Algorithms

Sir Ashish

Algorithm is a step by step procedure to solve computational problem similar to program but the following are the ~~class~~ differences b/w them →

Algorithm	Program
(1) written at design time	(1) written at implementation time
(2) Person should have domain knowledge	(2) Programmer
(3) Any Language	(3) Programming language
(4) Machine Independent (H/w & OS)	(4) H/w & OS dependency (Linux/windows)
(5) Analyze (Time/Space)	(6) Testing

Priori Analysis	Posteriori Testing
1. Algorithm	1. Program
2. Language Indepent	2. Language dependent
3. Hardware Independent	3. Hardware dependent
4. Time & Space Function	4. Watch time & bytes

Algorithm Characteristic's →

1. Input → 0 or more
2. Output → atleast one o/p
3. Definiteness → every statement should be clear $\sqrt{-1}$ X
4. Finiteness → Algorithm must have finite steps
5. Effectiveness → Every step must be used

How To write a Algorithm →

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

⇒

```
Algorithm  swap (a,b)
Begin
    temp ← a ;
    a ← b ;
    b ← temp ;
end
```

How to Analyze an Algorithm →

1. Time
2. Space
3. N/W Consumption → How much data transfer is done.
4. Power consumption
5. CPU Registers consumption

Algorithm swap(a,b)

```
{   temp = a;   — 1 unit of time
    a = b;      — 1
    b = temp;   — 1   each statement
}
```

f(n) = 3
time fn

Space
a — 1
b — 1
temp — 1
S(n) = 3 words
Space fn

x = 5 * a + 6 * b → 1

O(1)
Constant

O(1)

Frequency Count method →

Algorithm sum(A, n)

```
{   s=0                          — 1
    for (i=0; i<n; i++)          — n+1
                                   n
    { s = s + A[i];              — 1
    }
    return s;
}
```

f(n) = 2n + 3
O(n)

Space

A — n words
n — 1
s — 1
i — 1

(n+3) S(n) = n+3 O(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)
    }
  }
}
```

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

$O(n^2)$

S(n)

A	n^2	$(n*n)$
B	n^2	
C	n^2	
n	1	
i	1	
j	1	

$3(n^2+1) \rightarrow O(n^2)$

```
Algorithm multiply (A, B, n)
{ ① for (i=0; i<n; i++)
  ② { for (j=0; j<n; j++)
    ③ { C[i,j] = 0;
    ④   for (k=0; k<n; k++)
    ⑤ { C[i,j] = C[i,j] + A[i,k]
          * B[k,j]; } }
}
```

S(n)

A	n^2	
B	n^2	$3n^2+4$
C	n^2	
n	1	$O(n^2)$
i	1	Time $\rightarrow O(n^3)$
j	1	space $\rightarrow O(n^2)$
k	1	

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

①	$O(n^3)$	$n+1 = n+1$
②		$(n+1)n = n^2+n$
③		$n * n = n^2$
④		$(n+1) * n + n = n^3+n^2$
⑤		$n * n * n = n^3$

① `for (i=0; i<n; i++)` $n+1$

```
{
    stmt;        n
}
```

$O(n)$

② `for (i=1; i<n; i=i+2)`

`{ stmt; }` $\rightarrow n/2$ $f(\frac{n}{2})$ $O(n)$

③

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

i	j	no. of times
0	0 x	0
1	0 ✓	1
	1 x	
2	0	2
	1	
	2 x	
3		3
⋮		⋮
n		n

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

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

$O(n^2)$

④

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

Assume stop when $p > n$

i	p
1	0+1 = 1
2	1+2 = 3
3	1+2+3
4	1+2+3+4
k	1+2+3+...+k

$\therefore$

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

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

$k^2 > n$

$k > \sqrt{n}$

$O(\sqrt{n})$

⑤

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

Assume $i \geq n$

$2^k \geq n$

$k \geq \log n$

$\lceil \log n \rceil$

$O(\log_2 n)$

$\log_2 10 \approx 3.2$

ceil value 4

i
1
1×2 = 2
2×2 = 2^2
⋮
2^k

n = 10

i	time
1	1<10
2	2<10
4	4<10
8	8<10
16	16≮10

⑥

```
for (i=n; i>=1; i=i/2)
```

	$\frac{1}{n}$
$i < 1$	$\frac{n}{2}$
$\frac{n}{2^k} < 1$	$\frac{n}{2^2}$
$2^k > n$	$\frac{n}{2^k}$
$k > \log_2 n$	
$k = \log_2 n$	

⑦

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

$i^2 > n$

$i^2 = n$ $O(\sqrt{n})$

$i = \sqrt{n}$

⑧

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

⑨

```
p = 0
for (i=1; i<n; i=i*2)              log n
{ p++; }
for (j=1; j<p; j=j*2)              log p     O(log log n)
{ stmt; }                          log p
```

(10)

```
for (i=0 ; i<n ; i++)          n+1
{
    for(j=1 ; j<n ; j*=2)      n logn
    { stmt; }                  n logn
}
```

$2n\log n + n$

$O(n \log n)$

```
for ( i=0 ; i<n ; i++)          O(n)
```

$\frac{n}{200} \rightarrow O(n)$

```
for (i=0 ; i<n ; i+=2)          n/2 → O(n)
```

incrementing by any

```
for (i=n ; i>1 ; i--)           O(n)
for (i=1 ; i<n ; i*=2)          O(log2 n)
for (i=1 ; i<n ; i*=3)          O(log3 n)
for (i=n ; i>1 ; i=i/2)         O(log2 n)
```

Analysis of if and while →

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

```
i=0          — 1
while (i<n)  — n+1
{ stmt;      — n        3n+2
  i++        — n        O(n)
}
```

```
a=1;
while (a<b)
{   stmt;
    a*=2;
}                O(log n)
```

```
while (k<n)
{ stmt;
  k=k+i
  i++;
}
                 O(√n)
```

```
i=n;
while (i>1)
{ stmt;          O(log n)
  i/=2;
}
```

```
while (m!=n)
{ if (m>n)
    m-=n;        min time
  else           O(1)
    n-=m;        max time
}
                 O(n)
```

```
Algorithm Test(n)
{   if (n<5)
      print(n)   — 1
}
else
    for (i=0; i<n; i++)
      print(i)   → n
}
Best → O(1)
Worst → O(n)
```

```
if(n>5)
{ for(i=0; i<n; i++)
    stmt
}                O(n)
```

Types of time functions

$O(1)$ → Constant

$O(\log_2 n)$ → Logarithmic

$O(n)$ → linear

$O(n^2)$ → Quadratic

$O(n^3)$ → Cubic

$O(2^n)$ → Exponential

$O(3^n)$

$O(n^n)$

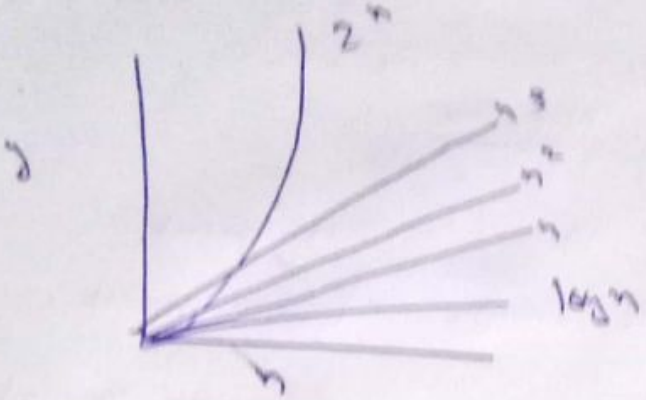
Classes of fns

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

$\log n$	n	n^2	2^n
0	1	1	2
1	2	4	4

$n^* < 2^n$

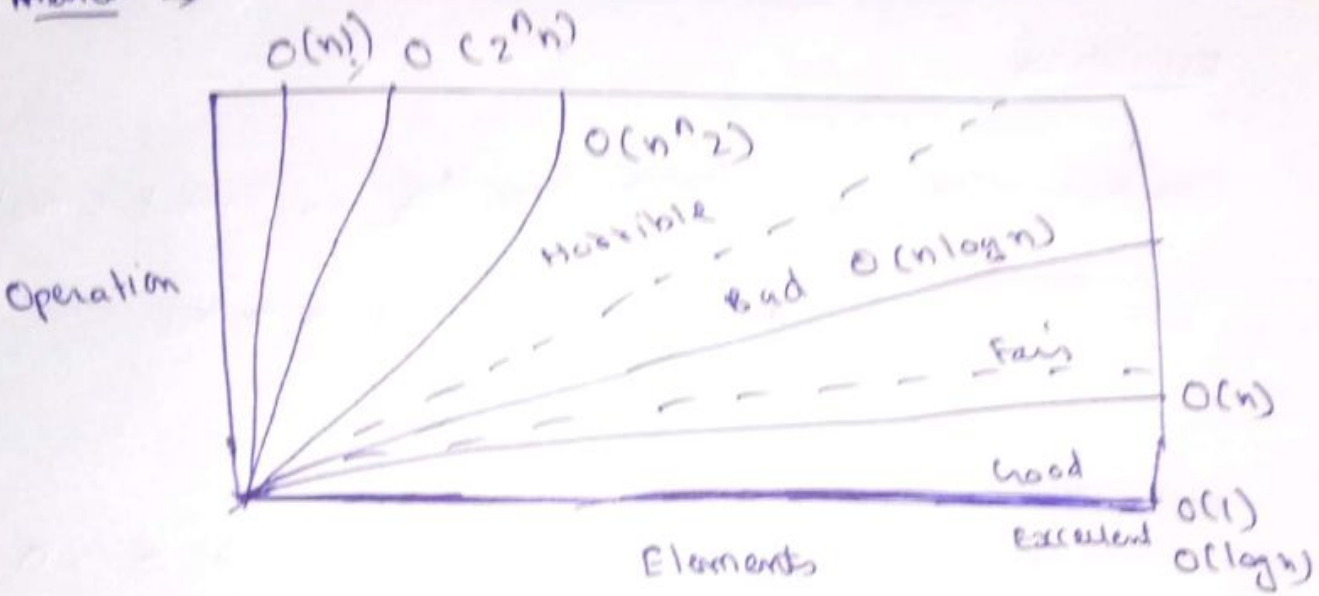
Abdul Rai



Rules

1. Always worst case
2. Remove Constants
3. Different i/ps have diff. variables (a+b)
4. Drop non-dominant terms

Andrew →



O(1) → constant no loops

O(log n) → logarithmic → usually searching algorithms have log n if they are sorted (binary search) (not on hash map)

O(n) → Linear for/while loop

O(n^2) → Nested loops every element needs to be compared with every other element

O(2^n 2) → Exponential → recursive algorithms

O(n!) → loop for every element

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

Asymptotic Notations

O	big-Oh	Upper bound
Ω	big-omega	Lower bound
Θ	theta	Average bound

use full →

Big Oh →

Function $f(n) = O(g(n))$ iff $\exists$ +ve constant C and n_0

such that $f(n) \leq c * g(n) \quad \forall \; n \geq n_0$

e.g. $f(n) : 2n+3$

$2n+3 \leq 10n \quad n \geq 1$

$f(n) \quad \downarrow \quad \downarrow$

$c \quad g(n)$

$f(n) = O(n)$.

$2n+3 \leq 2n+3n$

$\leq 5n$

$n \geq 1$

$f(n) = O(n)$ closest

$= O(n^2)$

$1 < \log n < \sqrt{n} < n < n^2 < \dots < n^n < 2^n < n!$

lower bound — Avg bound — Upper bound

Big - Omega Notation

The fn $f(n) = \Omega(g(n))$ iff $\exists$ +ve constants C and n_0 such that $f(n) \geq C\,g(n)$ $\forall\ n \geq n_0$

$f(n) = 2n + 3$

$2n + 3 \geq 1 \cdot n$ $\forall\ n \geq 1$ degree 1.

$f(n) \geq C\,g(n)$

$\boxed{f(n) = \Omega(n)}$ nearest

$f(n) = \Omega(\log n)$ Not useful.

theta Notation →

The function $f(n) = \theta(g(n))$ iff $\exists$ +ve constants C_1, C_2 and n_0 such that $C_1 g(n) \leq f(n) \leq C_2 g(n)$

e.g:- → $f(n) = 2n + 3$

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

$C_1 n \quad f(n) \quad C_2 n$

Notations for representing bounds only.

$\boxed{f(n) = \theta(n)}$

Exact ✓

Any notation for best case and worst case.

Q. $f(n) = 2n^2 + 3n + 4$

$2n^2 + 3n + 4 \leq 2n^2 + 3n^2 + 4n^2$

$2n^2 + 3n + 4 \leq 9n^2$

$f(n) = O(n^2)$

$2n^2 + 3n + 4 \geq n^2$

$f(n) = \Omega(n^2)$

$n^2 \leq 2n^2 + 3n + 4 \leq 9n^2$

$\Theta(n^2)$.

Q. $f(n) = n^2 \log n + n$

$n^2 \log n \leq n^2 \log n + n \leq 10 n^2 \log n$

$\Theta(n^2 \log n)$ $\Omega(n^2 \log n)$ $O(n^2 \log n)$

$n^2 \leq n^2 \log n \leq n^3$

Q. $f(n) = n!$

$1 \leq 1 \times 2 \times 3 \times \dots \times n \leq n^n$

$1 \leq n! \leq n^n$ $\Theta(n!) = n^n$ $\Omega(n!) = 1$

Q. $f(n) = \log n!$

$1 \leq \log n! \leq \log n^n$

$1 \leq \log n! \leq n \log n$

$\Omega(n)$ $O(n)$

Properties of Asymptotic Notations →

General Properties

- if $f(n)$ is $O(g(n))$ then $af(n) = O(g(n))$

  e.g. → $f(n) = 2n^2 + 5$ is $O(n^2)$

  then $7 \cdot f(n) = 7(2n^2 + 5) = 14n^2 + 35$ $O(n^2)$

- Reflexive,

  if $f(n)$ is given then $f(n) = O(f(n))$

  $f(n) = n^2$ $f(n) = O(n^2)$

- Transitive

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

  ∀ for notations

  $f(n) = O(h(n))$

  $f(n) = n$ $g(n) = n^2$ $h(n) = n^3$

  $f(n) = O(n^3)$

$f(n) = \Theta(g(n))$

$g(n) = \Theta(f(n))$

$f(n) = n^2 \quad g(n) = n^2$

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

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

- Transpose symmetric (Θ and Ω)

$f(n) = \Theta(g(n)) \quad g(n) = \Omega(f(n))$.

$f(n) = n \quad g(n) = n^2$

$n = O(n^2)$

$n^2 = \Omega(n)$

- If $f(n) = \Theta(g(n))$

$f(n) = \Omega(g(n))$

$\Rightarrow f(n) = \Theta(n)$

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

$d(n) = O(e(n))$

$f(n) + d(n) = O(\max(g(n), e(n)))$.

$f(n) = n \quad = O(n)$

$d(n) = n^2 \quad = O(n^2)$

$f(n) + dn = n + n^2 = O(n^2)$

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

$d(n) = O(e(n))$

$f(n) * d(n) = O(g(n) * e(n))$

$a^b = n$

$b = \log_a n$

$b\log a = \log n$

$b = \frac{\lg n}{\log a} = \lg n$

① n^2 & n^3 Comparison of Functions

n	n^2	n^3
2	2^2	2^3
3	3^2	3^3
⋮	⋮	⋮

② Apply log on both sides

$\log n^2 \quad \log n^3$

$2\log n < 3\log n$

Q. $f(n) = n^2 \log(n)$

$g(n) = n(\log n)^{10}$

$\log(n^2 \log n) \quad \log(n(\log n)^{10})$

$= 2\log n + \log\log n \quad \log n + 10\log\log n$

$\Rightarrow \quad \log n + 10\log\log n$

$\Rightarrow \quad f(n) > g(n)$

Q. $f(n) = 3^{n\sqrt{n}} \qquad g(n) = 2^{\sqrt{n}\log_2 n}$

$3^{\sqrt{n}\log n} \qquad 2^{\log n} \qquad 2^{\log_2 n^{\sqrt{n}}}$

$3^{n\sqrt{n}} \quad n^{\sqrt{n}}$

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

Asymptotically equal

Q $f(n) = n^{\log n}$ $g(n) = 2^{\sqrt{n}}$

$\log n \log n$ $\sqrt{n}$

$\log^2 n$ $n^{1/2}$

$2 \log\log n < \frac{1}{2} \log n$ (Apply again)

$f(n) < g(n)$.

Q. $f(n) = 2^{\log n}$ $g(n) = n^{\sqrt{n}}$

$\log n < \sqrt{n} \log n$

Q. $f(n) = 2^n$ $g(n) = 2^{2n}$

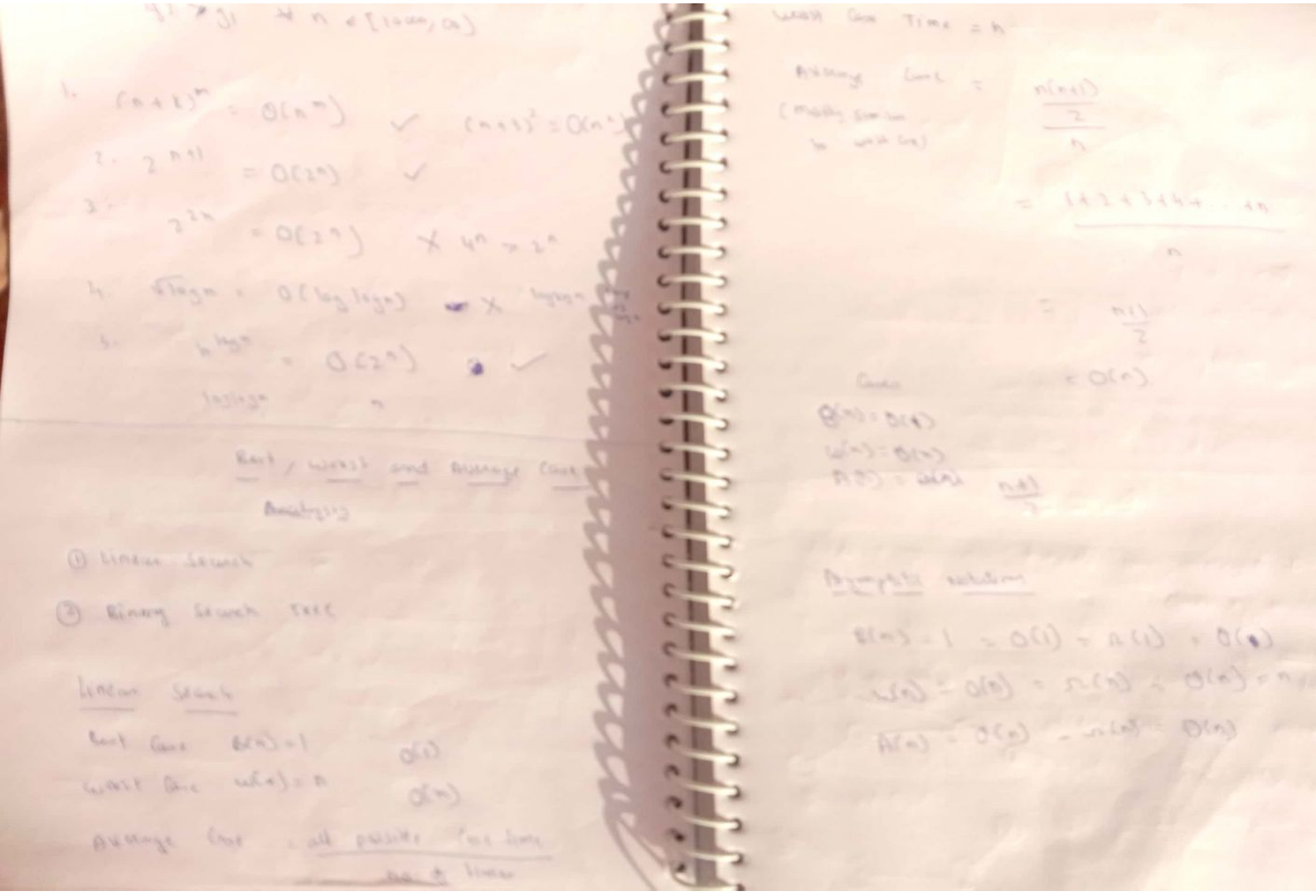
n $2n$

Q. $g_1(n) = \begin{cases} n^3 & n < 100 \\ n^2 & n \geq 100 \end{cases}$

$g_2(n) = \begin{cases} n^2 & n < 10000 \\ n^3 & n \geq 10000 \end{cases}$

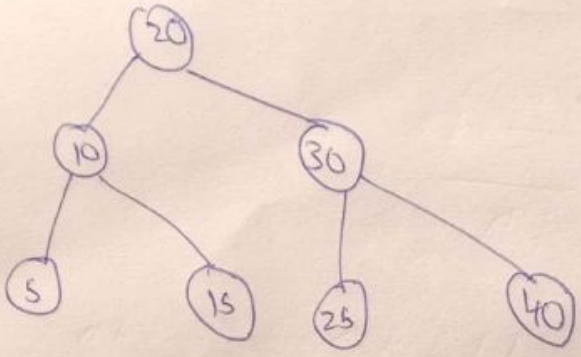
100 $g_1(n)$ 10000

$n \rightarrow$ $g(n)$ $g_2(n)$ $g_2(n)$



Scanned by CamScanner

Binary search tree

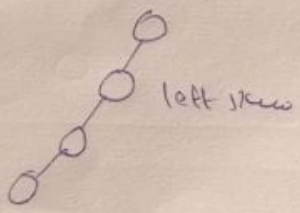


$\log n$ = height

$B(n) = 1$

$\min w(n) = \log n$

$\max w(n) = n$



Disjoint sets

Kruskal Algorithm uses disjoint sets which detects cycle in a graph.

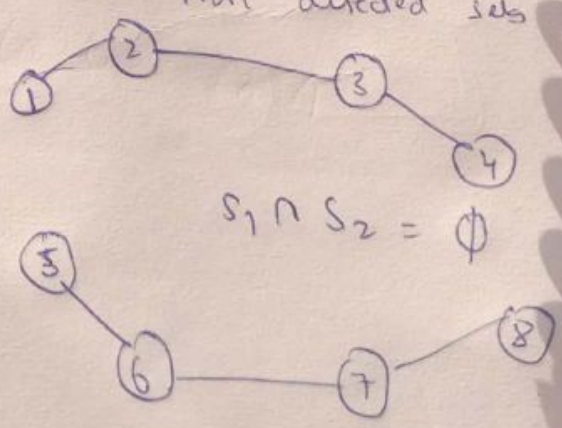
Operations

1. Find — find element in set
2. Union — $S_1 \cup S_2$ find joined edge

$S_1 = \{1, 2, 3, 4\}$

$S_2 = \{5, 6, 7, 8\}$

Non connected
Non directed sets



$S_1 \cap S_2 = \phi$

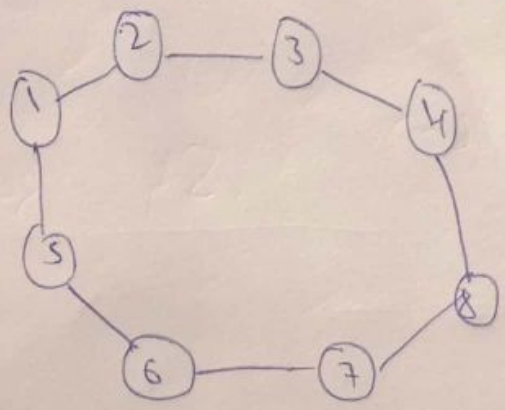
Operations

① find $\leftarrow$ ①, 2, 3, 4}

② union

$S_1 = \{1, 2, 3, 4\}$

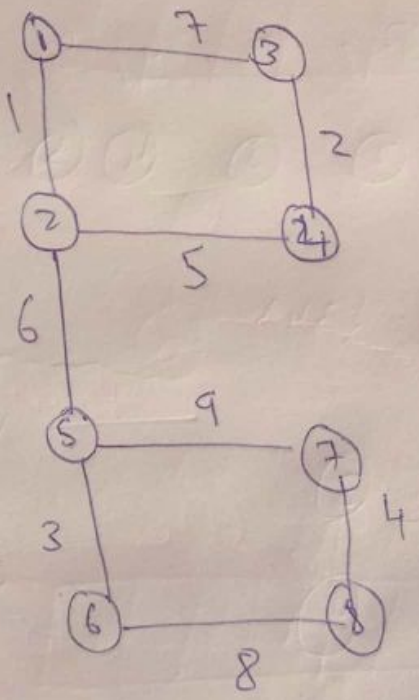
$S_2 = \{5, 6, 7, 8\}$



$S_1 \cup S_2 = \{1, 2, 3, 4, 8, 5, 6, 7\}$ (4,8)

(1,5) cycle in a graph

Q.



(7,8)
(1,2)
(3,4)
(5,6)

$S_1 = \{1, 2\}$

$S_2 \{3, 4\}$

$S_3 = \{5, 6\}$

$S_4 = \{7, 8\}$

$V = \{1, 2, 3, 4, 5, 6, 7, 8\}$

(2,4)
S_1 S_2

(2,4) $S_1 \cup S_2 = \{1, 2, 3, 4\}$

(2,5) $S_1 \cup S_3 = \{1, 2, 5, 6\}$

S_6 {1, 2, 3, 4, 5, 6}

S_4 = {7, 8}

8 →

S_7 = ~~{6, 8}~~ (6, 8)

{1, 8, 3, 4, 5, 6, 7, 8}

5, 7 → Cycle.

Same set

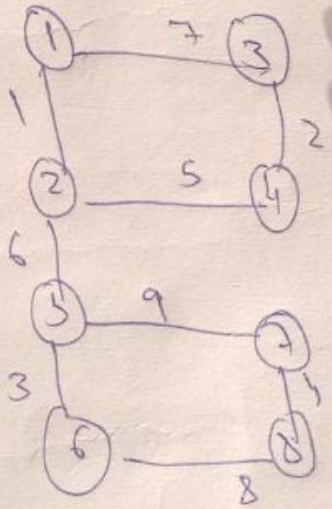
---

u = {1, 2, 3, 4, 5, 6, 7, 8}

① ② ③ ④ ⑤ ⑥ ⑦ ⑧

S_1 = {1, 2}

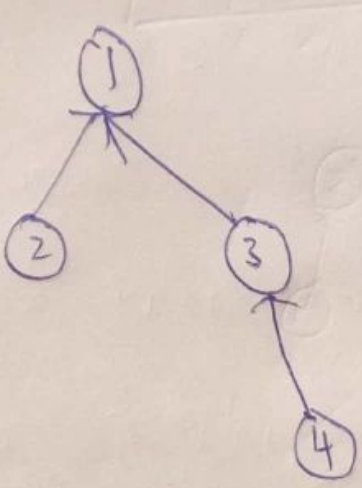
S_2 = {3, 4}



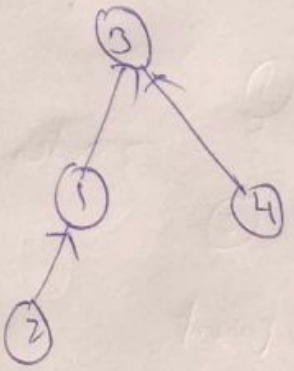
S_3 = {5, 6}

S_4 = {7, 8}

(3,4)

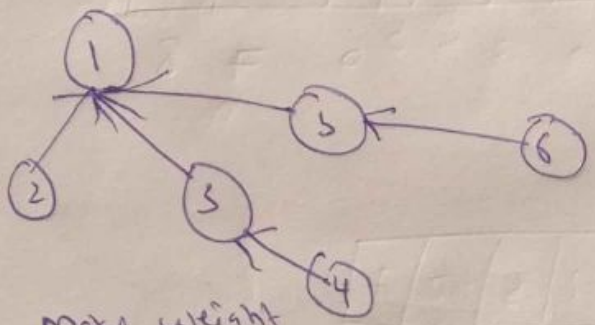


OR



(5,6)

$S_6 = 2, 1, 2, 3, 3, 5, 6$ 3

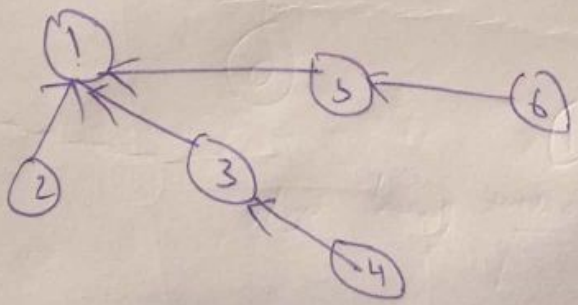


Make weight

(1,3)

Same

Cycle.



Parent

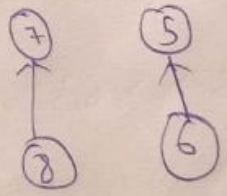
-1	-1	-1	-1	-1	-1	-1	-1
1	2	3	4	5	6	7	8

Parent

-2	+1	-2	3	-1	-1	-1	-1
1	2	3	4	5	6	7	8

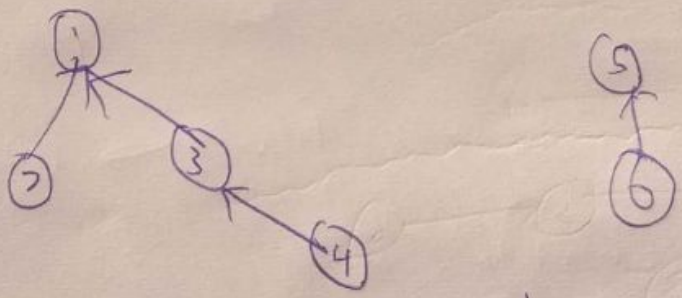
2 nodes

1 parent



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

-4	+1	1	3	-2	5	-2	7

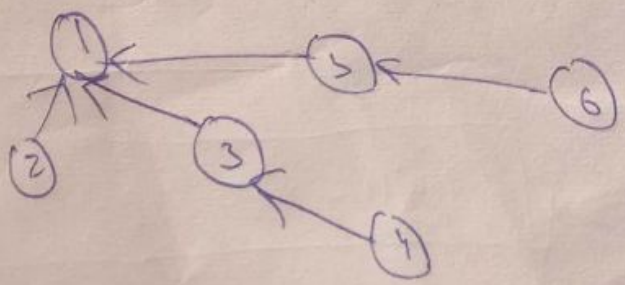


weight
rank

same parent cycle

⇒

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

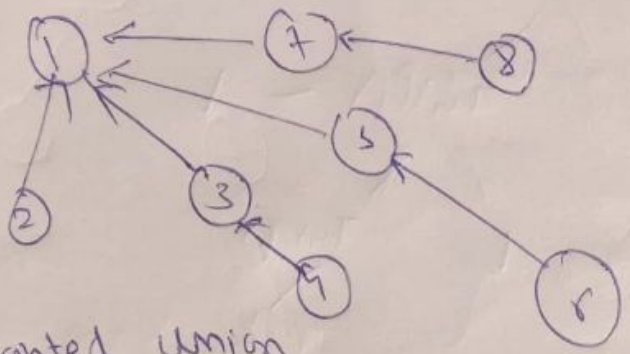


π

(1, 3)

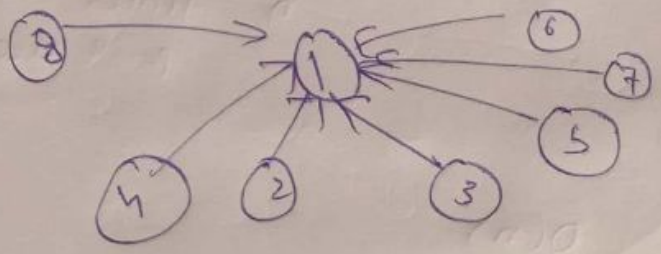
7 ← 8

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



Weighted union

Collapsing union ⇒ To easily find root



4 → 3

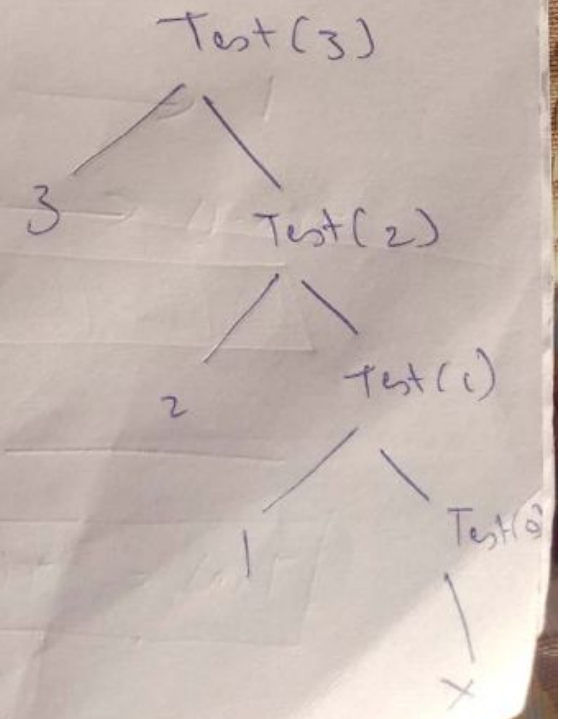
3 → 1

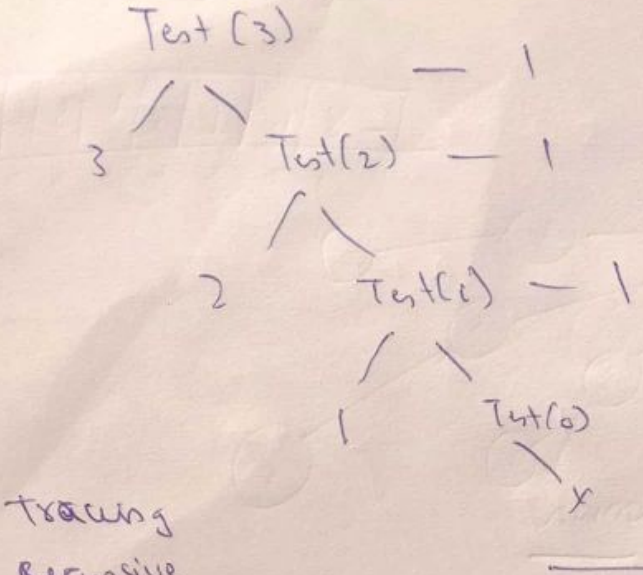
4 → 1

Divide And Conquer →

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

3 3





Tracing Recursive Tree

3+1 Calls

n — (n+1) Calls

n times printf execute

$f(n) = n+1$ $O(n)$

$T(n) \leftarrow$

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

1 ←

T(n-1) ←

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

Constant $C \rightarrow 1$

1 or 2 or C.

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

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

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

~~$T(n) = T(n) + n$~~

$T(n) = T(n-k) \neq$ ~~k~~

(After k steps execution stops)

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

Assume $n - k = 0$

$n = k$

$$\boxed{T(n) = n + 1}$$ $O(n)$.

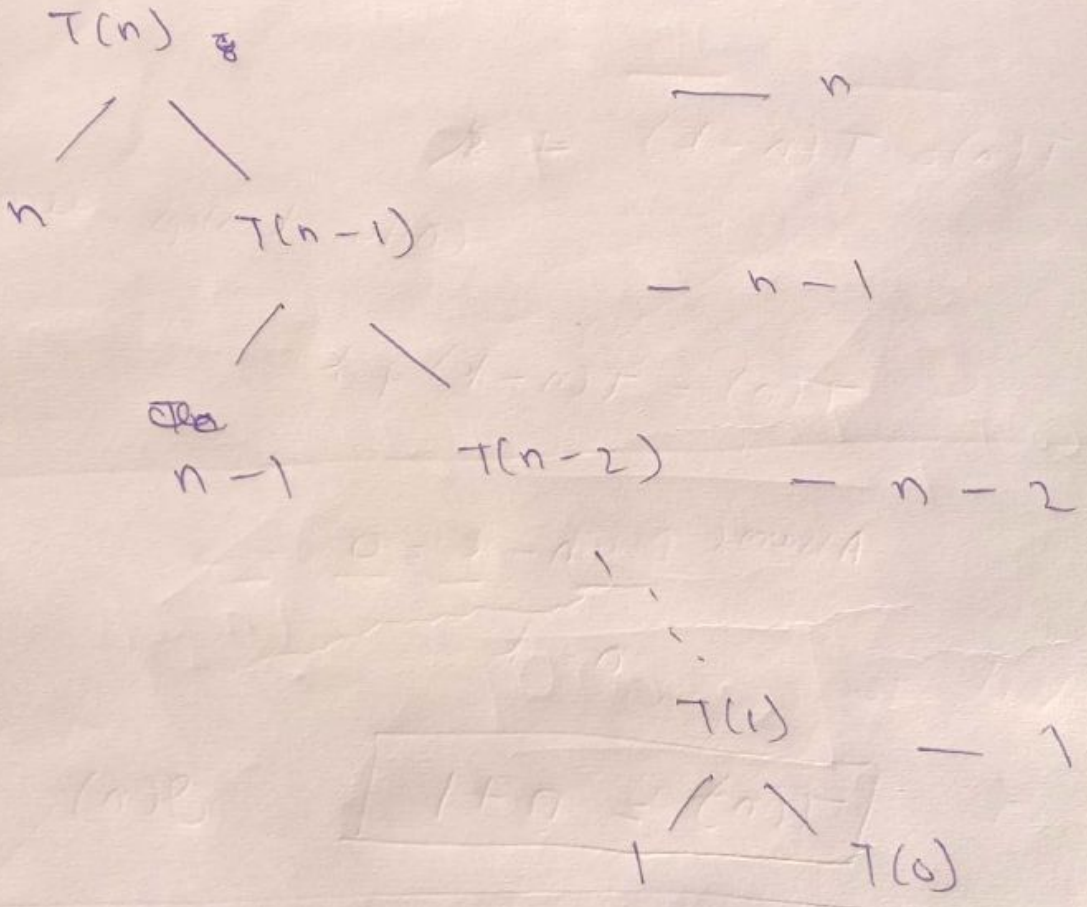
Q.

