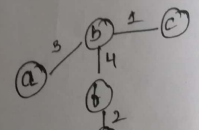
= 15



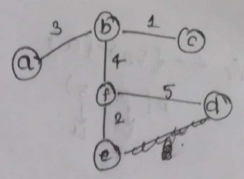


= 13.



Tree Vertices	Remaining Vertices.	Illustration
a(-,-)	b(a,3) f(a,5) e(a,6) c(-,∞) a(-,∞).	
b(a,3)	c(b,1), f(b,4). a(-,∞), e(a,~~6~~). ↓ a is visited so via a e to e.	
c(b,1).	f(c,4), d(c,6). ~~e(c,∞)~~ f(b,4). e(a,6).	 
f(b,4).	e(f,2), d(f,~~5~~).	

e(f,2). d(e,8), d(f,5)



=15

Find out the feasible soln.

no. of edge $C_{\text{no. } v-1}$ - no. of loops.

$= {}^{10}C_5 - 5$

$= 252 - 5$

$= 247$

$\frac{10!}{5!\,5!} \quad \frac{2\times1}{2\times1}$

$\frac{10\times9\times8\times7\times6}{5\times4\times3\times2\times1}$

Algorithm prim(G).

I/P : a connected graph G=(V,E)

O/P : ET, the all of edges comprising of a MST $V_T \leftarrow \{v_0\}$

ET → set of edges.

$ET \leftarrow \phi$

for $i \leftarrow 1$ to $|V|-1$ do.

find minimum weight edge $e^* = (v^*, u^*)$ among edges (v,u) such that v is in V_T & u $V-V_T$

$V_T \leftarrow V_T \cup \{u^*\}$

$E_T \leftarrow E_T \cup \{e^*\}$

end for

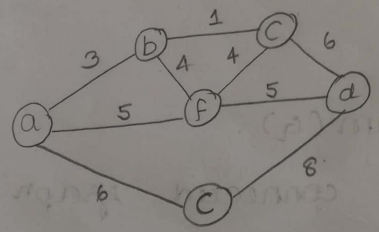
return ET.

$V_T \leftarrow \{v_0\}$ $\{a\}$

b(a, 3)

Efficiency = $O(n^2)$ for weight matrix representation of graph.

$O(m, \log n)$ for adjacency list representation



Kruskals

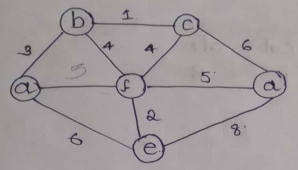
Tree edge

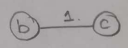
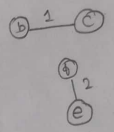
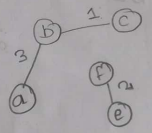
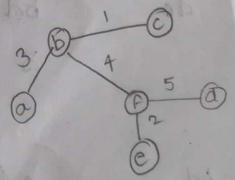
bc
1

e
2.

a

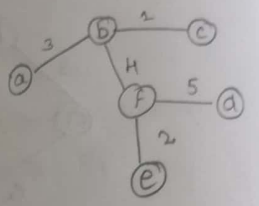
Kruskkals



Tree edges	Remaining Vertices	Illustration
-	bc ef ab bf cf 1 2 3 4 4 af df ae cd de 5 5 6 6 8.	
bc 1	ef ab bf cf af 2 3 4 4 5 df ae cd de 5 6 6 8.	
ef 2.	ab bf cf af 3 4 4 5 df ae cd de 5 6 6 8.	
ab. 3.	bf cf af df 4 4 5 5 ae cd de 6 6 8.	