```
T(n) ——  void Test (int n) {
1    ——    if (n > 0)
n+1  ——    { for ( i=0; i<=n ; i++)
n    ——      { printf ("%d", n); }
T(n-1) ——    Test (n-1);
           }
```

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

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

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



$O(n^2)$

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

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

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

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

$n - k = 0$

$\Rightarrow n = k$

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

$\Rightarrow \boxed{T(n) = \Theta(n^2)}$

$T(n) \rightarrow$

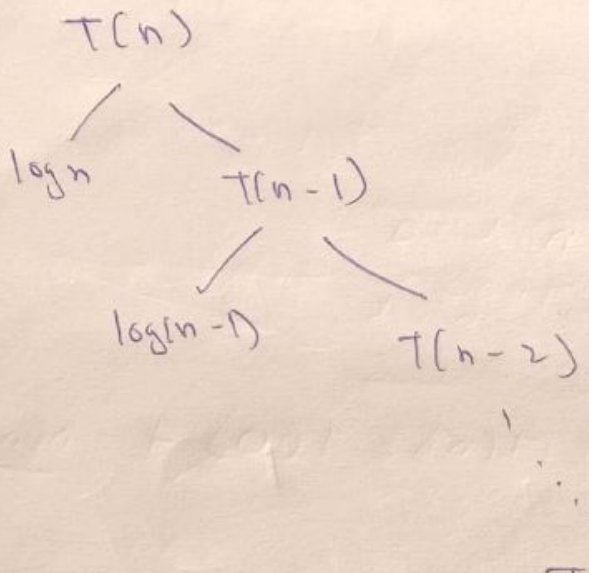
```
void Test(int n)
{
    if(n > 0)
    {
        for(i=1; i<n; i=i*2)
        {
            printf("%d", i);
        }
        Test(n-1);
    }
}
```

$\log_2 n \leftarrow$ (the `for` loop)

$T(n-1) \leftarrow$ `Test(n-1);`

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

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



$\log n + \log(n-1) + \ldots + \log 1$

$\Rightarrow \log n!$

$O(n \log n)$

Induction or substitution method

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

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

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

$T(n) = T(0) + \log(n-1) + \log 1 + \ldots + \log n$

$= T(0) + \log n!$

$\Rightarrow T(n) = 1 + \log n!$

$O(n \log n)$

$T(n) = T(n-1) + 1 \rightarrow O(n)$

$T(n) = T(n-1) + n \rightarrow O(n^2)$

$T(n) = T(n-1) + \log n \rightarrow O(n \log n)$

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

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

$O(n)$

$T(n) = T(n-100) + 1 \rightarrow \frac{n}{100}$

100 every step

$O(n)$

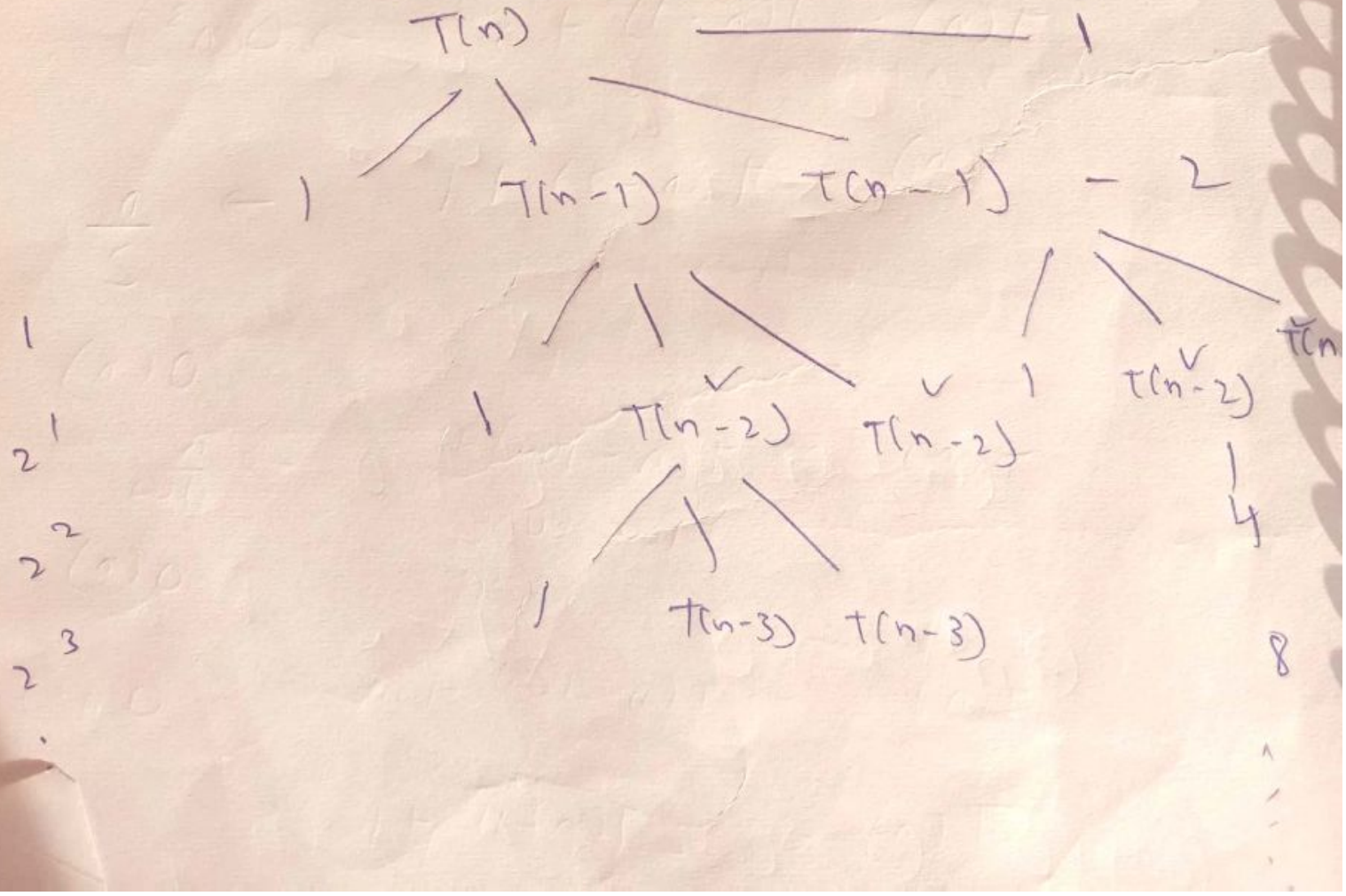
$T(n) = T(n-100) + n$ $O(n^2)$

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

```
{ if(n>0)
1 ← { printf("%d",n);
T(n-1) ← Test(n-1);
T(n-1) ← Test(n-1);
3 3
```

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

$$T(n) = \begin{cases} 1 & n = 0 \\ 2T(n-1)+1 & n > 0 \end{cases}$$



$2^{k+1} - 1$

$n = k$.

$2^{n+1} - 1$

$\frac{a(r^{k+1} - 1)}{r - 1}$

$O(2^n)$.

Q. $T(n) = \begin{cases} 1 & n = 0 \\ 2T(n-1) + 1 & n > 0 \end{cases}$

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

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

$= 4T(n-2) + 3$

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

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

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

$T(n) = 2^{n+1} - 1$ $\quad O(2^n)$ $\Theta(2^n)$

Master Theorem for decreasing function

$T(n) = (T(n-1)) + 1 \rightarrow O(n)$

$T(n) = T(n-1) + n \rightarrow O(n^2)$

$T(n) = T(n-1) + n\log n \rightarrow O(n \log n)$

$T(n) = 2T(n-1) + 1 \rightarrow O(2^n)$

$T(n) = 3T(n-1) + 1 \rightarrow O(3^n)$

$T(n) = 2T(n-1) + n \rightarrow O(n2^n)$

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

$a > 0$
$b > 0$

$f(n) = O(n^k)$
$k \geq 0$

* if $a = 1$,

$O(n^{k+1})$ $O(n * f(n))$

* if $a > 1$, $O(a^{n/b} n^k)$

* if $a < 1$, $O(n^k)$ $O(f(n))$

Dividing functions →

T(n) Algorithm Test(int n)

{ if(n > 1)

1 — { printf("%d", n);

T(n/2) — T(n/2);

3

3

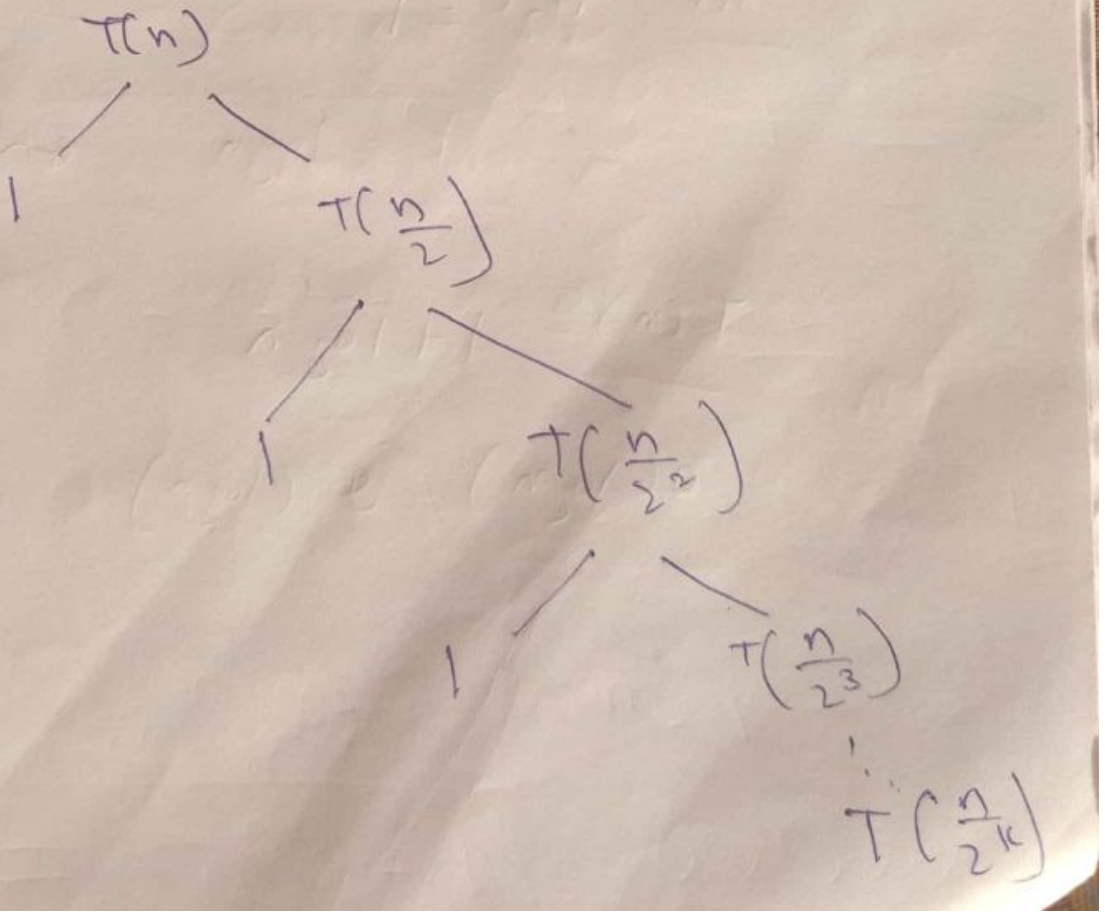
Decreasing fns.

[n
  n - 1

n/2
√n

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

$$T(n) = \begin{cases} 1 & n = 1 \\ T(\frac{n}{2}) + 1 & n > 1 \end{cases}$$



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

$2^k = n$

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

$O(\log n)$

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

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

let $n/2^k = 1$

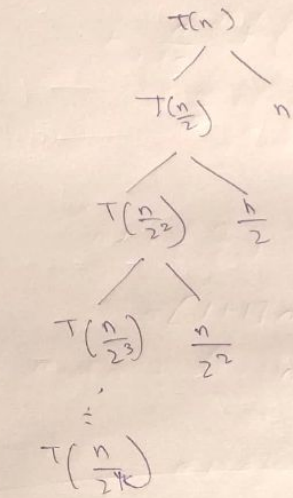
$\rightarrow \quad k = \log n \quad$ ~~$\theta(\log n)$~~

$T(n) = 1 + \log n$

$O(\log n) \quad \theta(\log n)$

Tree method
Substitution/
Induction method.

$$T(n) = \begin{cases} 1 & n = 1 \\ T(\frac{n}{2}) + n & n > 1 \end{cases}$$



$n = 2^k$

$n\left(1 + \frac{1}{2} + \frac{1}{2^2} + \dots + \frac{1}{2^k}\right)$

$n\left(\frac{\frac{1}{2}(\text{[illegible]}^k + 1)}{+\frac{1}{2}}\right)$

$= n\left(\frac{\text{[illegible]}}{2}\right)$

$= n \sum_{j=0}^{k} \frac{1}{2^j} =$

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

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

$n = 2^k$

$k = \log n$

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

$= 1 + 2n$

$O(n)$.

$T(n)$ — void Test (int n)

{ if (n > 1)

{ for (i=0; i<n; i++)

n —— stmt;

T(n/2) —— Test(n/2);

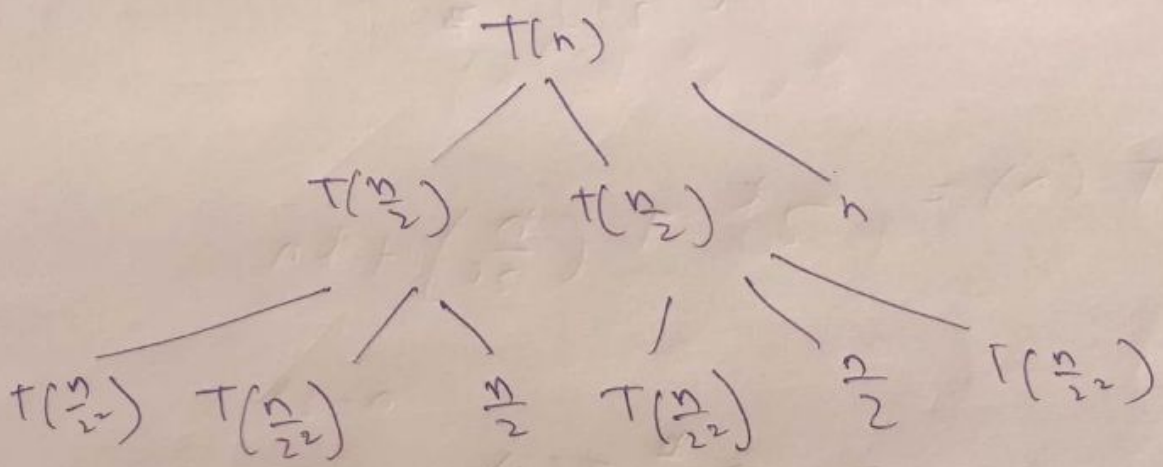
T(n/2) —— Test(n/2);

}

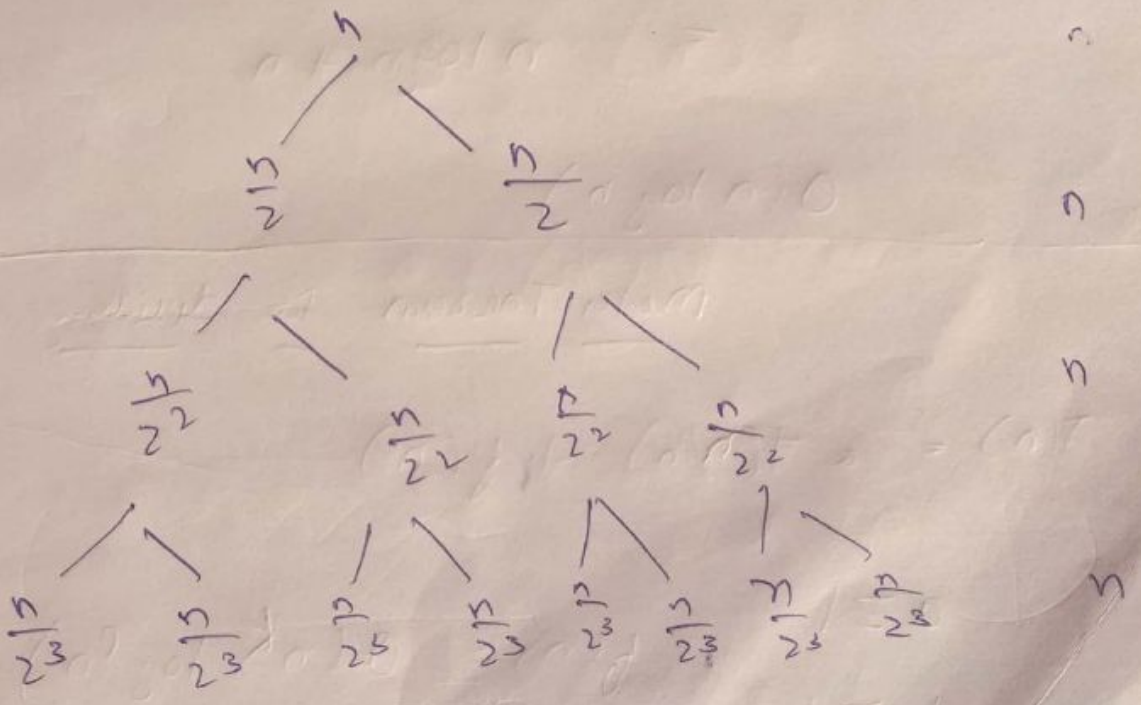
}

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

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\frac{n}{2}\right) + n & n > 1 \end{cases}$$



or.



n k

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

$n \log n$

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(\frac{n}{2}) + n & n > 1 \end{cases}$$

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

$$T(\tfrac{n}{2}) = 2T(\tfrac{n}{2^2}) + \tfrac{n}{2}\,2$$

$$T(n) = 2^k T(\tfrac{n}{2^k}) + kn$$

$$= n\,2^k T(1) + n \log n \qquad n = 2^k$$

$$= n \log n + n$$

$\rightarrow O(n \log n)$

Master Theorem for dividing fns →

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

$a \geq 1$

$b > 1$

$f(n) = \Theta(n^k \log^p n)$

① $\log_b a$

② k

Case I → $\log_b a > k$

$O(n^{\log_b a})$

Case II → $\log_b a = k$

(a) $p > -1$

$O(n^k \log^{p+1} n)$

(b) $p = -1$

$O(n^k \log \log n)$

(c) $p < -1$

$O(n^k)$

Case III → if $\log_b a < k$

(a) if $p \geq 0$

$O(n^k \log^p n)$

(b) $p < 0$

$O(n^k)$

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

$a = 2$

$b = 2$

$f(n) = \theta(1)$

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

$k = 0$

$a = 2 \quad b = 2$

$\log_2 2^2 > 1$

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

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

① $\log_a b$

② k

$\theta(n^{\log_b a})$

$\log_2 4 > 1 \quad p = 0$

$\theta(n^2)$

$T(n) = 8T(n/2) + n$

$O(n^3)$

Q. $T(n) = 9\,T(\frac{n}{3}) + 1$

$\log_3 9 > 0$

$2 > 0$

$O(n^2)$

Q. $T(n) = 2T(\frac{n}{2}) + \frac{n}{\log n}$

$\log_2 2 = 1 \quad k = 1 \quad p = -1$

$O(n \log\log n)$

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

$\log_2 2 = 1 < k = 2$

$O(n^2)$

Recurrence Relation for Root Fn →

$$T(n) = \begin{cases} 1 & n = 2 \\ T(\sqrt{n}) + 1 & n > 2 \end{cases}$$

```
void Test (int n)
{ if (n > 2)
  { Stmt;
T(√n)   Test (√n);
  }
}
```

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

$$T(n) = \begin{cases} 1 & n = 2 \\ T(\sqrt{n}) + 1 & n > 2 \end{cases}$$

(at each 2 the root $\sqrt{n}$)

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

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

$= T(n^{1/2^k}) + k$

$n = 2^m \Rightarrow m = \log_2 n$

$T(2^m) = T(2^{m(\frac{1}{2}^k)}) + k$

$= T(\quad^{m/2^k}) + k$

$T(2^{m/2^k}) = T(2)$

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

$2^k = m$

$k = \log_2 m$ $\quad k = \log \log_2 n$

$\Rightarrow$ ~~$T(2^m) = 1 + k$~~.

$\Rightarrow O(\log \log_2 n)$

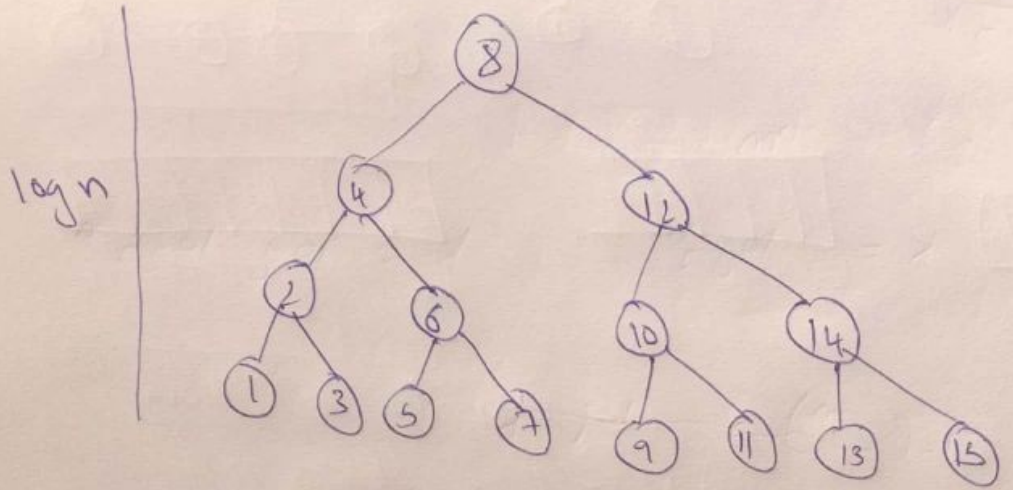
Binary Search

(Divide And Conquer)

Break large problem into small one's. Solve and combine them.

```
int BinSearch (A, n, key)
{ l=1 ; h=n ;
  while (l<=h)
  { mid = (l+h)/2;
    if (key == A[mid])
        return mid;
    if (key < A[mid])
        h = mid - 1;
    else
        l = mid + 1;
  } return 0; }
```

3	6	8	12	14	17	25	29	31	36	42	47	53	55	62
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



$\log_2 16 = 4$.

min O(1)

max O(log(n))

Recursive method →

```
Algorithm RBinsearch (l, h, key)
{
    if (l == h)
    { if (A[l] == key)
        return l;
      else
        return 0;
    }
    else
    { mid = (l+h)/2;
      if (key == A[mid])
        return mid;
```

return RBinSearch (l, mid-1, key);

or

return RBinSearch (mid+1, h, key);

}}

$$t(n) = \begin{cases} 1 & n = 1 \\ T(\frac{n}{2}) + 1 & n > 1 \end{cases}$$

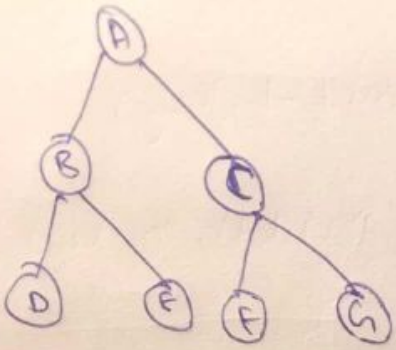
$T(n/b) + n^0 \log^0 n$

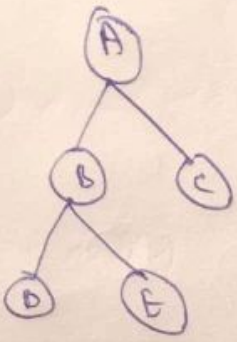
$\log 2 = 1$

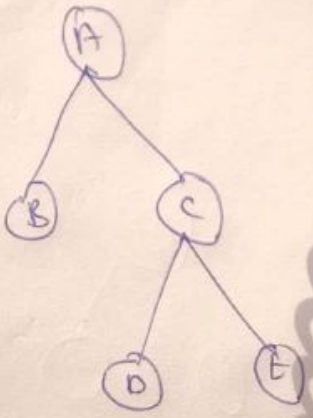
~~O(n)~~ $O(n \log n) = O(\log n)$

Heap →

1. Array Representation of BT
2. Complete BT
3. Heap
4. Insert & Delete
5. Heap sort
6. Heapify
7. Priority Queue.







A	B	C	D	E	F	G
1	2	3	4	5	6	7

A	B	C	D	E
1	2	3	4	5

A	B	C	-	-	B	E	
1	2	3	4	5	6	7	

Node is at index → i

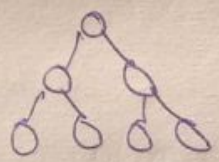
left child is at → 2 * i

right child is at → 2 * i + 1

parent is at → $\left\lfloor \frac{i}{2} \right\rfloor$ floor

Full Binary Tree → Fully occupied at any height

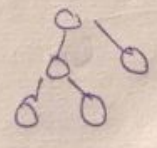
(Also complete)



h $2^{h+1} - 1$ nodes

Complete binary tree → No missing element

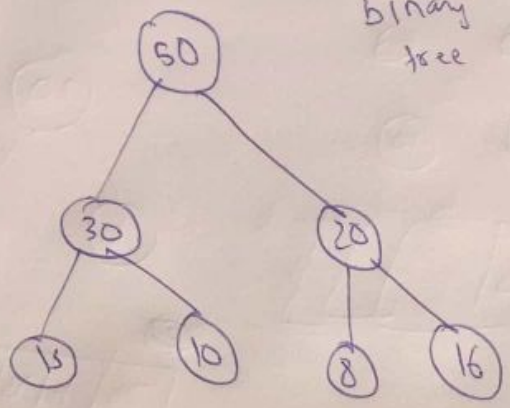
(left to right)



log n

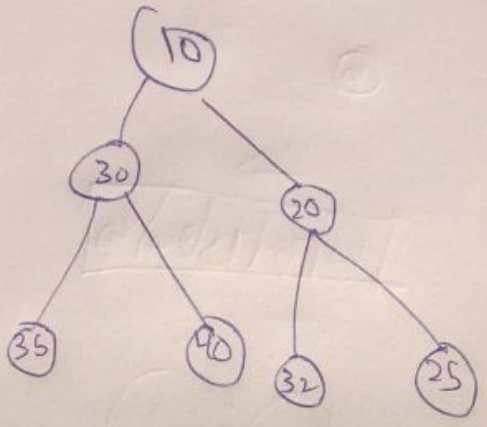
Max Heap

Complete binary tree



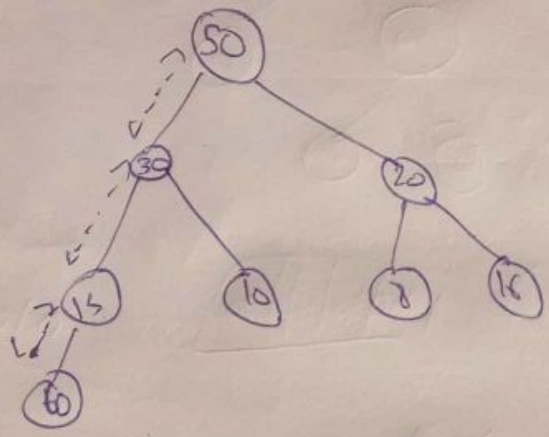
50	30	20	15	10	8	16

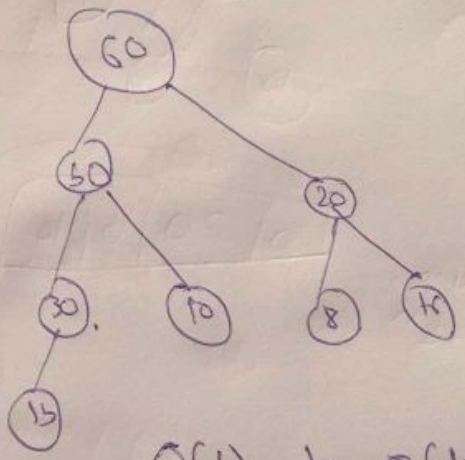
Min Heap



10	30	20	35	40	32	35

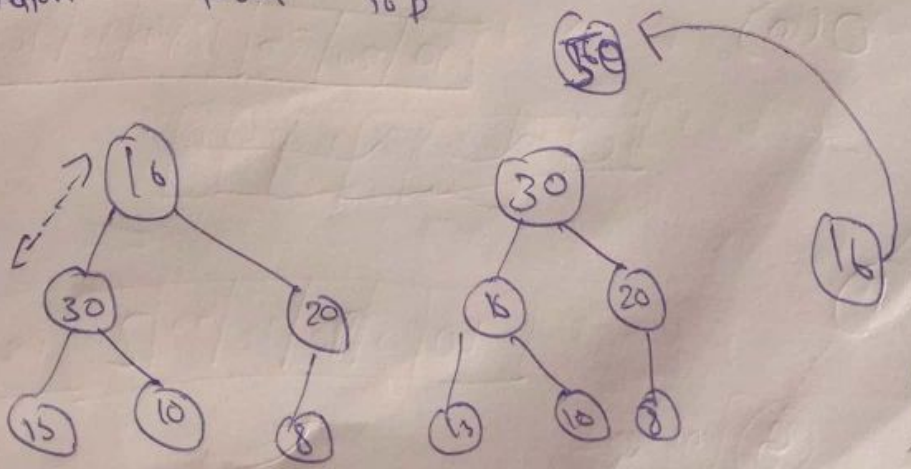
Insert 60



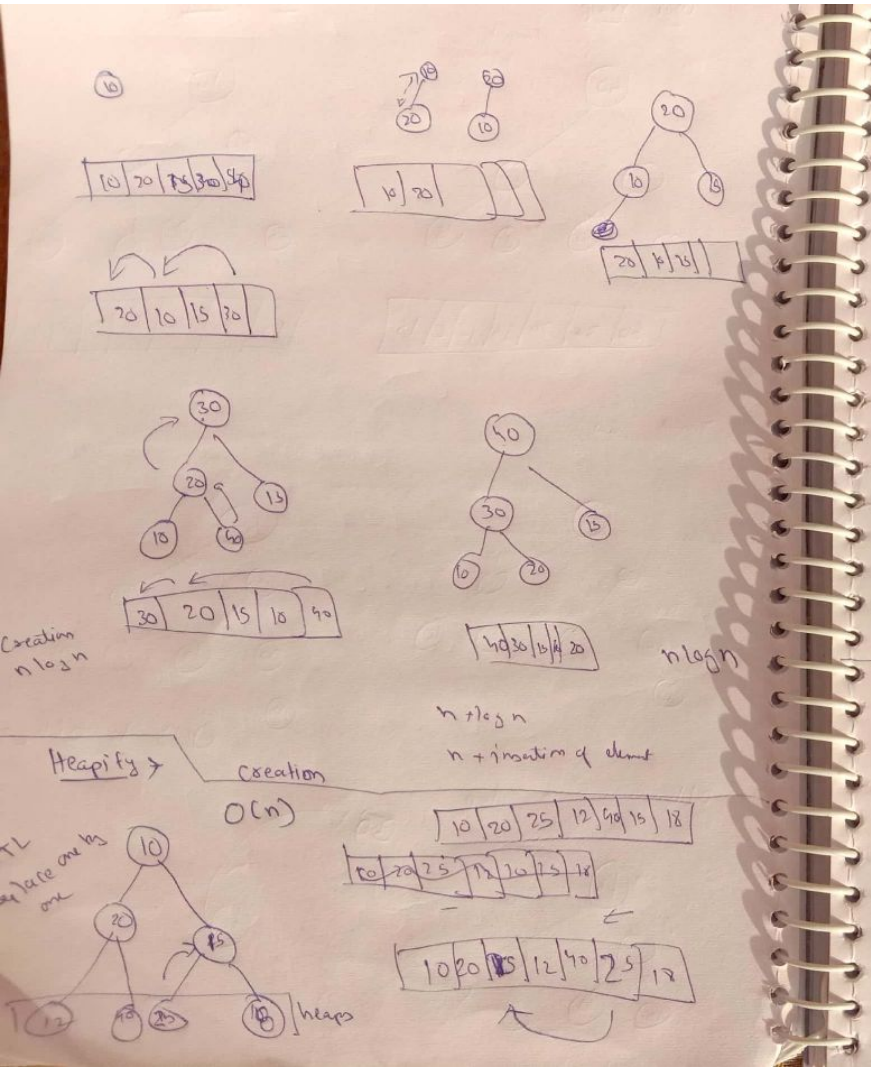


O(1) to O(log n)

Deletion from top



log n



Priority Queue → Number with higher priority must be inserted & deleted

Smaller Number
Higher Priority

Larger Numbers
higher priority

8 6 3 10 5 4 7

8 6 3 10 5 4 7

min Heap

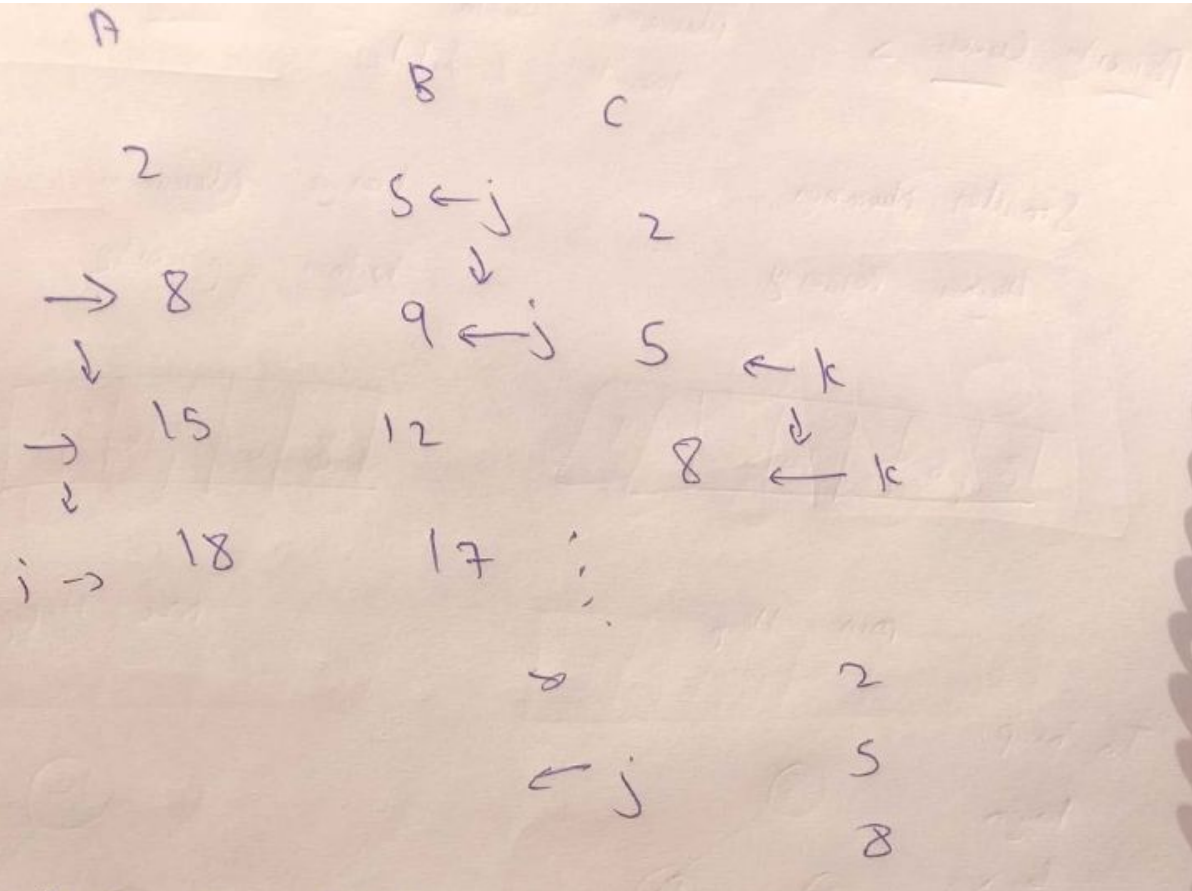
max Heap

In heap,
log n
log 1
other,
one of n
after insertion
or deletion

Heap is best to implement priority Queue.

Merge →

A	B	C
i → 2	j → 5	2 ← k
8	9	
15	12	
18	17	



Algorithm merge (A, B, m, n)

{

i=1 j=1 k=1

while (i<=m && j<=n)

{ if(A[i] < B[j])

C[k++] = A[i++];

else

C[k++] = B[j++]; }

}

for (; j<=m ;i++)

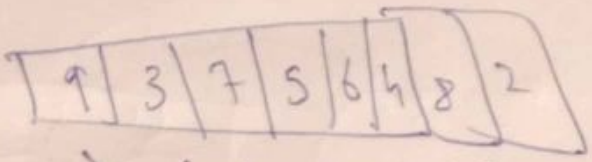
C[k++] = A[i];

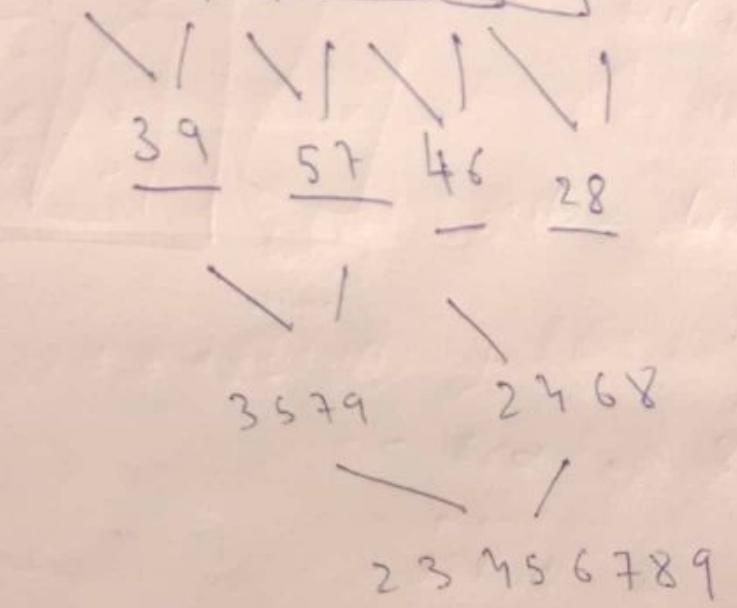
for (; j<=n ;j++)

C[k++] = B[j];

}

$O(m+n)$





n log n

Algorithm Merge sort (l, h)