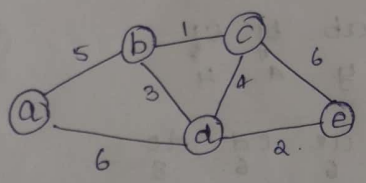
cf cannot be taken its forming a loop.

af cannot be taken.
(5)



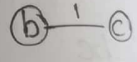
af (5) ae 6 cd 6 de 8

3+1+4+5+2 = 15

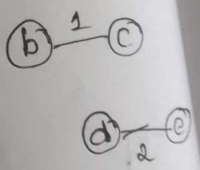


$\Rightarrow {}^{7}C_{4} - 3 = 31$

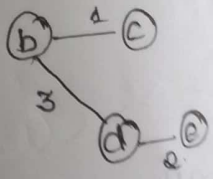
bc de bd cd ab ad ce.
1 2 3 4 5 6 6.



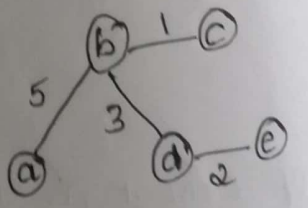
bc de bd cd ab ad ce
1 2 3 4 5 6 6



de bd cd ab ad ce
3 4 5 6 6.



bd cd ab ad ce.
4 5 6 6



ab ad cd
6 6

5+3+1+2 = 11 //.

Algorithm Kruskal
construction

// input : weighted
// output : ET, &

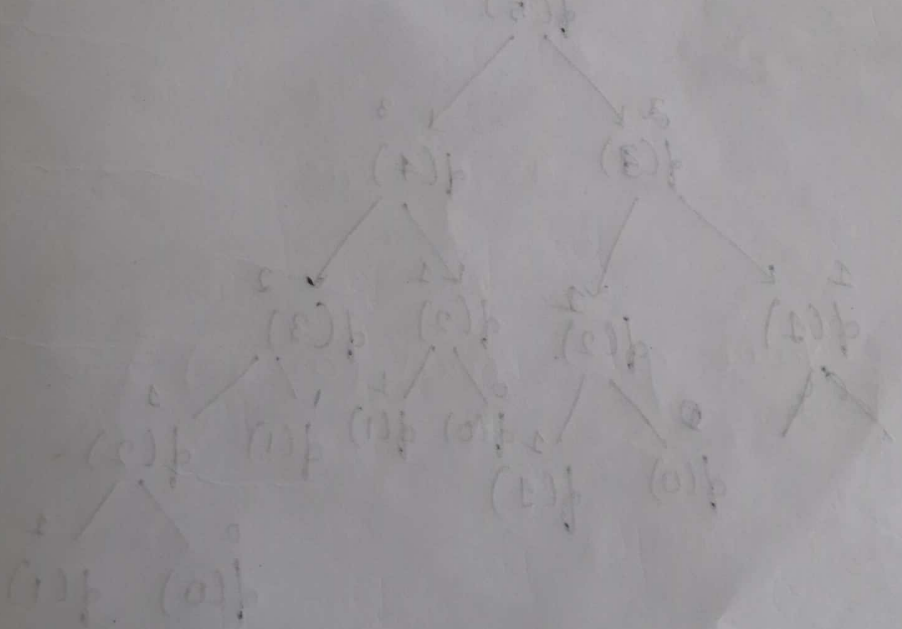
Effecincy mlog

Algorithm Kruskal's algorithm for constructing a minimum spanning tree.

//input : weighted connected graph $G = V, E$

//output : ET, set of edges composing

Effecincy $m \log m$



Dynamic programming

Fibonacci numbers.

F[0], F[1], F[2], F[3], F[4], F[5]