```
{ if (l<h)
  { mid = (l+h)/2;
    mergesort (l, mid),
    mergesort (mid+1, h);
    Merge (l, mid, h);
  }
}
```

O(n log n)

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

$\log 2^2 = 1$ $O(n \log n)$

Pros and Cons

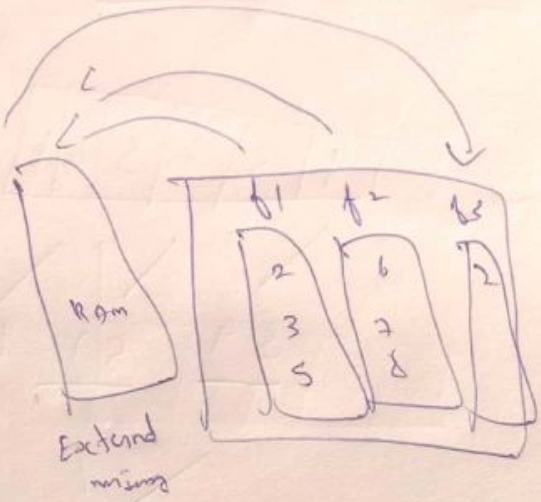
Pros

① Large size list

② linked list

③ External sorting

④ Stable



8 a b 8 - -

After sorting a b 8 8 - -

Cons

① Extra Space

② No small problem

Insertion Sort $O(n^2)$ Small problem

Merge Sort $(n \log n)$ slower

$n \leq 15$
slower

Bubble Sort $O(n^2)$

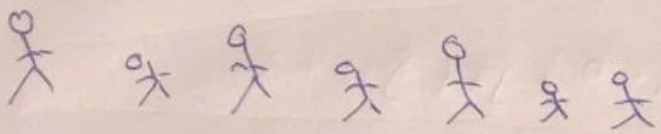
③ Recursive

○ $(n + \log^n)$

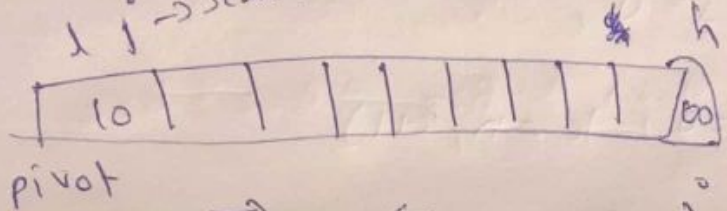
$O(n)$

Quick Sort

Divide & Conquer



i → Searches for element greater than i



→ ←

j → Smaller than j

then exchange i, j

while j > i

```
Quicksort (l, h)
{ if (l < h)
  { j = partition (l, h);
    Quicksort (l, j);
    Quicksort (j+1, h);
  }
}
```

Best case

$O(n \log n)$

Worst case

$O(n^2)$

This can be solved by selecting middle as pivot

merge

$\overline{O(n \log n)}$

Strassen's Matrix Multiplication →

(Divide And Conquer)

$$C_{ij} = \sum_{k=1}^{m} A_{ik} * B_{kj}$$

```
for (i=0; i<n; i++)
  { for (j=0; j<n; j++)
    { c[i,j]=0;
      for (k=0; k<n; k++)
      { c[i,j] += [A[i,k]] * B[k,j];
        } } }
```

$O(n^3)$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

2×2 $\quad$ 2×2

$c_{11} = a_{11} * b_{11} + a_{12} * b_{21}$

$c_{12} = a_{11} * b_{12} + a_{12} * b_{22}$

$c_{21} = a_{21} * b_{11} + a_{22} * b_{21}$

$c_{22} = a_{21} * b_{12} + a_{22} * b_{22}$

$A[a_{11}] \quad b = [b_{11}]$

$c = [a_{11} * b_{11}]$

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

4×4

A_{21} A_{22}

4×4

2×2

B_{11} B_{12}

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

4×4

B_{21} B_{22}

4×4

2×2

Algorithm mm(A, B, n)

{

if ($n \leq 2$)

{ $C = 4$ formulas

}

else

{

mid = $n/2$

mm ($A_{11}, B_{11}, \frac{n}{2}$) + mm ($A_{12}, B_{21}, \frac{n}{2}$)

mm ($A_{11}, B_{12}, \frac{n}{2}$) + mm ($A_{12}, B_{22}, \frac{n}{2}$)

$C_{11} = A_{11} * B_{11} + A_{12} * B_{21}$

$C_{12} = A_{11} * B_{12} + A_{12} * B_{22}$

Direct Multiplication (8 multiplications)

$C_{11} = A_{11} * B_{11} + A_{12} * B_{21}$

$C_{12} = A_{11} * B_{12} + A_{12} * B_{22}$

$C_{21} = A_{21} * B_{11} + A_{22} * B_{21}$

$C_{22} = A_{21} * B_{12} + A_{22} * B_{22}$

Algorithm mm(A, B, n)

{ if (n ≤ 2)

{ C = 4 formulas

}

else

{ mid = n/2

mm $(A_{11}, B_{11}, \frac{n}{2})$ + mm $(A_{12}, B_{21}, \frac{n}{2})$

mm $(A_{11}, B_{12}+\frac{n}{2})$ + mm $(A_{12}, B_{22}, \frac{n}{2})$

mm $(A_{21} * B_{11}, \frac{n}{2})$ + mm $(A_{22}, B_{21}, \frac{n}{2})$

mm $(A_{21}, B_{12}, \frac{n}{2})$ + mm $(A_{22}, B_{22}, \frac{n}{2})$

}

}

Internally uses Stack insumes more space

33

$\theta(n^3)$

$$T(n) = \begin{cases} 1 & n \leq 2 \\ 8\,T(\frac{n}{2}) + n^2 & n > 2 \end{cases}$$

$\log_2 8$

$3 > 2$

Strassen (7 multiplication)

(+, - takes less time compared to ×)

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$Q = (A_{21} + A_{22}) B_{11}$$

$$R = A_{11}(B_{12} - B_{22})$$

$$S = A_{22}(B_{21} - B_{11})$$

$$T = (A_{11} + A_{12}) B_{22}$$

$$U = (A_{21} - A_{11})(B_{11} + B_{12})$$

$$V = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$C_{11} = P + S - T + V$$

$$C_{12} = R + T$$

$$C_{21} = Q + S$$

$$C_{22} = P + R - Q + U$$

$$T(n) = \begin{cases} 1 & n \leq 2 \\ 7T\left(\frac{n}{2}\right) + n^2 & n > 2 \end{cases}$$

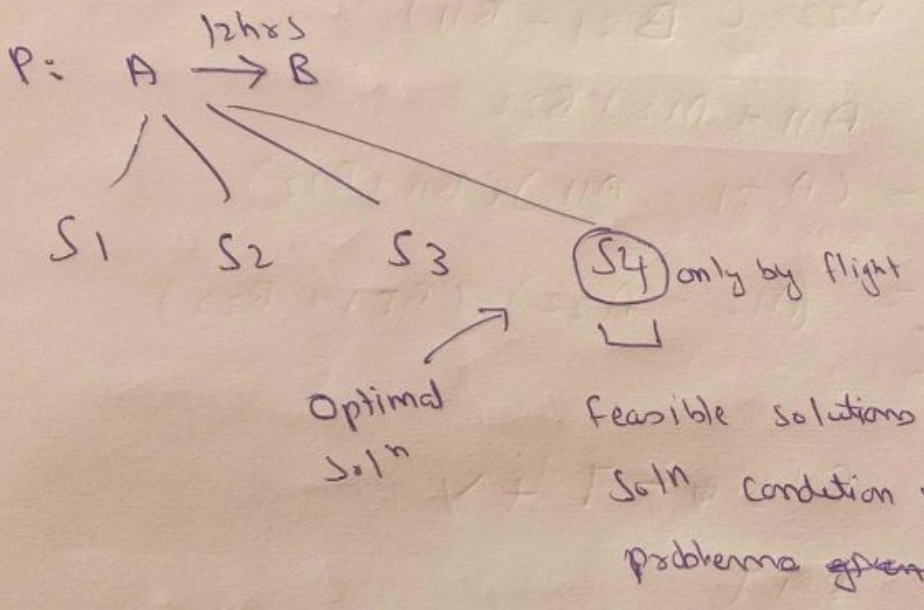
$$\log_2 7 = 2.81 > 2$$

$$O(n^{2.81}).$$

Greedy Method

One of the approach (design) for solving similar problems.

- Solving optimization Problems (which requires either min or max result).



min-cost Minimization Problem

Optimal Soln → Soln which is feasible and giving best result.

Optimization Problem → which requires either minimum or maximum result

Optimization Problems →

1. Greedy Method
2. Dynamic Programming
3. Branch and Bound

Algorithm Greedy (a, n)

```
{   for i=1 to n do
    {   x = select (a);
        if Feasible (x) then
            solution += x;
    }
}
```

n = 5

a_1	a_2	a_3	a_4	a_5

when we select the known methods & quickly solve the problem that approach is known to be greedy.

Knapsack Problem Greedy method

Objects	0	1	2	3	4	5	6	7
Profits P	10	5	15	7	6	18	3	
weights w	2	3	5	7	1	4	1	

n = 7
m = 5

Constraint

Fill objects in bag such that profit is maximized.

x 0-1

x_1 x_2 x_3

$0 \leq x \leq 1$

~~First~~ ~~fill~~ objects ~~with~~ Small weights.

Greedy first select the method

* Select highest profit / weight.

objects	0	1	2	3	4	5	6	7
Profits	10	5	15	7	6	18	3	
weights		2	3	5	7	1	4	1

P/w

5	1.3	3	1	6	4.5	3

1	2/3	1	0	1	1	1
x_1	x_2	x_3	x_4	x_5	x_6	x_7

15 - 1 = 14 kg

14 - 2 = 12

12 - 4 = 8

8 - 5 = 3

3 - 1 = 2

2 - 2 = 0

~~Profit~~
weight = $1 \times 2 + \frac{2}{3} \times 3 + 1 \times 5 + 0 \times 7 + 1 \times 1 + 1 \times 4 + 1 \times 1$

$\sum x_i w_i$

$= 2 + 2 + 5 + 1 + 5$

$= 15$

Profit $= 10 \times 1 + \frac{2}{3} \times 5 + 1 \times 15 + 1 \times 6 + 1 \times 18 + 1 \times 3$

Constraint

$\sum x_i w_i \leq$ m Capacity (15)

objective

$\sum x_i p_i$

= max $= 10 + \frac{10}{3} + 39 + 3$

$= 52 + \frac{10}{3}$

$= 55.34.$

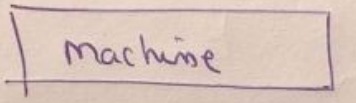
Zero knapsack

x can't be indivisible

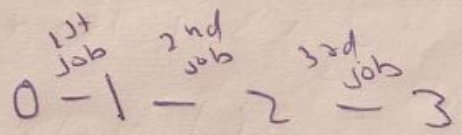
Job Sequencing with Deadlines

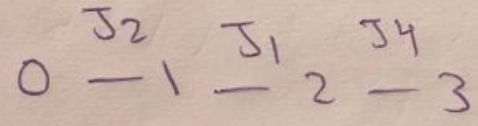
$n = 5$

Jobs	J_1	J_2	J_3	J_4	J_5
Profits	20	15	10	5	1
deadlines	2	2	1	3	3
	✓	✓	✗		



1 Unit (1 hr)





$\{ J_2, J_1, J_4 \}$

$J_2 \rightarrow J_1 \rightarrow J_4$

$J_1 \rightarrow J_2 \rightarrow J_4$

2 hrs

$20 + 15 + 5 = 40$

0-1-2-3

Job consider	Slot assign	Soln	profit
–	–	ϕ	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
J_5 ×	11	11	40

Q.

$n = 7$

Jobs	J_1	J_2	J_3	J_4	J_5	J_6	J_7
Profits	35	30	25	20	15	12	5
Deadlines	3	4	4	2	3	1	2

0 —J_4— 1 —J_3— 2 —J_1— 3 —J_2— 4

20 25 35 30

110.

Optimal Merge Pattern

(Greedy method)

A	B	C
3	5	3
8	9	5
12	11	8
20	16	9
m	n	11
		12
4	4	16
		20

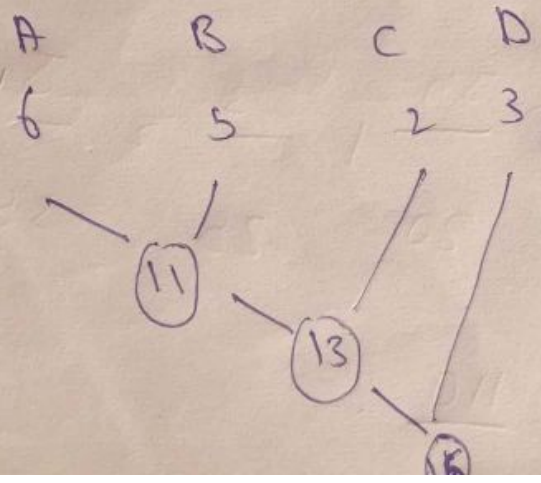
$O(m+n)$

List → A B C D

Size → 6 5 2 3

(4-way merge)

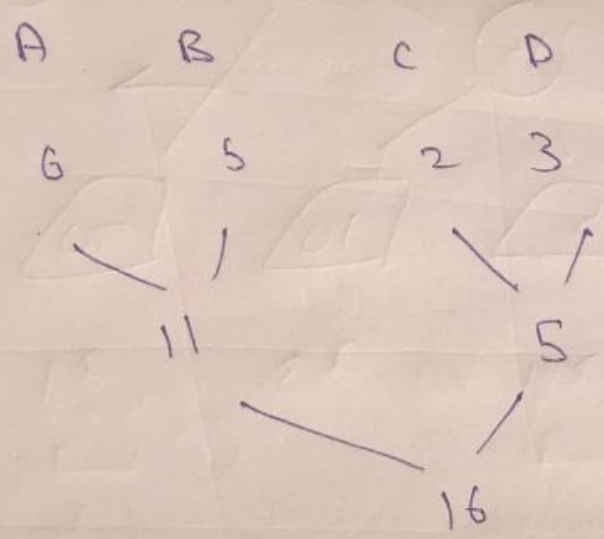
At a time 2, (2-way merge)



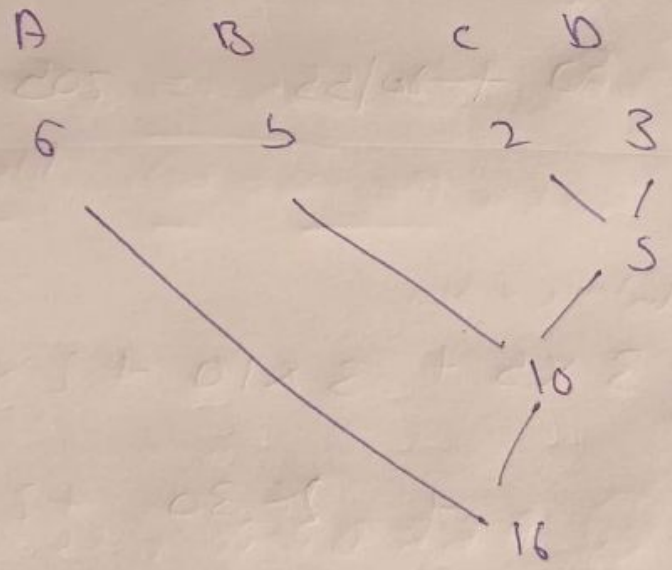
Total Cost

= 11 + 13 + 16

= 40



Total cost = 11 + 5 + 16 = 32



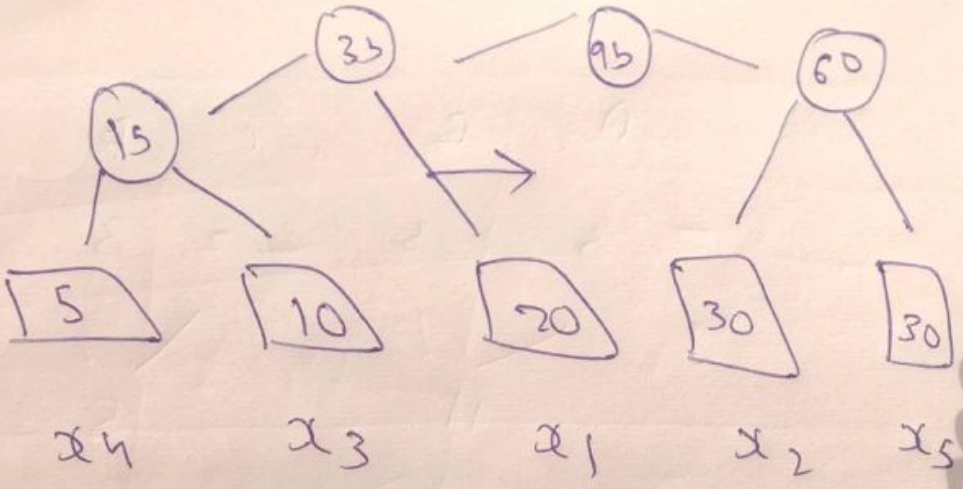
Total cost = 31

~~Optimal~~

Q. |1|5|1> → x_1 x_2 x_3 x_4 x_5

Sigs → 20 30 16 5 30

Greedy Method -



Smaller pairs 35 + 30 > 30 + 30

Total cost

= 50 + 55 = 205. for merging files

distance * size

3 x 5 + 3 x 10 + 2 x 20

+ 2 x 30 + 2 x 30

= 205

$\sum d_i * x_i$.

Hoffman Coding

(Greedy method using optimal merge pattern) →

Compression technique

Message → BCCABBDDAECCBBAEDDCC (length 20 sent using ASCII code 8 bits)

A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01001001

8X20 = 160 bits

Character	Count / frequency		Code
A	3	3/20	000
B	5	5/20	001
C	6	6/20	010
D	4	4/20	011
E	2	2/20	100

5

0-7

20 x 3

= 60 bits.

5 x 8 bit

characters

40

5 x 3

codes

15

msg → 60 bits

Table → 55 bits (then only decoding can be done)

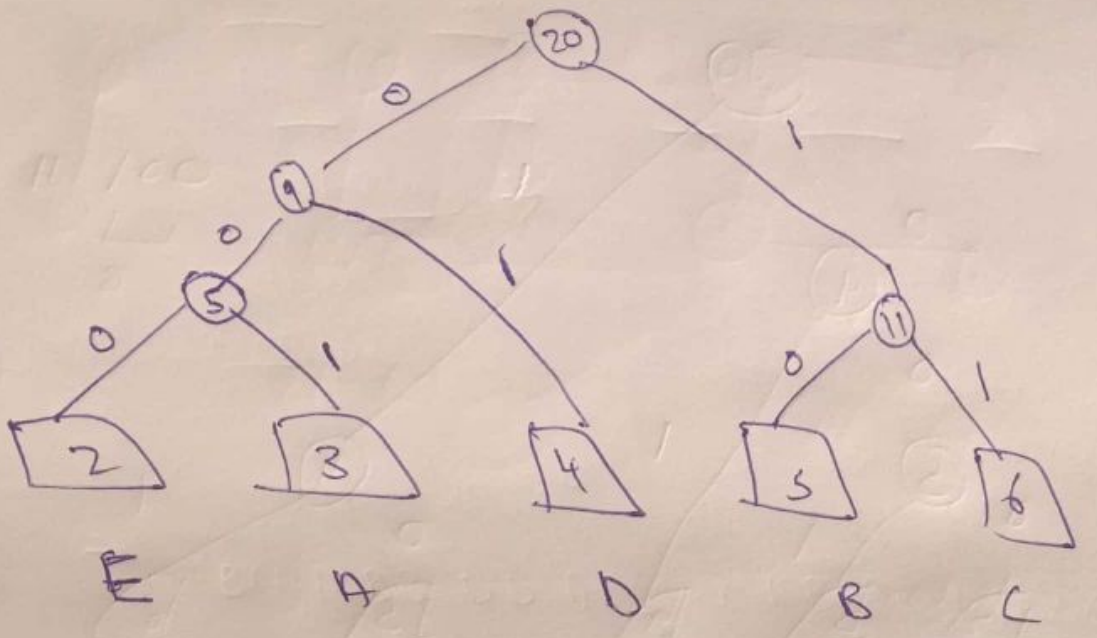
total size = 115 bits

table should be passed

Message →

B CC A BB DD A E CC BB AE DD CC

char	count	code		
A	3	001	9	
B	5	10	10	
C	6	11	12	
D	4	01	8	45 bits
E	2	000	6	
	20			



Optimal Merge Pattern Tree

Select Smaller

$MS_D \rightarrow 45$ bits

Tree / Table $\rightarrow 8 \times 5 + 12 = 52$

Total size = 97 bits.

$\sum d_i * f_i = 3 \times 2 + 3 \times 3 + 4 \times 2 + 10 + 12$

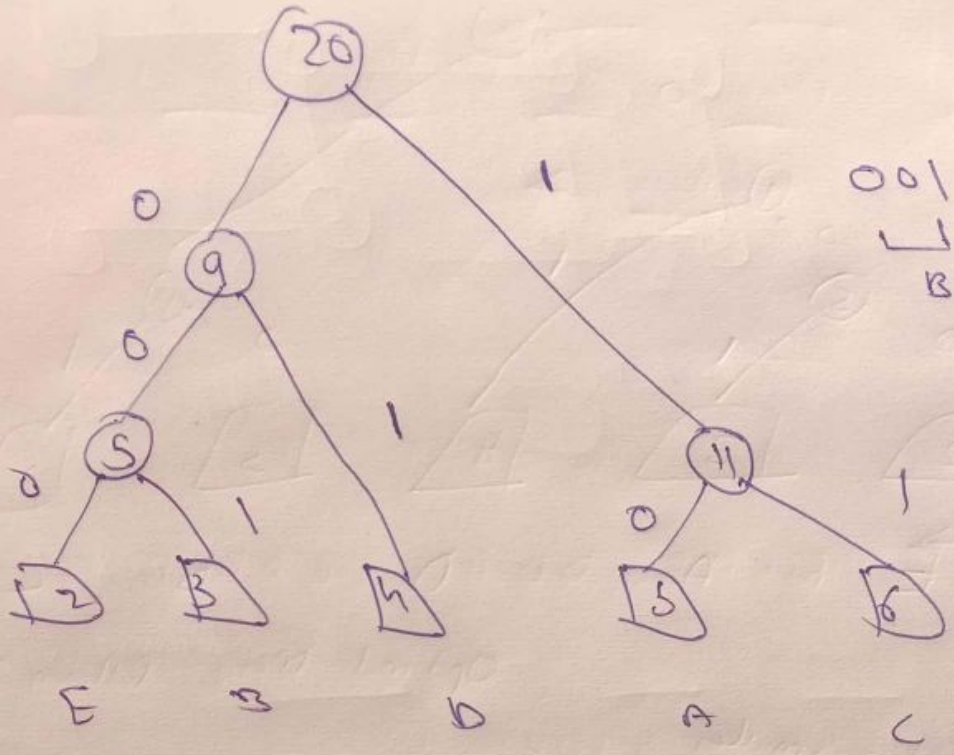
$= 45$

Decoding →

B C C D A C C B D A B C C D E A A E D A

001 11 11 01 10

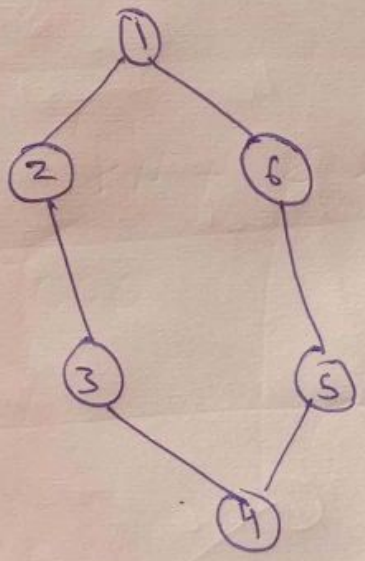
001



00| || || ...
⌊ ⌋
B

B C C . . . .

Minimum Cost Spanning Tree →



$G = (V, E)$

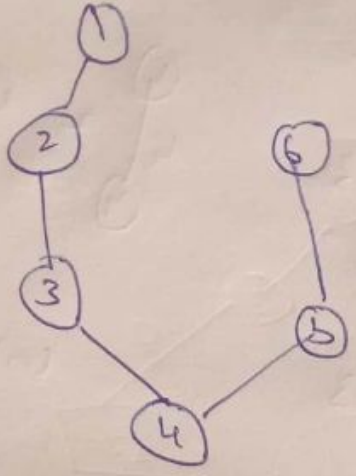
$V = \{1, 2, 3, 4, 5, 6\}$

$E = \{ (1,2), (2,3), (3,4) \ldots \}$

$|V| = 6 = n$

$n - 1 = 5$

all vertices but $n-1$ edges



$|V| = 6 = n$

$|V| - 1 = 5$

Subgraph $S \subseteq G$ Graph

$S = (V', E')$

$V' = V$

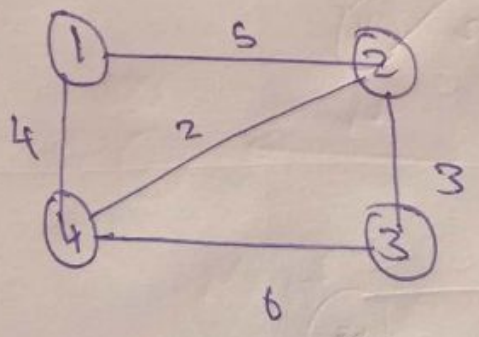
$E' = |V| - 1$

${}^6C_5 = 6$

No. of spanning trees $= {}^6C_5 = 6.$

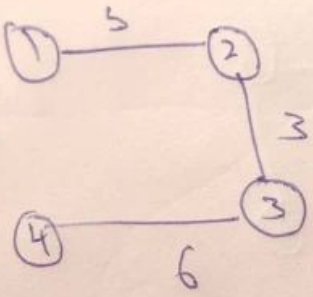
$= {}^{|E|}C_{|V|-1}$ - no. of cycles.

Q.

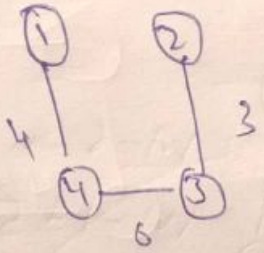


No. of spanning trees

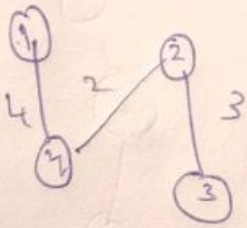
$= {}^5C_3 - 2 = 8.$



Cost = 14



Cost = 13



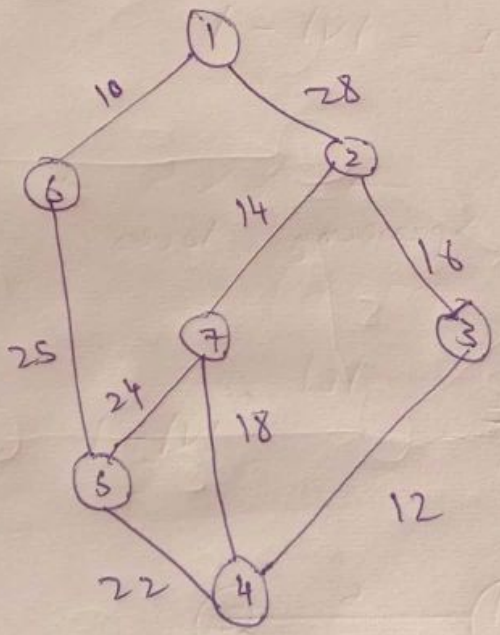
Cost = 9

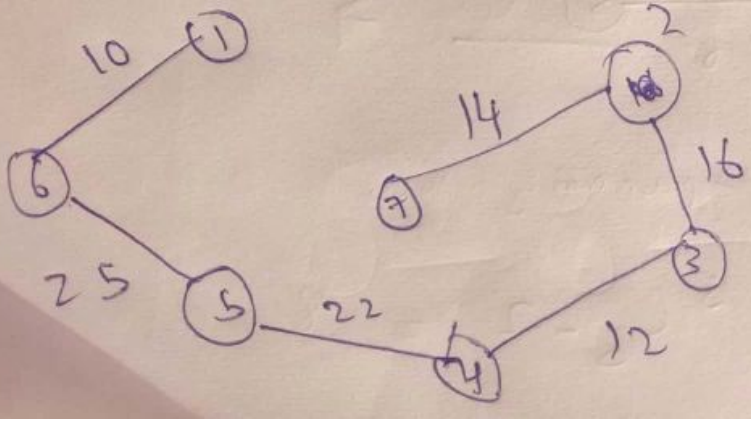
1. Prim's
2. Kruskal

(1) Prim's

select smallest edge

~~Select~~ Select next minimum which should be connected

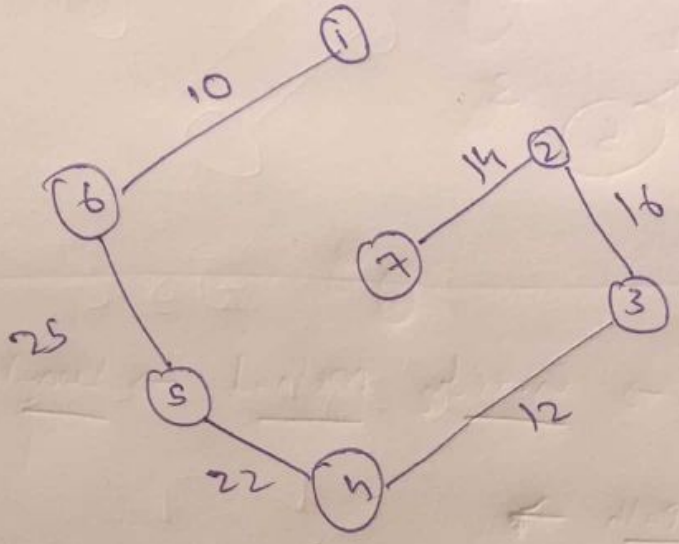




99

For non connected graphs we can't find spanning tree

Kruskal →



Cost = 99

4-7 ✗

If it is forming a cycle, discard it

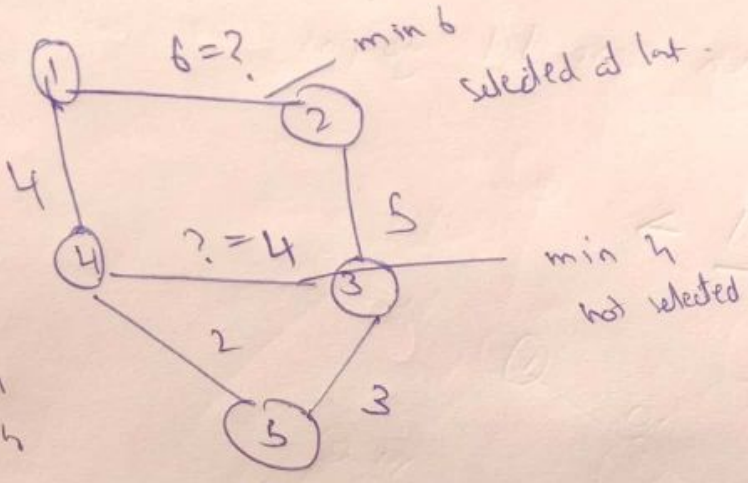
$O(|V||E|)$

$O(n \cdot e) = O(n^2)$ → kruskal

* Min Heap always gives min value.
  $O(n \log n)$

* kruskal can find spanning tree for all the components but not the entire graph.

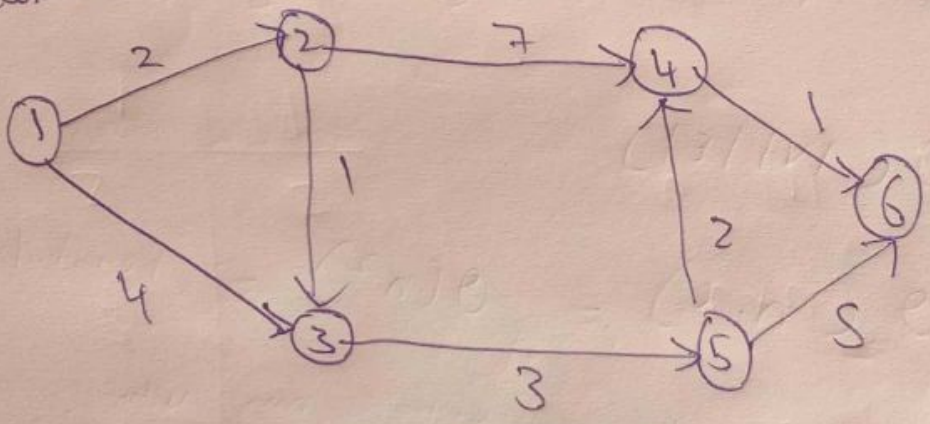
Q.



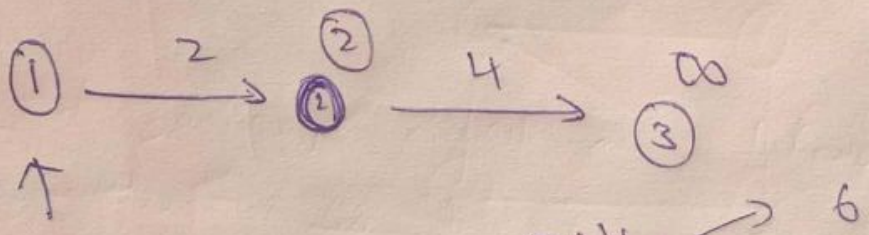
Q.

Dijkstra → Greedy method → Single Source Shortest Path →

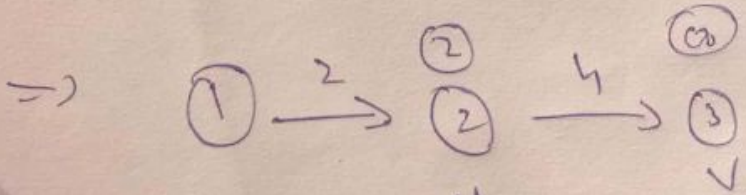
directed & non-directed



$2 < \infty$



$2+4 \rightarrow 6$



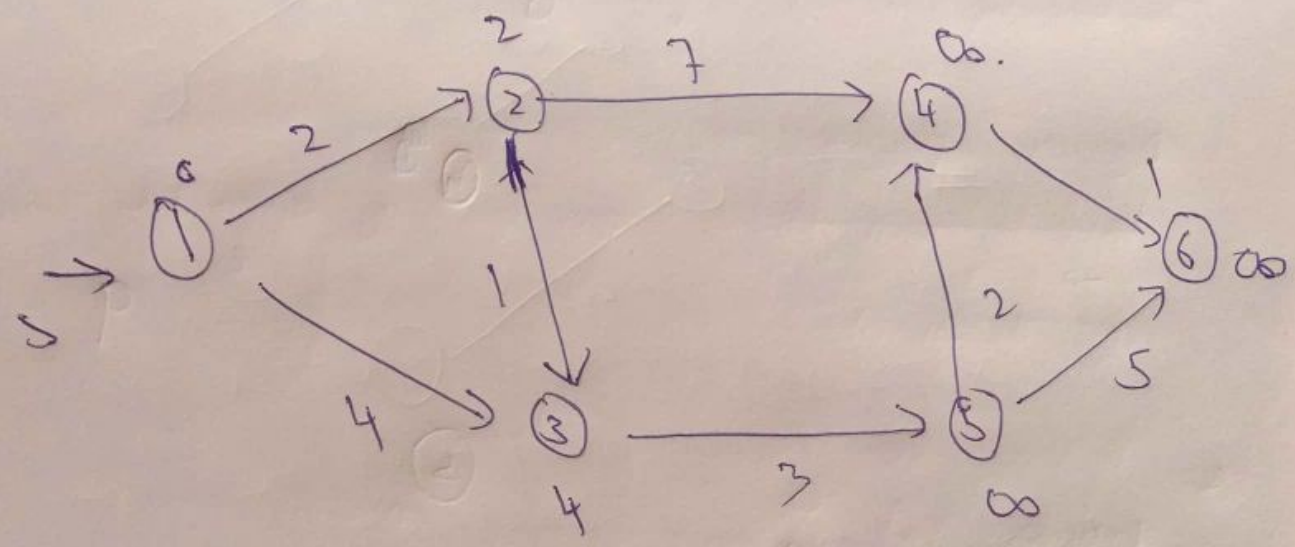
Relaxation.

Relaxation

if (d[u] + c(u,v) < d[v])

d[v] = d[u] + c[u,v]

Q.



Select Closed vertices

Smallest

⇒

0

0

9

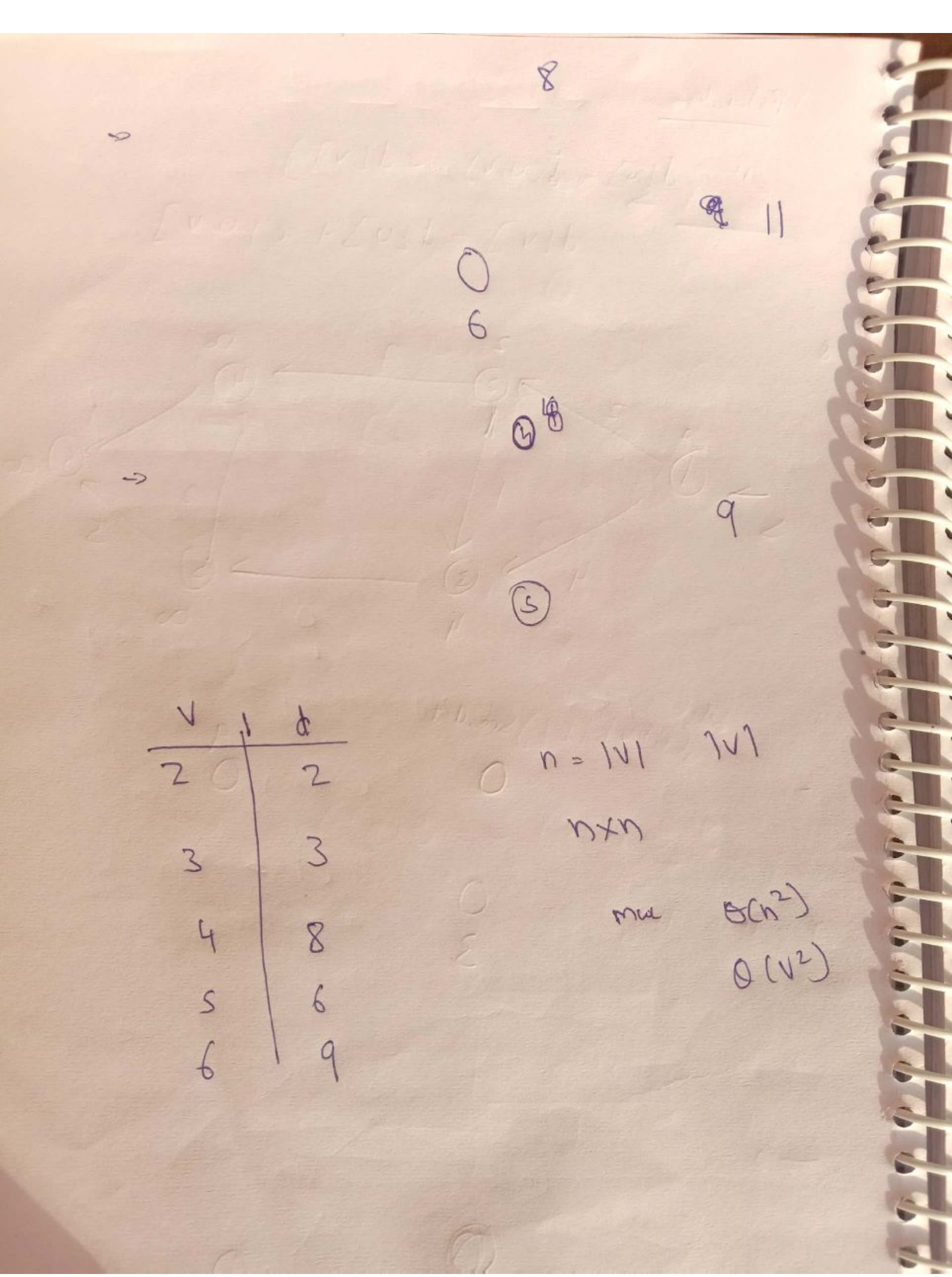
0

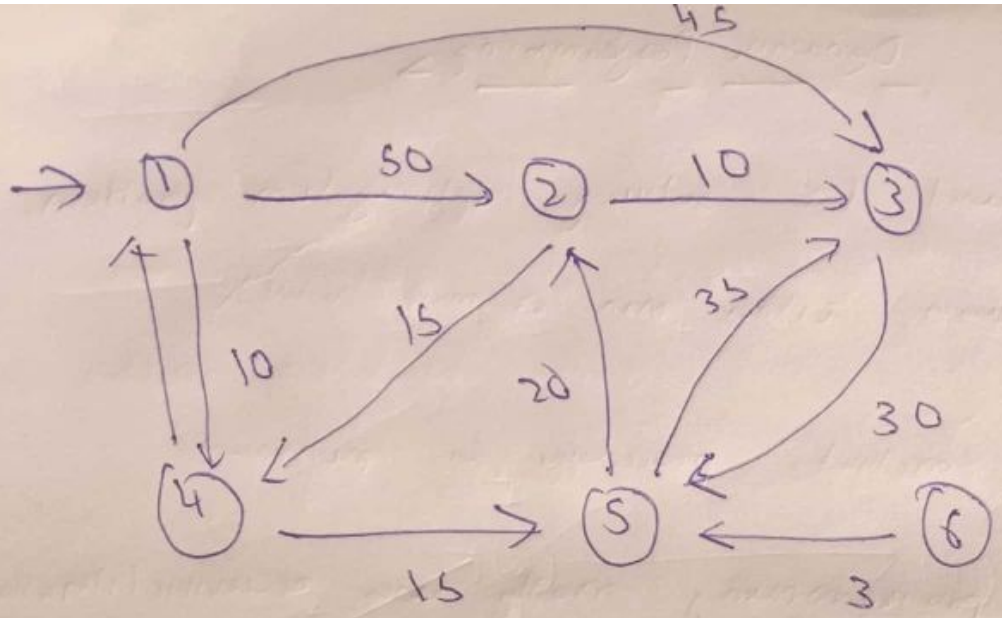
3

⇒

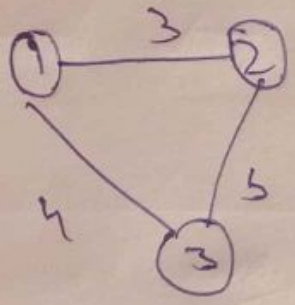
3

6

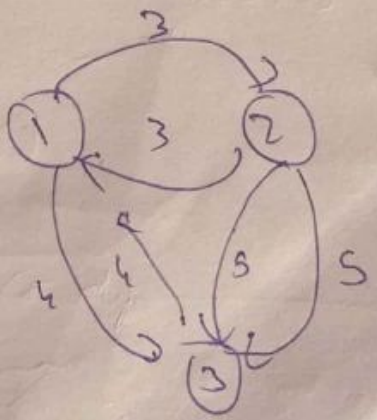




Selected vertex	2	3	4	5	6
4	50	45	(10)	∞	∞
5	50	45	(10)	(25)	∞
2	(45)	45	(10)	(25)	∞
3	(45)	45	(10)	(25)	



⇒



Drawbacks →

① It may or may not work in case of -ve edges.

Dynamic Programming →

It's also used for solving optimization problem.
(which requires either min or max result)

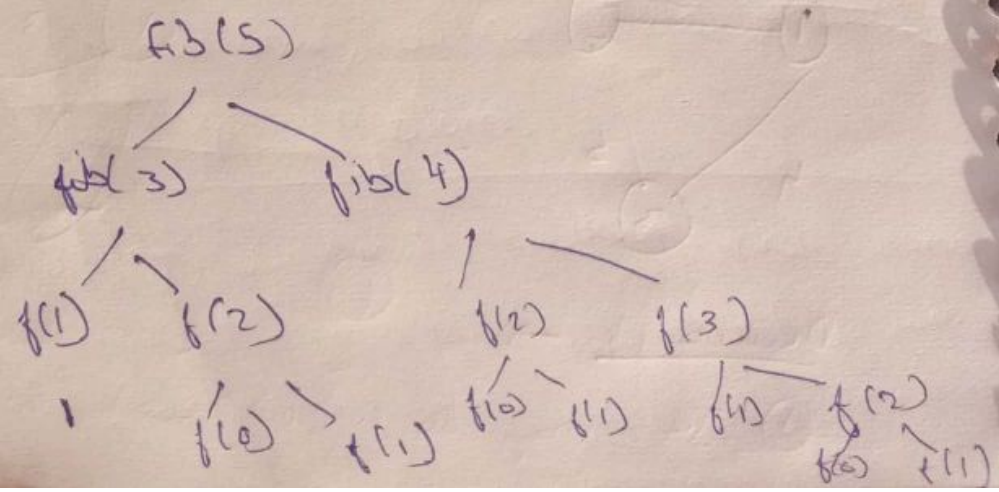
In Greedy method, procedure is minimum

Dynamic programming mostly uses recursive (iterative) methods and follows principle of optimality (sequence of decisions).

Q.

$$fib(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ fib(n-2) + f(n-1) & n > 1 \end{cases}$$