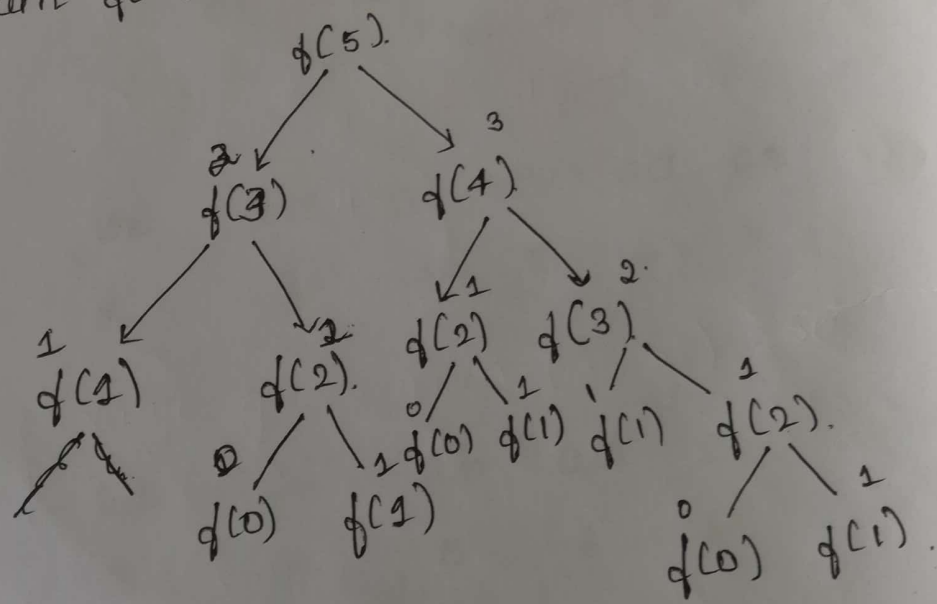
0 1 1 2 3 5

```
int fib
{
  if (n <= 1)
    return n;
    f[0] = 1, F[1] = 1.

  if (n > 1).
    for i ← 2 to n)

    f(n) = f(n-2) + f(n-1)

}
return f(n).
```



using dynamic

f(0)	f(1)
0	1

Dijsktra's al

dist[s]

for a

wh

time

using dynamic programming

$f(0)$	$f(1)$	$f(2)$	$f(3)$	$f(4)$	$f(5)$
0	1	1	2	3	5

Dijkstra's algorithm s → source node

```
dist[s] ← 0
for all v ∈ V - {s} do
    dist[v] ← ∞
    S ← ϕ
    Q ← V
while (Q ≠ ϕ) do
    u ← min dist(Q, dist)
    S ← S ∪ {u}
    for all v ∈ neigbours[u] do
        if (dist[v] > dist[u] +
                cost[u][v])
        then
            dist[v] = dist[u] + cost[u][v]
return dist
```

time effeciency

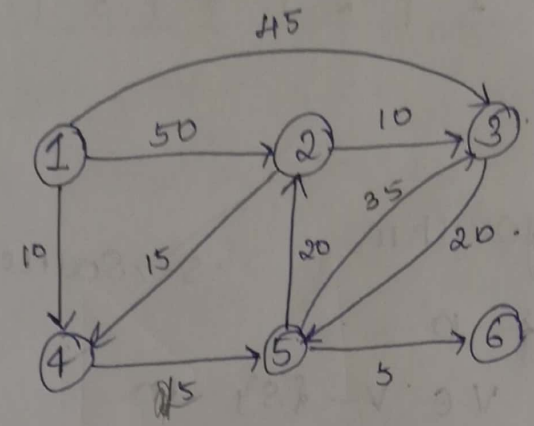
$n \times n = n^2$. or $|V|^2$.

$V * V \rightarrow$ no. of applied relaxation

$\rightarrow$ no. of vertices

$T(n) \in \Theta(|V|^2)$

Djikstra's algm & problem.



Source Vertex (1)	2	3	4	5	6.
4	50	45	10	∞	∞ →
5	50	45	10	25	∞ →
2	45	45	10	25	∞ →
3	45	45	10	25	∞ →
6	45	45	10	25	∞ →

4 = { value of 2 = min (∞, 50) = 50.
value of 3 = min (∞, 45) = 45.

4 → 5 via 1 = 10 + 15 = 25

5 = 2 = 25 + 20 = 45

as

V	d[v]	path
1	0	- -
2	45	1-4-5-2
3	45	1-3
4	10	1-4
5	25	1-4-5
6	∞	- -

(1,2) = 50

(1,3) = 45

(1,4) = 10

(1,5) = (1,6) = ∞

(4,2) = min (∞, 50) = 50 (∞ → no value, 50 → above value.)

(4,3) = min (∞, 45) = 45.

(4,4) = 10

(4,5) = min (∞, 10+15)
= min (∞, 25)
= 25

(4,6) = ∞.

6.

∞ → 1

∞ → 4

∞ → 5

∞ → 2

∞ → 3.

(5,2) = 25 + 20 = 45.

(5,3) = min (35+25, 45)
= 45

(5,4) = 10

(5,5) = 25

(5,6) = min (∞, ∞) = ∞.

(2,2) = 45

(2,3) = min (50+10, 45) = 45

(2,4) = min (15, 10) = 10

(2,5) = min (∞, 25) = 25

(2,6) = ∞

(3,2) = min (∞, 45) = 45

(3,3) = 45

(3,4) = min (∞, 10) = 10

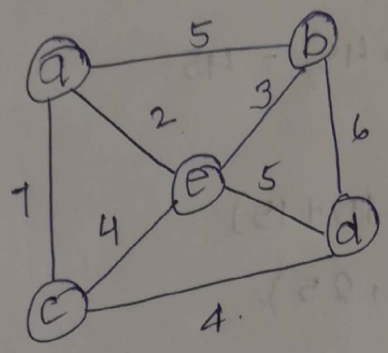
(3,5) = min (20, 45) = 20

(3,6) = ∞

Formula

if (d[u] + cost[u][v] < d[v])

d[v] = d[u] + cost[u,v].



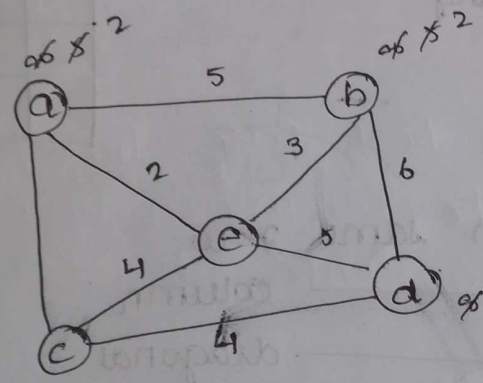
Source c.

c	a	b	d	e
d	7	∞	4	4
e	7	10	4	4
a	6	7	4	4
b	6	7	4	4
	6	7	4	4

d|b = min (6+4, ∞) = 10

e|a = min (2+4, 7) = 6.

V	d[v]	path
a	6	a > e, c / e
b	7	c - e - b.
d	4	c —4— d
e	4	c —4— e
c	–	–



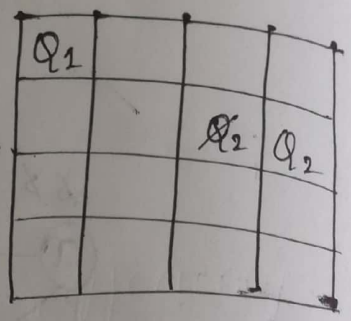
Module 5.