```
int fib (int n)
{ if (n<=1)
    return n;
  return fib(n-2) + f(n-1);
}
```



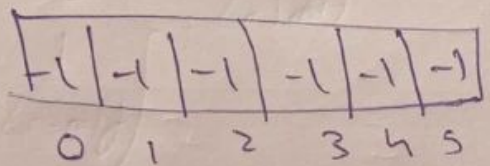
$T(n) = 2T(n-1)$ (Assume $T(n-2) = T(n-1)$)

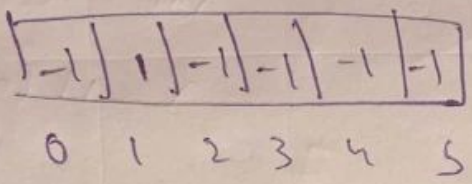
$+1$

$O(2^n)$

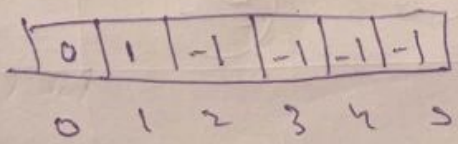
Once $f(2)$ is called it shouldn't be called again.

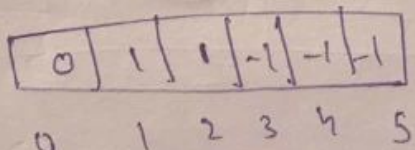
F

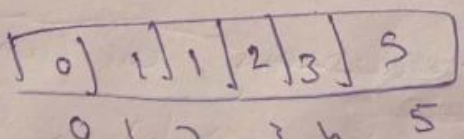




$2^n \longrightarrow n$ due to memoization







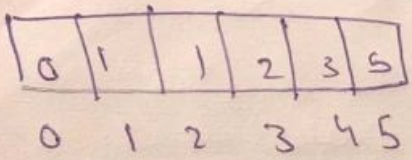
, ∨ memoization

6 calls. (earlier 13)

Fib(n) $\longrightarrow$ $n+1$ calls $\theta(n)$

Memoization follows top-down approach (starting from 5)

In dynamic programming, we can use memoization but mostly tabulation method is followed (iterative method).



$$f(n) = \begin{cases} 0 & n=0 \\ 1 & n=1 \\ f(n-2)+f(n-1) & n>1 \end{cases}$$

Bottom - Up Approach

Starting from 0.

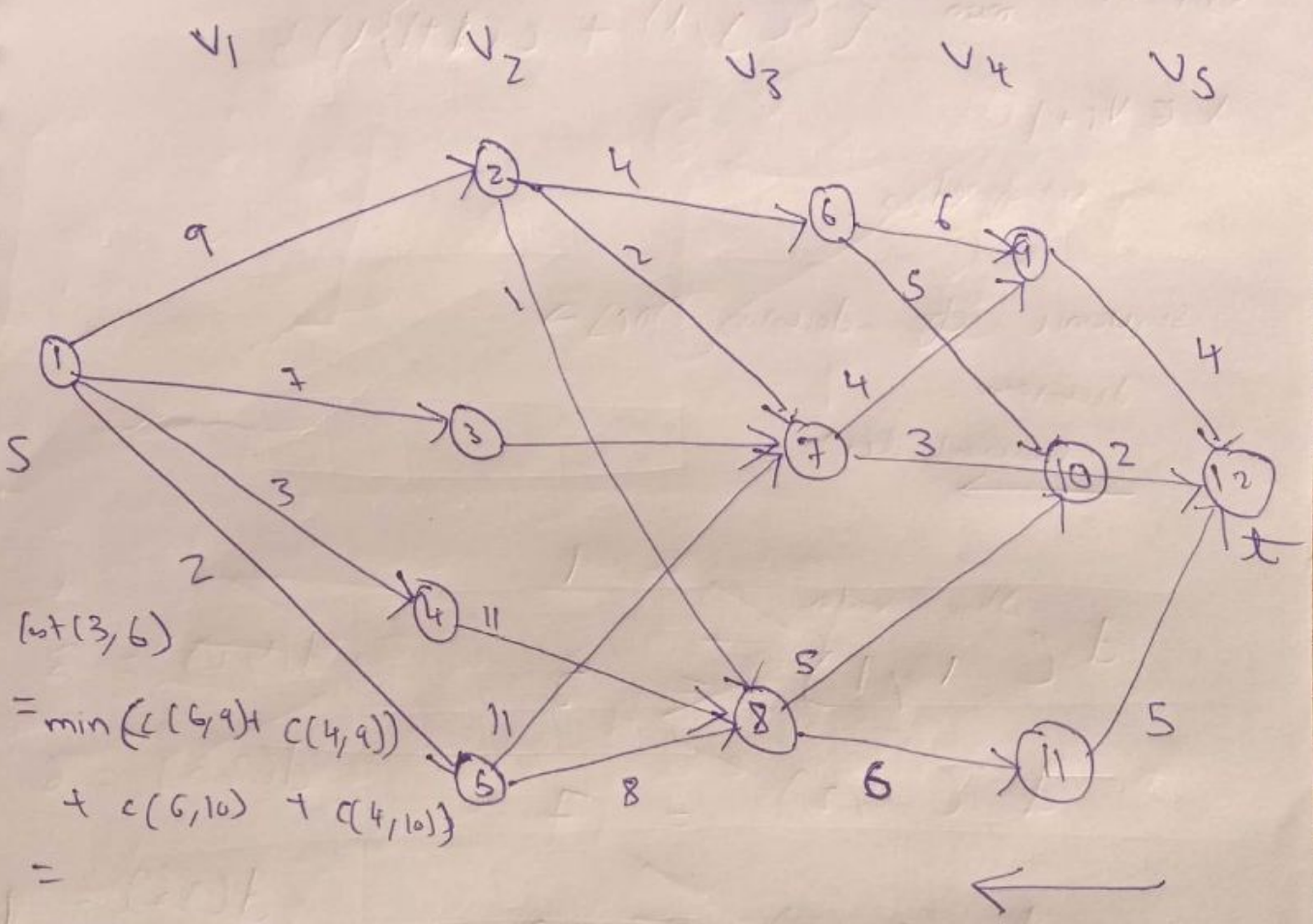
```
int f(int n)
{ if (n <= 1)
    return n;
  F[0] = 0; F[1] = 1;
  for (int i=2; i<=n; i++)
  { F[i] = f[i-2] + f[i-1];
  }
  return F[n];
}
```

In dynamic programming, we can devise our own idea to solve the problem.

In greedy Algorithm, we make whatever choice seems best at the moment in the hope that it will lead to global optimal solution. In Dynamic Programming, we make decision at each step considering current problem

and solution to previously solved sub problems to calculate optimal soln.

Multistage Graph



Cost(3,6)

= min {c(6,9) + c(4,9)
+ c(6,10) + c(4,10)}

=

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 or 3	7	6	8	8	10	10	10	12	12	12	12

2 or 3
2 3

According to DP, this problem should be solved in sequence of decisions

Stage value

Cost(5,12) = 0 Cost(4,10) = 2

Cost(4,9) = 4 Cost(4,11) = 5

Stage Vertex

$cost(i,j)$ =

$\min_{\langle j,i\rangle \in E,\ l \in V_{i+1}} \{c(j,l) + c(i+1,l)\}$

set of edges.

Sequence of decision (DP) → decisions.

forward dir^n →

Stage vertex

$d(1,1) = 2$

$d(2,2) = 7$

$d(3,7) = 10$

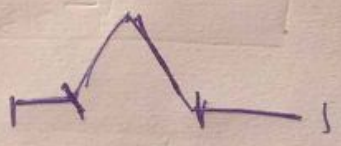
$d(4,10) = 12$

$d(1,1) = 3$

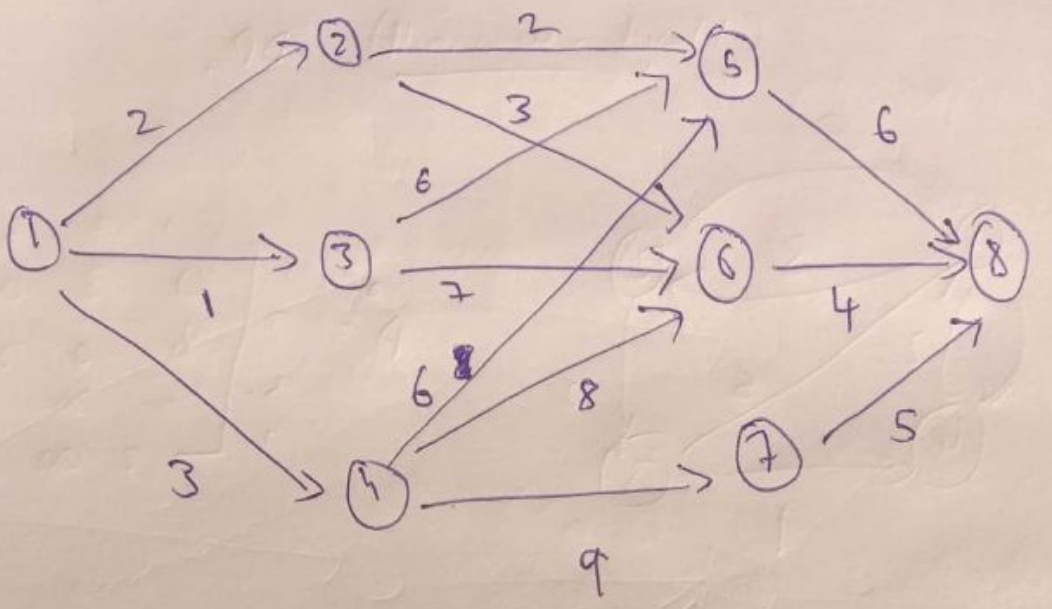
$d(2,3) = 6$

$d(3,6) = 10$

$d(4,10) = 12$



Multistage Graph (Program)



$O(n^2)$

C =

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	2	1	3	0	0	0	0
2									
3	0	6	0	0	0	6	7	0	0
4						6	8	9	0
5									
6									
7									
8									

1 to 8

Cost adjacency matrix

	0	1	2	3	4	5	6	7	8
Cost		9	~~7~~	11	12	6	4	5	0

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

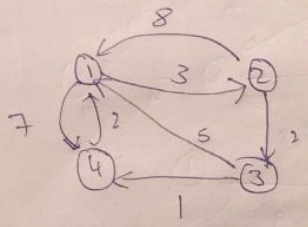
	0	1	2	3	4	5	6	7	8
Path		1	2	6	8				

p[3] = d[p[2]] = 6

p[i] = d[p[i-1]]

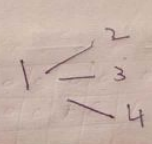
All Pairs Shortest Path

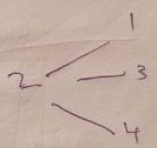
(Floyd - Warshall) DP.



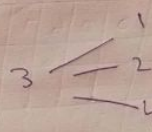
$A^0 =$

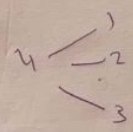
	1	2	3	4
1	0	3	∞	7
2	8	0	2	∞
3	5	∞	0	1
4	2	∞	∞	0





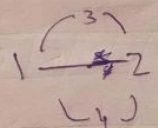
$n^2 + n = O(n^3)$





We can use dijkstra for solving each case.

Applying DP, Sequence of decisions.



$A^1 =$

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

IS

$A =$

0	3	∞	7
8	0		
5			
2			

$A^0[2,3] = A^0[2,1] + A^0[1,3]$

$2 < 8 \quad \infty$

$A^0[2,4] = A^0[2,1] + A^0[1,4]$

$\infty > 15$

$A^0[3,2] \quad A^0[3,1] + A^0[1,2]$

$\infty > 5 + 3$

$A^2 =$

0	3	5	7
8	0	2	7
5	8	0	1
2	5	7	0

$A^3 =$

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

$A^4 =$

0	3	5	6
5	0	2	3
3	6	0	1
2	5	7	0

$+ A^{k-1}[k,j])$