Backtracking & Lower bound Arguments

** 8m

→ Queen's problem

4 x 4



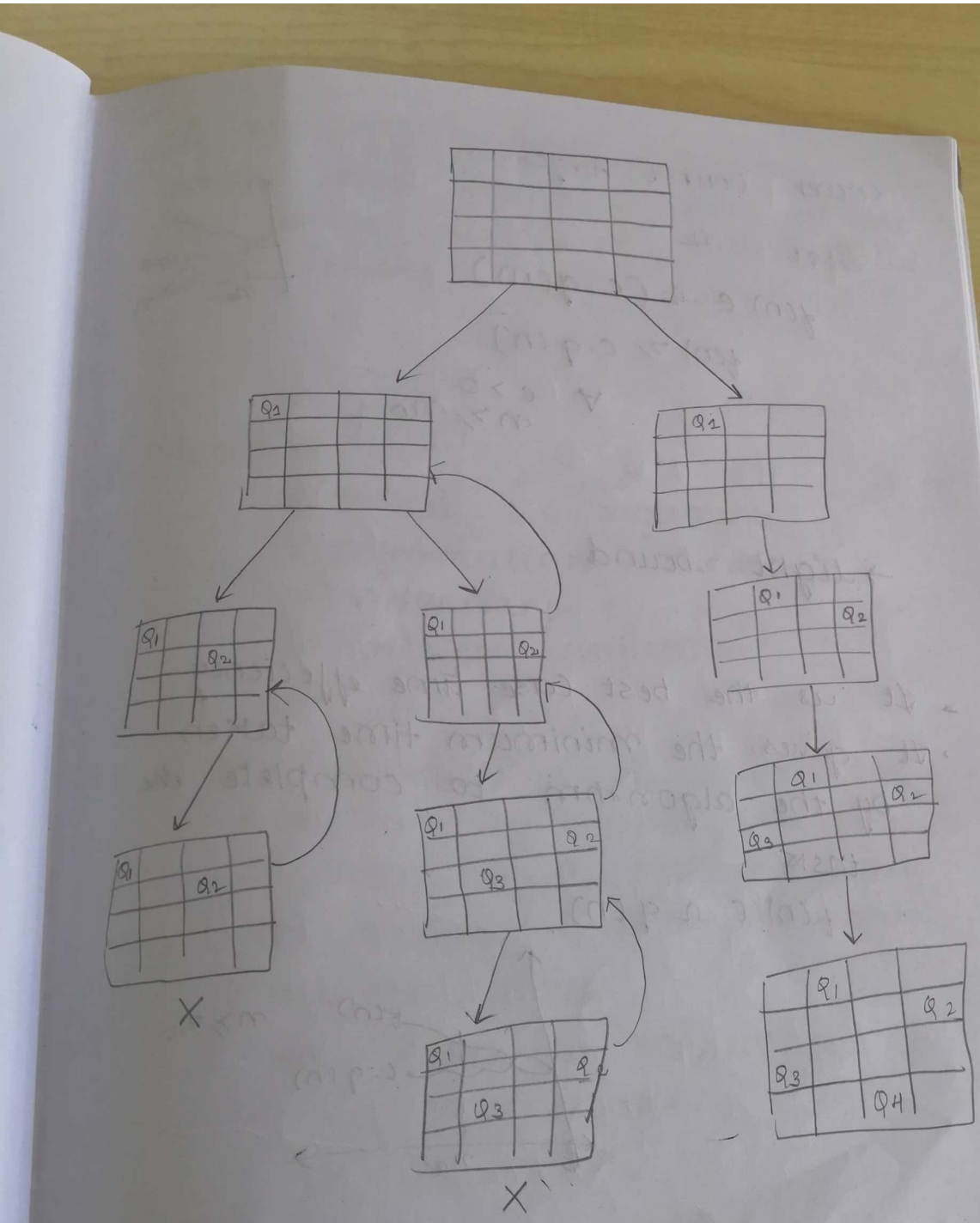
without backtracking

Soln = $^{16}C_4$ =

Condition

1) no queens on same row
2) ——— " ——— column
3) ——— " ——— diagonal

State space tree



soln

Q1	Q2	Q3	Q4
2	4	1	3
3	1	4	2

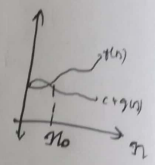
Lower Bound Argts.

Best - Ω.

$f(n) \in \Omega(c.g(n))$.

$f(n) \geq c.g(n)$

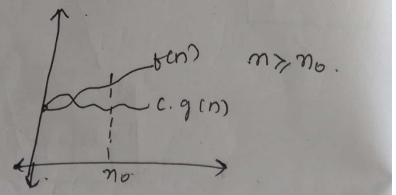
$\forall \; c > 0$
$n \geq n_0$.



→ tight bound.

- It is the best case time effeciency.
- It gives the minimum time taken by the algorithm to complete the task.

$f(n) \in \Omega g(n)$.



Advantages of lower bound.

algorithm will consume atleast the specified time

Improvement can be analysed by taking better algorithm with new lower bound.

if we get a new algorithm, whose lower bound is same as that of current algorithm, this is called tight bound.

* Methods
  1) Trivial (lower bound) L.B. arguments
  2) Information - theortic
  3) Adversary
  4) problem reduction

1) The trivial lower bound can be obtained by counting number of items in the problems input tht must be process & write all the outputs.
   eg - finding maximum of n elements
   $\Omega(n)$

2) Information - theoretic.
   here the lower bound is based on amount of information it has to be produced based. on all sorting & searching.
   eg - searching linear - $\Omega(1)$, binary - $\Omega(\log_2 n)$
   bubblesort → $\Omega(n^2)$, selection → $\Omega(n^2)$

the lower bound using this method by can be obtained by using decision trees (comparing trees).

3) Adversary L.B Argts

the L.B is obtained by measuring the amount of work required ~~of~~ to reduce a set of potential inputs to single input

eg - number guessing game

the L.b for this method is $\Omega(\log_2 n)$

4) problem reduction

A given problem is transform into another problem for which the soln is exist in the form of a known algorithm.

eg - Knapsack

P, NP, NP- complete

Problems

Polynomial time, determinis

$T(n) \in \Omega$

greedy

searchin

sorting

→ 4 differ

P - class

P

this p

polynom

This

eg -

The p

reph

Linea

bine

bub

P, NP, NP- complete, Np-hard class.

** Problems.

- Polynomial time, deterministic
  - $T(n) \in \Omega(n^k)$
  - greedy method.
  - searching n^2,
  - sorting $\Omega(n \log n)$.
- non polynomial time.
  - non deterministic.
  - $T(n) \in \Omega(k^n)$
  - eg. 2^n
  - TOH, Knapsack.

→ 4 different

P - class

P stands for polynomial ~~type~~ time.

this problem can be solved & verified in polynomial time

This problems are easy to solve & verify.

eg - all searching and sorting problems.

The polynomial time efficiency can be represented in terms of n^k.

Linear search time efficiency is $\theta(n)$ & $\Omega(n^0)$.

binear search $\Omega(n^1)$.

bubble sort $\theta(n) = \Omega(n^2)$

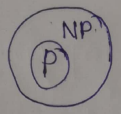
NP - class.

The problem can be solved in non polynomial time (exponential) but verified in polynomial time

time effecieney of np class can be represented by $T(n) \in \Omega(k^n)$

example : Sudoku, Knapsack, finding prime factors, Tower of Hanoi

effeciency of Knapsack is 2^n

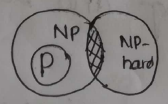


$P \subseteq NP$

all p ~~pr~~ class problems can be NP class problems & vice versa not possible

the problems cums under p class are tractable & np - class are intractable

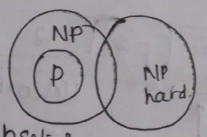
→ NP- Hard class.



A problem is NP hard if every problem in NP class can be polynomially reduced to another set of problem are called NP - hard.

eg - All Optimising problem.
Knapsack, travelling Sales ~~Search~~ problem.
↓.

→ NP - complete
A problem which is present in both NP - class & NP - hard.
eg - finding largest no.



Knapsack →
finding no. of subsets

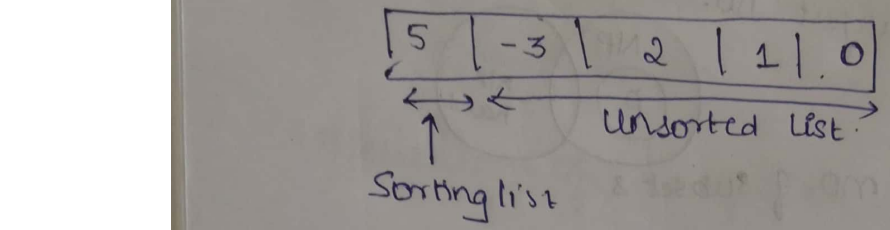
→ Knapsack is a NP - complete since it ~~is~~ NP - hard is used for optimising & NP class is

all NP complete problems are NP hard but all NP hard problems are not NP complete

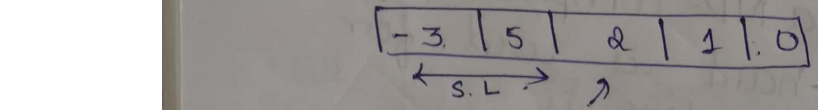
Decision Tree.