```
for (i=1 ; i<=n ; i++)
{ for (j=1 ; j<=n ; j++)
A[i,j] = min (A[i,j],
              A[i,k] + A[k,j]);
```

3 3 3

$O(n^3)$

Matrix Chain Multiplication (DP)

A_1	A_2	A_3	A_4
5×4	4×6	6×2	2×7

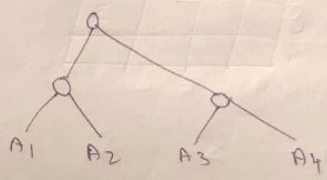
A × B = C

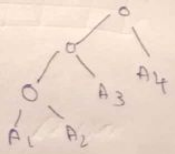
5×4 × 4×3 → 5×3 = 15

cost = 5×4×3 multiplications

$(((A_1 . A_2) . A_3) . A_4)$

$((A_1 . A_2) . (A_3 . A_4))$





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

$T(3) = 5$.

$A_3 A_4 = 4 \times 2 \times 2 = 84$

m

	1	2	3	4
1	0	120		
2		0	48	
3			0	84
4				0

S (k value)

	1	2	3	4
1		1		
2			2	
3				3
4				

m[1,1] → A_1

m[2,2] → A_2

m[1,2] → $A_1 . A_2$ (5×4 4×6)

m[2,3] → $A_2 A_3$ 48

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				

m[1,3]

$A_1(A_2A_3)$ $(A_1A_2)A_3$

5x4 4x6 6x2 5x4 4x6 6x2

m[1,1] + m[2,3]

+ 5x4x2

= 0 + 48 + 40

= 88

min

$(A_1A_2)A_3$

5x4 4x6 6x2

m[1,2]

+ m[3,3]

+ 5x6x2

= 120 + 0 + 60

= 180

m[2,4]

$A_2(A_3 \cdot A_4)$

4x6 6x2 2x7

m[2,2] + m[3,4] + 4x6x7

= 252

$(A_2 \cdot A_3) \cdot A_4$

4x6 6x2 2x7

m[2,3] + m[4,4]

+ 4x2x7

= 104

A_1	A_2	A_3	A_4
5×4	4×6	6×2	2×7

$m[1,4]$

$= \min \{$ $m[1,1] + m[2,4] + 5 \times 4 \times 7$,

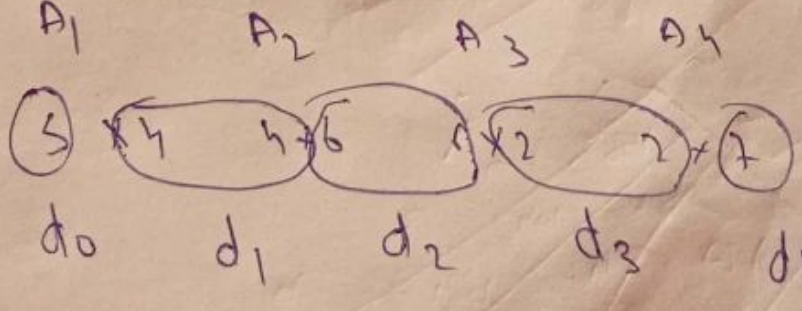
$m[1,2] + m[3,4] + 5 \times 6 \times 7$

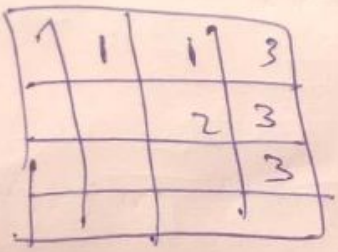
$+ m[1,3] + m[4,4] + 5 \times 2 \times 7$

$\}$

$= \{ 254, 414, 158 \}$ ✓ 3

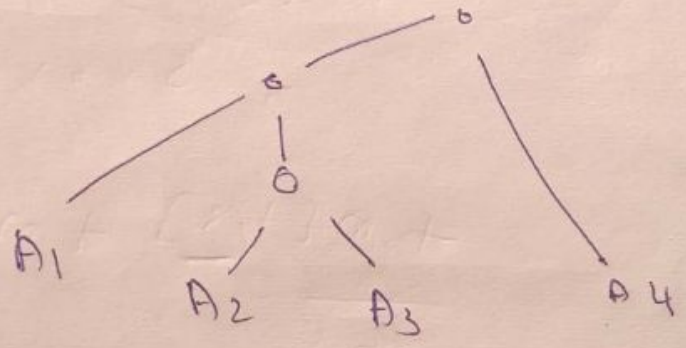
$m[i,j] = \min \{ m[i,k] + m[k+1,j] + d_{i-1} \times d_j \times d_k \}$





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

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

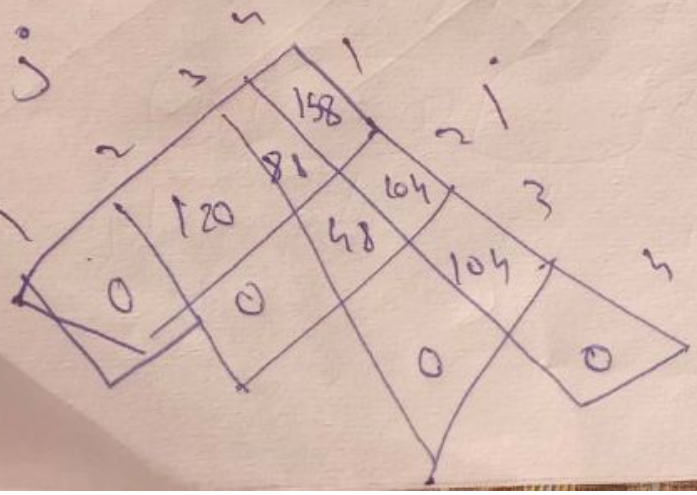


Time complexity $= \frac{n(n+1)}{2} \times n$

or $\frac{n(n-1)}{2} \times n$

~~$O(n^2)$~~

$O(n^3)$



Matrix Chain Multiplication

- Effort required for multiplying should be less

$A_1 \times A_2 \times \ldots \times A_n$

Best possible parenthesis to reduce cost of multiplication

$$c[i,j] = \min_{i \le k < j} \{ c[i,k] + c[k+1, j] + d_{i-1} * d_j * d_k \}$$

$A_1 \times A_2 \times A_3 \times A_4$

$d_0\ d_1 \quad d_1\ d_2 \quad d_2\ d_3 \quad d_3\ d_4$

$n = 4 - 1 = 3$

Catalog no.

$$\frac{2n C_n}{n+1} = 5$$

1. $A_1 (A_2 (A_3 A_4))$
2. $A_1 ((A_2 A_3) A_4)$
3. $(A_1 A_2)(A_3 A_4)$
4. $(A_1 (A_2 A_3)) A_4$
5. $((A_1 A_2) A_3) A_4$

$A_1(A_2 A_3 A_4)$
$(A_1 A_2)(A_3 A_4)$
$(A_1 A_2 A_3) A_4$

$$c[1,4] = \min_{1 < k \le 4} \begin{cases} k=1 & c[1,1] + c[2,4] + d_0 d_1 d_4 \\ k=2 & c[1,2] + c[3,4] + d_0 d_2 d_4 \\ k=3 & c[1,3] + c[4,4] + d_0 d_3 d_4 \end{cases}$$

$A_1 \times A_2 \times A_3 \times A_4$

3 2 2 4 4 2 2 5

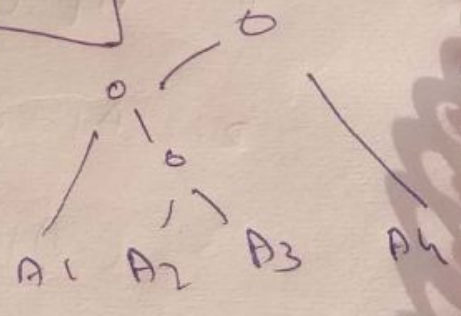
d_0 d_1 d_2 d_3 d_4

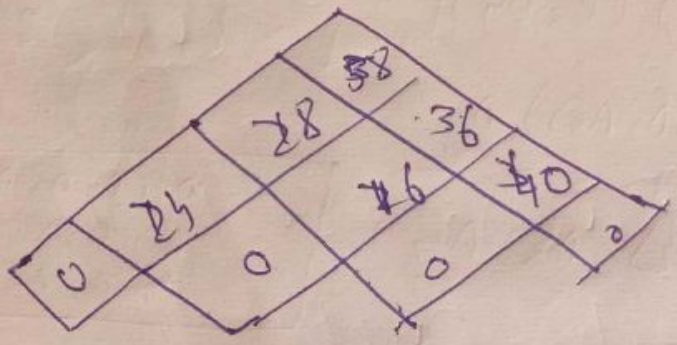
$$c[i,j] = \min_{1 \le k < 2} \{ c[1,1] + c[2,2] + d_0 d_1 d_2 \}$$

$$= 24$$

0	24	28	38
	0	16	36
		0	40
			0

	1	1	3
		2	3
			3





$((A_1 A_2)(A_3 A_4))$

$\frac{n(n+1)}{2} \times n \quad = O(n^3)$

Program →

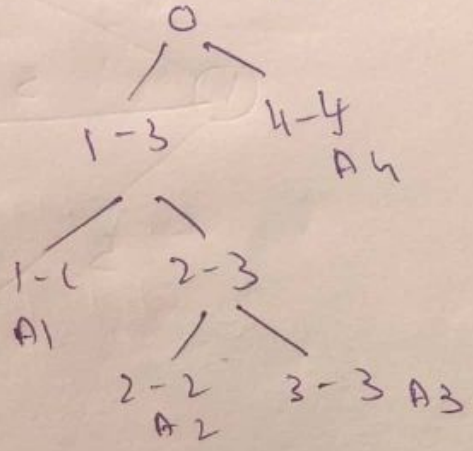
A_1	A_2	A_3	A_4
5 × 4	4 × 6	6 × 2	2 × 7

P

0	1	2	3
5	4	2	7

$(A_1.(A_2.A_3)).A_4$

Inorder traversal



c

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

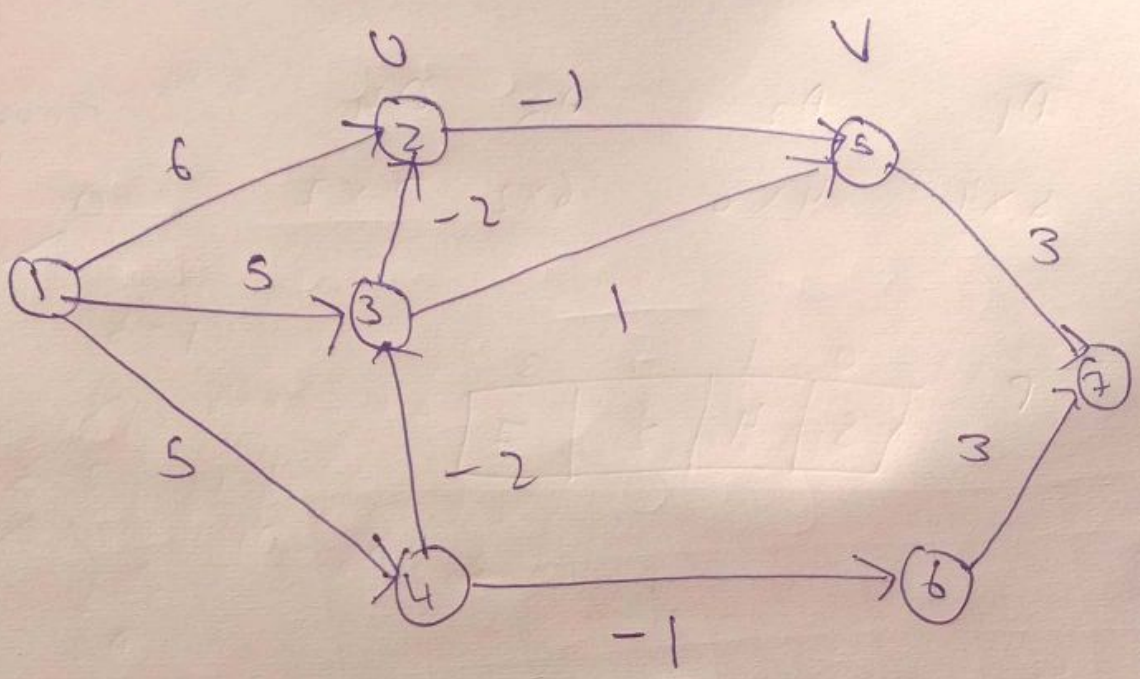
	0	1	2	3	4
0	0	0	0	0	0
1	0	0	1	1	3
2	0		0	2	3
3	0			0	3
4	0				0

i j = i+d

i	j	i	j	i	j
1	2	1	3	1	4
2	3	2	4		
3	4				

d=1 d=2 d=3

Bellman Ford → Single Source Shortest Path → Dynamic Programming



Dijkstra algorithm didn't work properly for -ve edges.

$|V| = n = 7$

$c(u, v)$

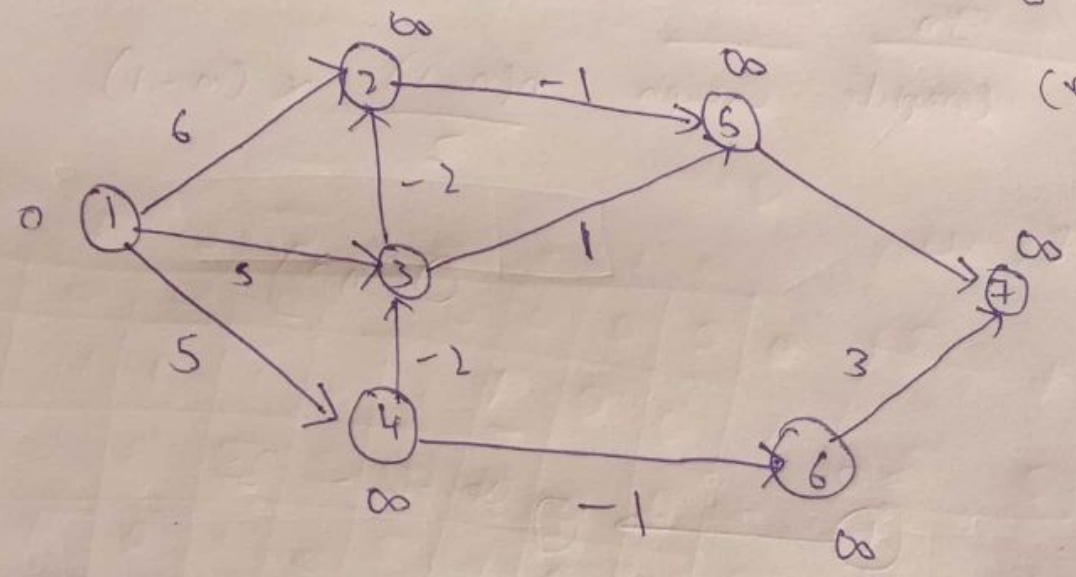
if $[d[v] + c(u,v) < d[v]]$

$d[v] = d[u] + c(u,v)$

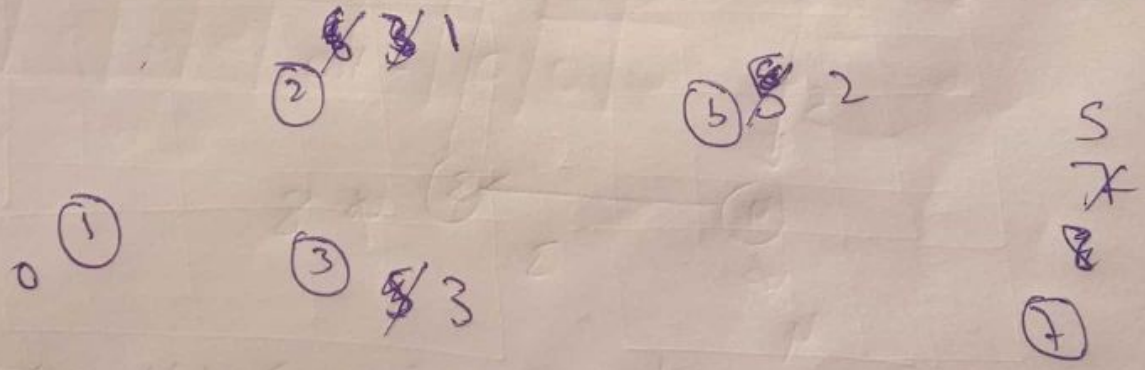
edgeList → (1,2), (1,3), (1,4), (2,5), (3,2), (3,5),
(4,3), (4,6), (5,7), (6,7)

Repeat 4 more times again (n-1)

Initially



⇒



Final Result

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

min time /

E Edges

|V|-1 $O(|E||V|)$

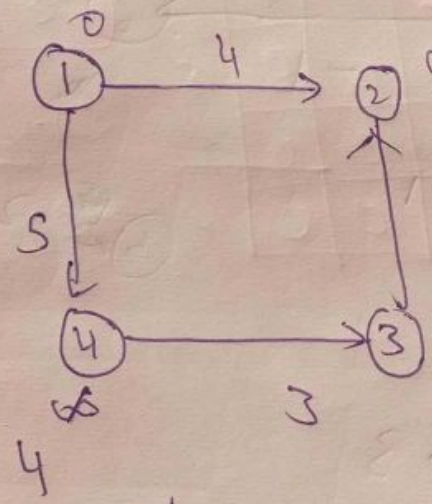
$O(n^2)$

In

Complete Graph $\frac{n(n-1)}{2} \times (n-1)$

$= O(n^3)$

Q.



$\cancel{4} - 2$

edges → (3,2) (4,3) (1,4), (1,2)

n = 4 - 1 = 3

1 → 0

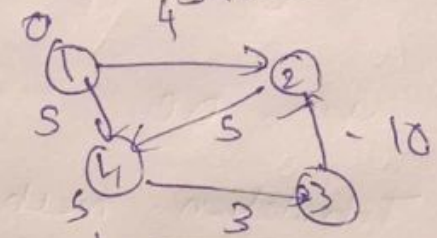
2 → -2

3 → 8

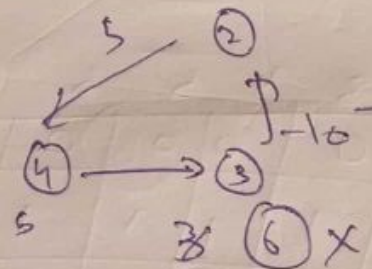
4 → 5 from source

Drawbacks

After completing $n-1$, there shouldn't be any relaxation.



But in case of cycles, changes may occur



-ve wt. cycle.

cannot be solved by Bellman fold

But it can be used to check -ve wt. cycle.

By performing 4th again after n-1 times

0/1 Knapsack Problem → DP

↓

we should try all possible sol^{n}, and pick the best one.

$m = 8$

$n = 4$

$P = \{1, 2, 5, 6\}$

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

$x_i = 0/1$

$x = \{1, 0, 0, 0\}$

Generally,

$O(2^n)$

0 0 0 0
1 1 1 1
0 0 0 1
X 1 0 0 0
1 1 0 0

Using DP

Tabulation method →

Pi Wi

1 2
2 3
5 4
6 5

V table

	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

Same

6 5 4+2

2(3+4 ≠10

Formula →

row column

$V[i, w]$

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

V [4,1]

= max { V[4-3, 1]

, V[3, 1-5] +6 }

= max { 0, V[3,-4]+6 }

× neglect

V[4, 5]

= max [V(3,5)

, V[3, 0] + 6]

= max [5, 6]

V[4,6] = max { V(3,6]

, V[3,1] +6 }

= max (6 , 0, 6)

= 6

V(4,8) = max (7,8) = 8.

x_1 x_2 x_3 x_4

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

2 object.

$8 - 6 = 2$

$2 - 2 = 0$

absent

(2)

8 absent

(8)

Q.

<u>Sets</u> <u>Method</u>

$m = 8$

$n = 4$

$P = \{ 1, 2, 5, 6 \}$

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

$S^0 = \{ (0,0) \}$ No profit No weight

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

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

$+ (2,3) . \quad + (2,3)$

$S_1^1 = \{(2,3), (3,5)\}$ ✓

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

$S^2_1 = \{(5,4), (6,6), (7,7),$ $(8,9)\}$ Profit & weight $9 > 8$ but capacity

$S^3 = \{(0,0), (1,2), (2,3), (3,5),$ ~~$(5,4)$~~ Dominance rule
P W
$(3,5)$ P 3 to 5 $(5,4), (6,6), (7,7),$ ~~$(8,9)$~~
$(5,4)$ W 5 to 4 X
discard smaller one
P ↑ W ↓ X

$S_1^3 = \{(6,5), (7,7), (8,8),$ ~~$(4,9)$~~, ~~$(12,11)$~~, ~~$(13,12)$~~$\}$ $+(4,5)$

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

$O(2^n)$

$(8,8) \in S^4$ $x_4 = 1$

$(8,8) \notin S^3, S^2, S^1$ $x_3 = 0$

$\Rightarrow (8-6, 8-5) = (2,3) \in S^3$ $x_2 = 1$

① $(2,3) \in S^2$ $\quad 2,3 - 2,3$