Insertion sort.

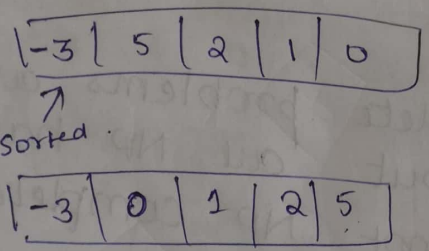
eg



comparing -3 with 5.



Compare 2, 1, 0 with -3.

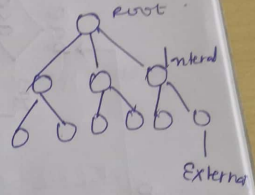


time efficiency $O(n^2)$.

It is also called as comparison tree which represents only the comparison of given elements in the array, while sorting or searching

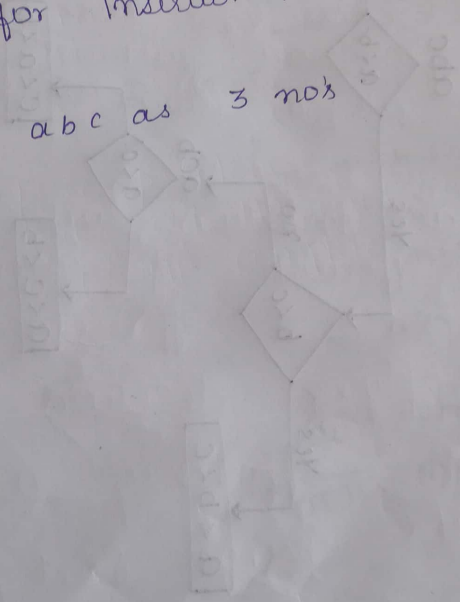
The internal nodes are represents comparision

External nodes represents outcomes.



Decision Tree for Insertion sort for 3 no:s.

Let consider a b c as 3 no's



Selection sort,

Ex
Insertion
elements = -3, 2,

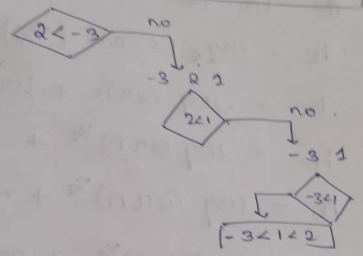
Time effic
decisio

→ n!

T(n)

Selection sort, max element, min elem

Ele
Insertion
elements = -3, 2, 1.



Time efficiency fo Insertion sort using decision tree

→ n!

$T(n) = 2^h$.

$T(n) = n!$ → ①.

$T(n) = 2^h$ → using tree.
↳ ②.

② in ①.

$2^h = n!$

taking log on b.s.

$\log(2^h) = \log(n!)$.

$h \log_2 2 = \log_2(n!)$.

$T(n) = h = \log_2(n!)$.

Using stirling formula.

$n! = \sqrt{2\pi n}\,(n/e)^n$ → ③

① in ②

$$h = \log_2\left(\sqrt{2\pi n}\,(n/e)^n\right)$$

$$h = \log\sqrt{2\pi n} + \log(n/e)^n$$

$$h = \log(2\pi n)^{1/2} + n\log(n/e)$$

$$h = \log(2\pi n)^{1/2} + n[\log n - \log e]$$

$$h = \log(2\pi n)^{1/2} + n\log n - n\log e$$

↓ highest degree

$$h = n\log_2 n$$

$$T(n) = h = \Omega(n\log_2 n)$$

It is tight lower bound because. mergesort, quicksort are having. $n\log_2 n$ lower bound.

as Insertion sort is not effecient & merge & quick Sort is effecient as after applying decision tree for time efficiency as $n\log n$ but merge sort & quick sort the time effeciney is $n\log n$ only before applying

decision tree

decision tree.