$= 0,0$

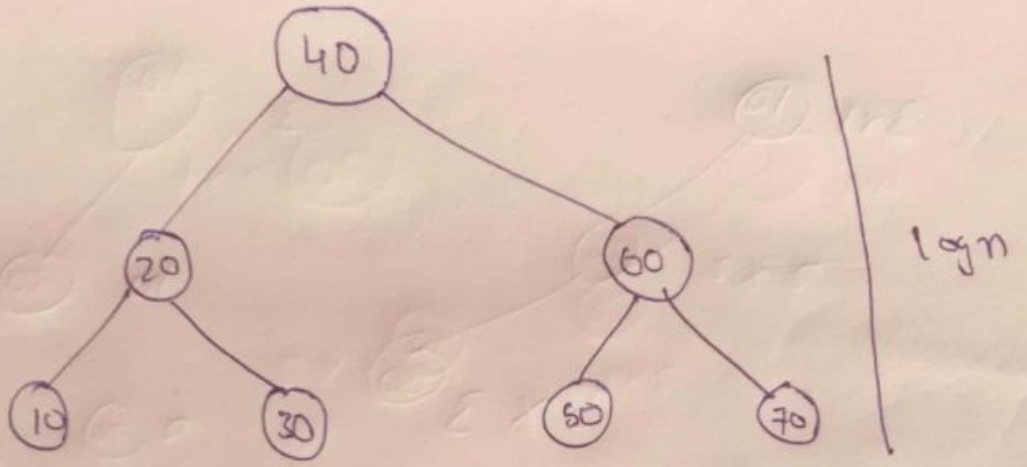
but $\notin S^1$. $x_2 = 1$

$(0,0) \in S^1$

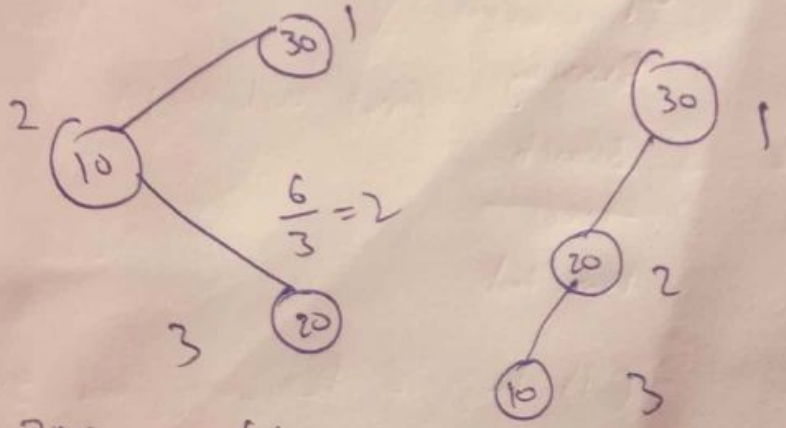
$(0,0) \in S^0$ $\quad x_1 = 0$ $\quad \{0,1,0,1\}$

(DP) Optimal Binary Search Tree →

Keys → 10, 20, 30, 40, 50, 60, 70



keys → 10, 20, 30

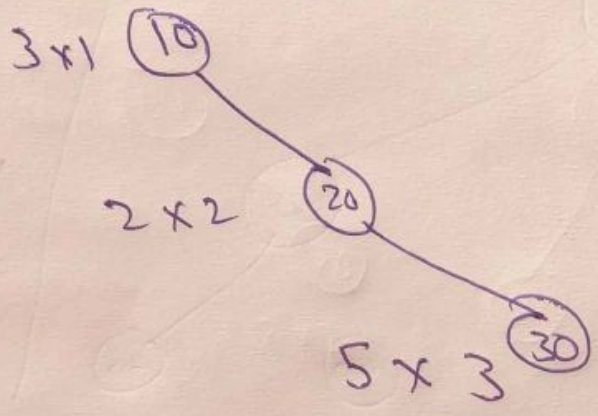


$\frac{2n C_n}{n+1} = \frac{6 C_3}{4} = 5$ trees.

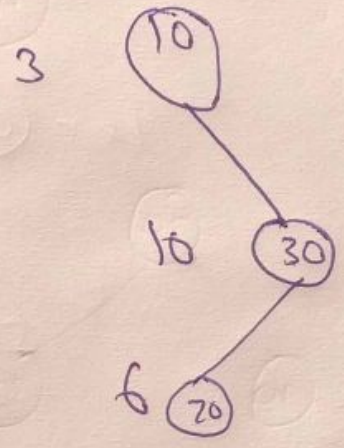
Optimal BST

keys → 10, 20, 30

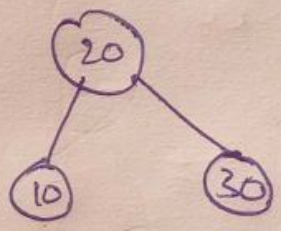
frequency → 3 2 5



22

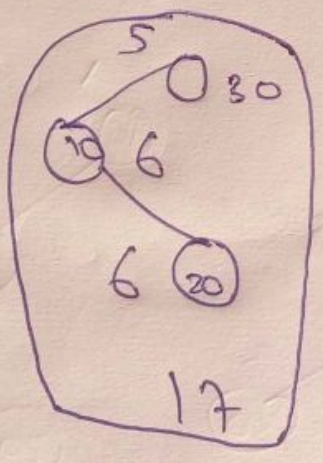


19



18

height balanced



optimal Binary Search Tree has cost

18

(not balanced)

(if keys & frequency are given)

Q.

	1	2	3	4
keys	10	20	30	40
frequency	4	2	6	3

i \ j	0	1	2	3	4
0	0	4	8^1	20^3	26^3
1		0	2	10^3	16^3
2			0	6	12^3
3				0	3
4					0

$d = j - i = 0$

$0-0=0$

$1-1=0$

$2-2=0$

$3-3=0$

$4-4=0$

$d = j - i = 1$

$1-0=1 \quad (0,1)$

$2-1=1 \quad (1,2)$

$3-2=1 \quad (2,3)$

$4-3=1 \quad (3,4)$

$d = j - i = 2$

$2-0=(0,2)$

$3-1=(1,3)$

$4-2=(2,4)$

$C[0,1]$ $C[1,2]$ $C[2,3]$ $C[3,4]$

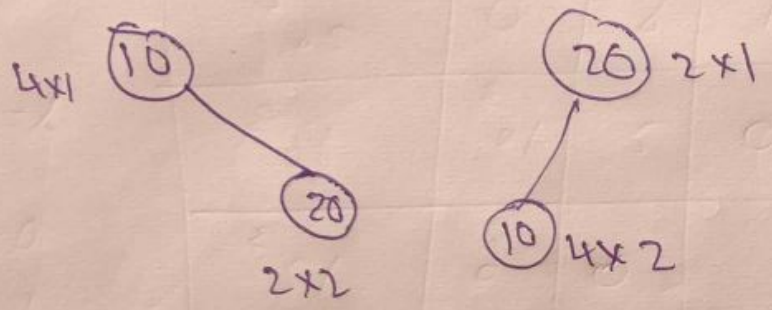
10 20 30 40

4 2 6 3

C[0,2] 2 keys

10 20

4 2



8 10

min

w(0,4)

$= \sum_{i=1}^{4} f(i)$

C[0,2]

= C[0,1] + C[2,2] + w[0,2]

C[0,0] + C[1,2] + w[0,2] = 10

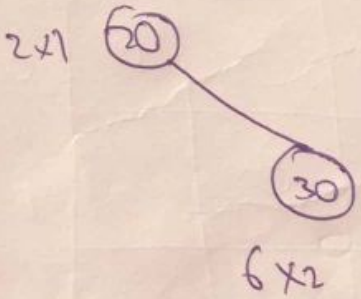
= 0 + 2 + (4+2)

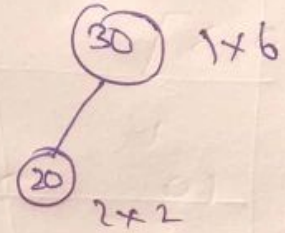
= 8.

C[1,3]

20 30

2 6





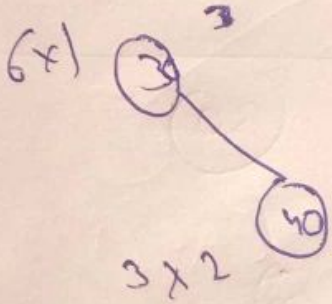
= 14 = 10

C[2,4]

3 4

30 40

6 3



12 15

12

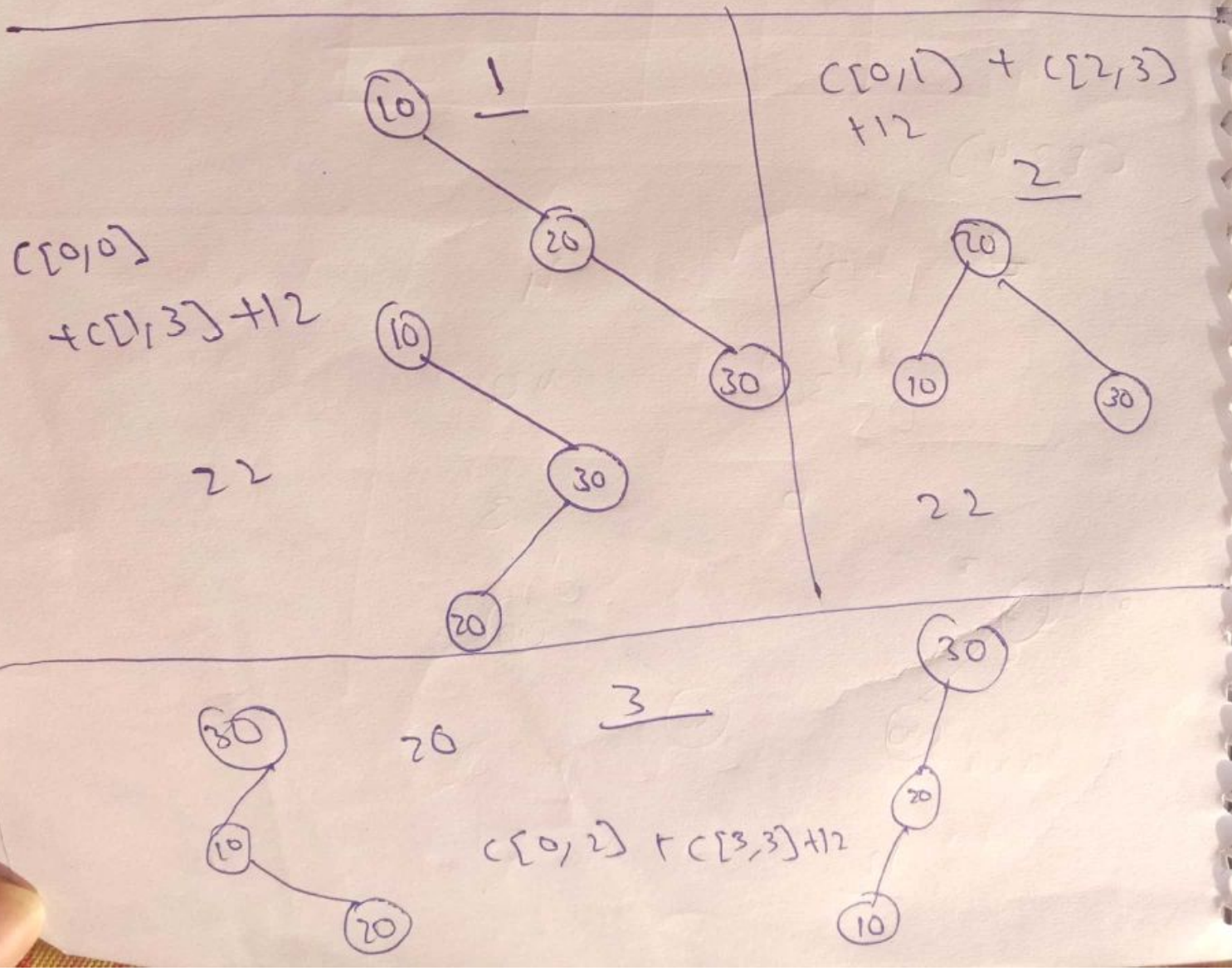
$\lambda = j - i = 3$

(0, 3)

(1, 4)

C[0,3]

1	2	3
10	20	30
4	2	6



c[0,3] w[0,3] = 12

~~c[0,0]~~

c[1,4]

2	3	4
20	30	40
2	6	3

$$c[1,4] = \min = \left\{ \begin{array}{l} c[1,1] + c[2,4] \\ c[i,2] + c[3,4] \\ c[1,3] + c[3,4] \end{array} \right\}$$

$$+ 11$$

$$= \left\{ \begin{array}{l} 12 \\ 5 \\ 10 \end{array} \right\} + 11$$

$$= 16.$$

C [0,4)

1	2	3	4
10	20	30	40
4	2	6	3

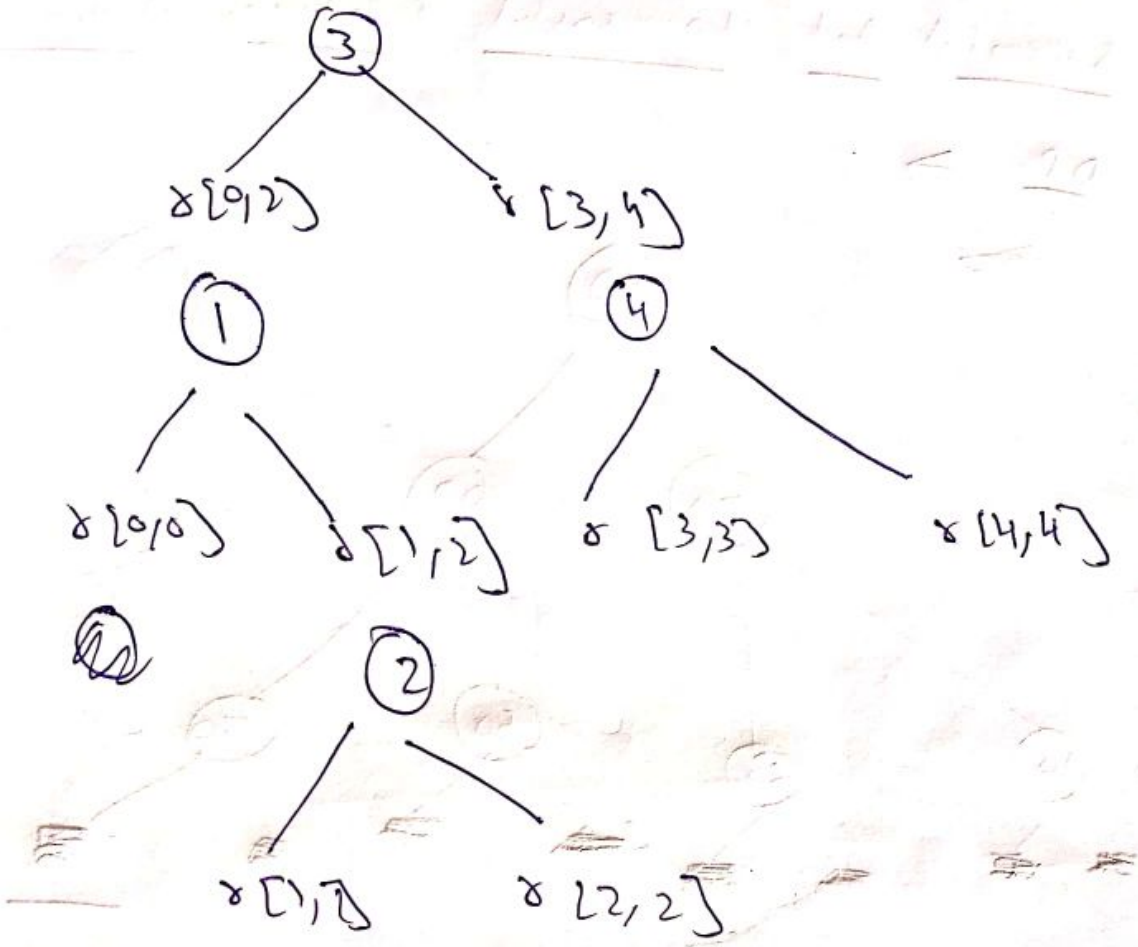
w[0,4) = 15.

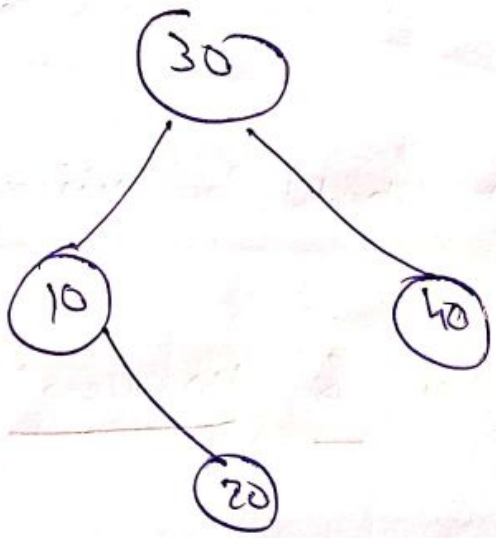
$$C[0,4] = \min \left\{ \begin{array}{l} c[0,0] + c[1,4] \\ c[0,1] + c[2,4] \\ c[0,2] + c[3,4] \\ c[0,3] + c[3,4] \end{array} \right\} + 15$$

$$= \min \left\{ \begin{array}{l} 16, \\ 12, \\ 17, \\ 20 \end{array} \right\} + 15$$

= 26.

d[0,4] = 3



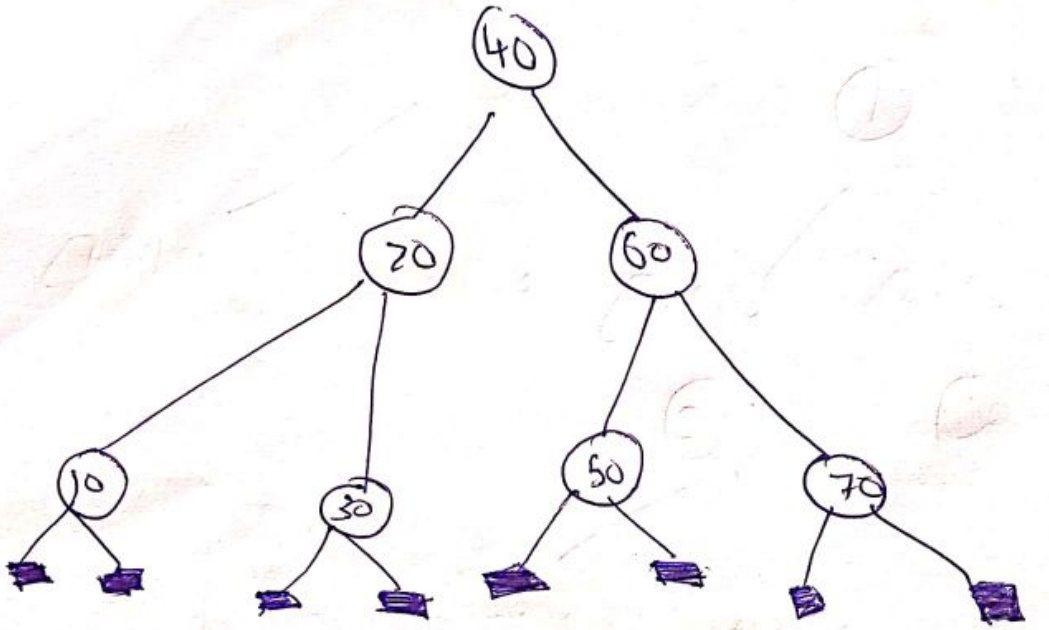


$$c[i,j]$$

$$= \min_{j<k\le j} \{ c[i,k-1] + c[k,j] \} + w(i,j)$$

Successful And Unsuccessful Probability - Optimal BST

DP →

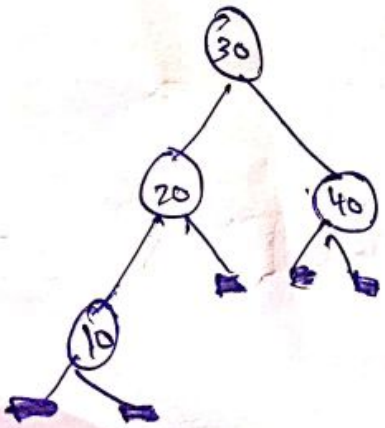


unsuccessful search nodes

Hight should be minimum

Successful & unsuccessful search Probability →

4 keys

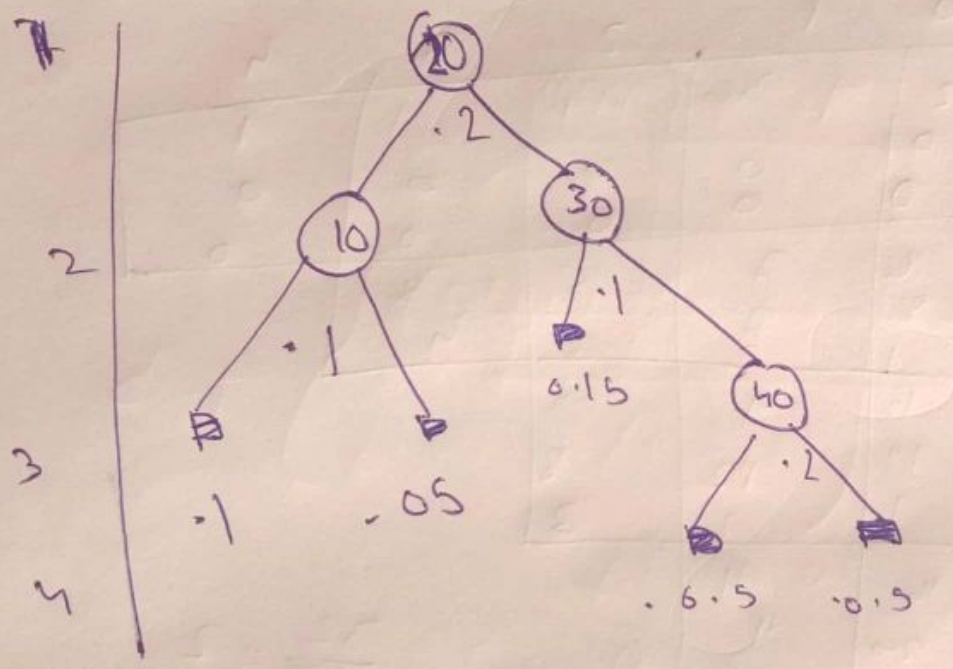


keys : (10) (20) (30) (40)

	p_1	p_2	p_3	p_4
p_i	1	2	1	2
q_i :	0.05	15	0.5	0.5

q_0 q_1 q_2 q_3 q_4 (q_4, ∞)

$(-\infty, q_0]$



$$C[0,n] = 1\times0.2 + 2\times.1 + 2\times.1 + 3\times.2 + 2\times.1 + 2\times0.5 + 2\times0.15 + 3\times0.05 + 3\times0.5$$

$$= \sum_{j \le i \le n} p_i * level(a_j)$$

$$\tau \le \sum_{0 \le i \le n} * (level(a_i - 1))$$

Q. keys = { 10 20 30 40 } (0 1 2 3 4)

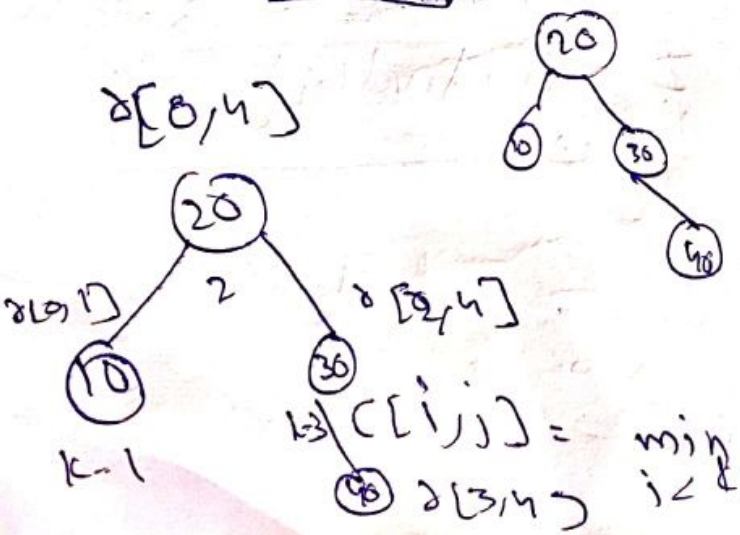
Successful search Prob P_i = { 3 3 1 1 }

Unsuccessful search Prob q_i = { 2 3 3 1 1 }

	j→ 0	1	2	3	4
j-i=0	w00 = 2, c00 = 0, r00 = 0	3, 0, 0	1, 0, 0	1, 0, 0	1, 0, 0
j-i=1	w01 = 8, c01 = 8, r01 = 1	7, 7, 2	3, 3, 3	3, 3, 4	
j-i=2	12, 19, 1	9, 12, 2	5, 8, 3		
j-i=3	14, 20, 2	11, 9, 2			
j-i=4	16, 32, 2				

K

$\frac{32}{16} = 2$



$w[i,j] = w[i,j-1] + p_j + q_j$

$c[i,j] = \min_{i<k\le j} \{c[i,k-1] + c[k,j]\} + w[i,j]$

Q.

key →	10	20	30
Frequency →	3	2	5

i \ j	0	1	2	3
0	0	3		
1		0	2	
2			0	5
3				0

for $l = 0$,

Required combinations are $(0,0)$, $(1,1)$, $(2,2)$, $(3,3)$.

$c = 0$.

for $l = 1$,

Desired combinations of i and j are $(0,1)$, $(1,2)$, $(2,3)$.

$c[0,1] = 3$

$c[1,2] = 2$

$c[2,3] = 5$

$(j-i)^2$,

Desired combinations if i and j are (0,2) and (1,3)

$C[0,2]$

$$= \min_{0<k\leq 2} \{ c[0,k-1] + c[k,2] \} + w(0,2)$$

$$= \min \begin{Bmatrix} c[0,0] + c[1,2] \\ c[0,1] + c[2,2] \end{Bmatrix} + 5$$

$$= \min \{2+0, 3\} + 5$$

$C[0,2] = 5+2 = 7$.

$$C[1,3] = \min_{1<k\leq 3} \{ c[1,k-1] + c[k,3] \}$$

$$= \min \begin{Bmatrix} c[1,1] + c[2,3] \\ c[1,2] + c[3,3] \end{Bmatrix} + 7$$

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

$C[1,3] = 9$.

i \ j	0	1	2	3
0	0	3	7	
1		0	2	9
2			0	5
3				0

for $l = 3$,

(0,3) is the required combination

$$C[0,3] = \min_{0<k\leq 3} \{ c[0,k-1] + c[k,3] \} + w(0,3)$$

$$= \min \begin{Bmatrix} c[0,0] + c[1,3] \\ c[0,1] + c[2,3] \\ c[0,2] + c[3,3] \end{Bmatrix} + 12$$

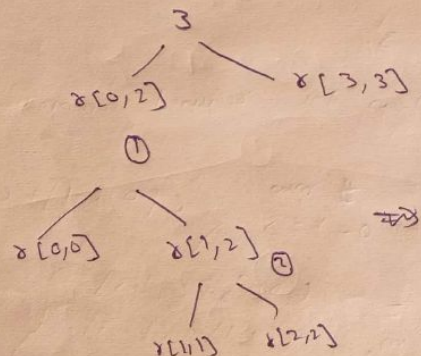
$= \min \{ 9, 8, 7 \} + 12$

$= 7 + 12$

$= 19.$

i \ j	0	1	2	3
0	0	3	7	19
1		0	2	9
2			0	5
3				0

$r[0,3]$



=>

Optimal BST

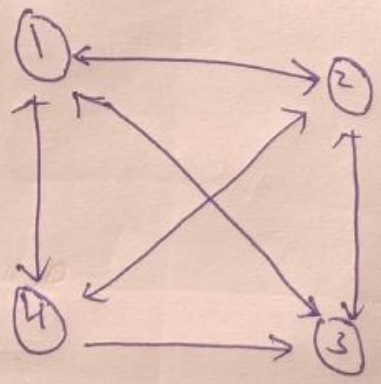
(cost = 5*1 + 3*2 + 2*3 = 17).

Traveling Salesman Problem →

(DP)

$$g(i,S) = \min_{k \in S} \{ C_{ik} + g(k, S-\{k\}) \}$$

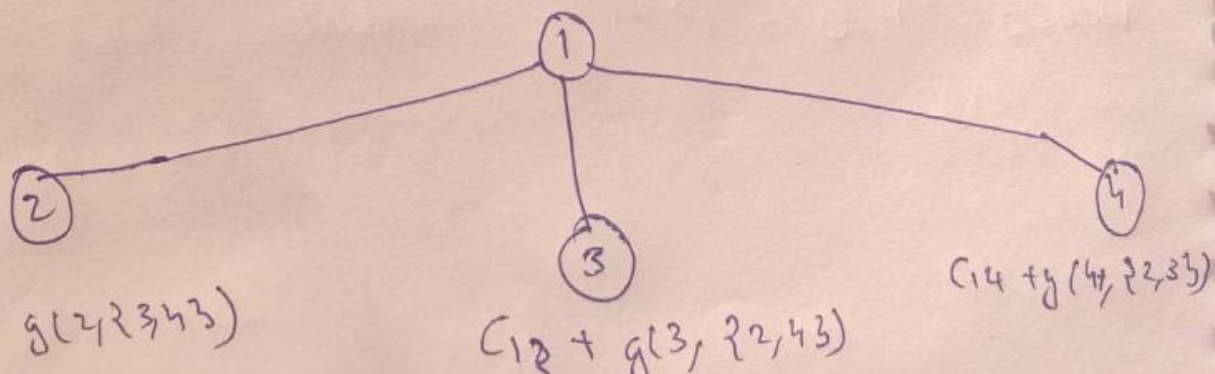
	1	2	3	4
1	0	10	15	20
2	5	0	9	0
3	6	13	0	12
4	8	8	9	0

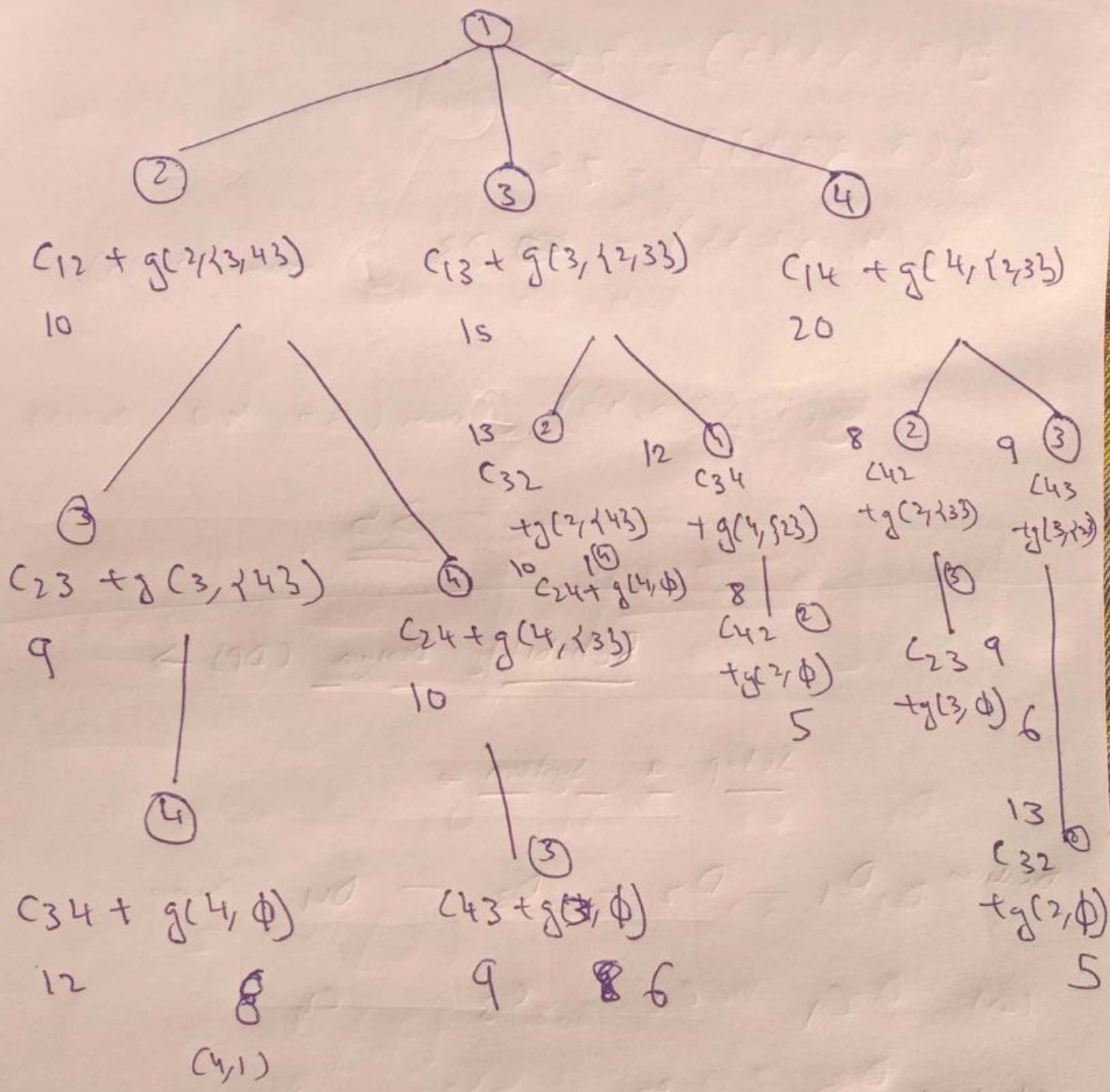


Traveling Salaseman problem is used to find the minimum cost to travel from a starting point and return back to it covering all other points in it's path.

→ starting vertex → remaining one's

$$g(1, \{2,3,4\}) = \min_{k \in \{2,3,4\}} \{ C_{1k} + g(k, \{2,3,4\}-\{k\}) \}$$





Dynamic Programming

dynamically generated table

$g(2,\phi) = 5$

$g(3,\phi) = 6$

$g(4,\phi) = 8$

$g(2,\{3\}) = 15$

$g(2,\{4\}) = 18$

$g(3,\{2\}) = 18$

$g(3,\{4\}) = 20$

$g(4,\{2\}) = 13$

$g(4,\{3\}) = 15$

$g(2, \{3,4\}) = 25$

$g(3, \{2,4\}) = 25$

$g(4, \{2,3\}) = 23$ min (23, 25).

$g(1, \{2,3,4\}) = \min(35, 40, 26+23)$

$= \underline{35}$.

Reliability Design (DP) →

Setup a System →

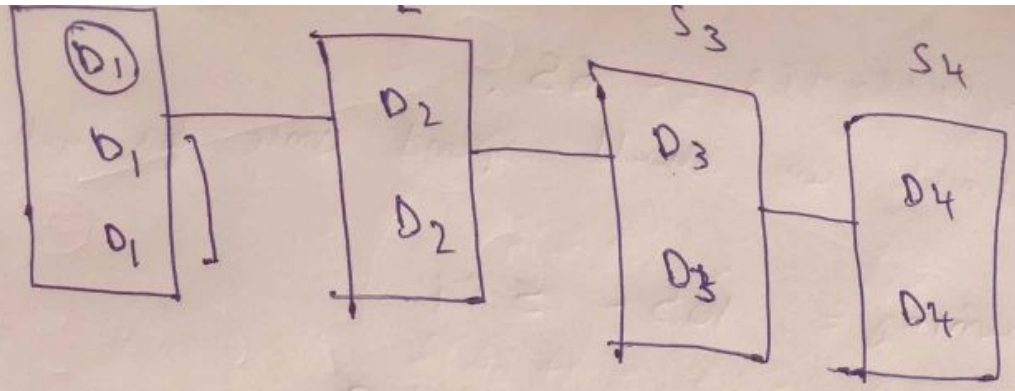
Devices $D_1 - D_2 - D_3 - D_4$ Series

Cost C_1 C_2 C_3 C_4

Reliability r_1 r_2 r_3 r_4

Total reliability $= \prod r_i = 0.9^4 = 0.6561$

(65% chances that device works fine)



$r_1 = 0.9$ (working)

$1 - r_1 = 0.1$ (not working)

$(1-r_1)^3 = 0.001$

$1 - (1-r_1)^3 = 0.999$.

It means that if we have multiple copies of same device, system reliability increases.

* How many devices which we should buy within the cost such that the system reliability is maximum.

Q.

D_i	C_i	r_i	v_i
D_1	30	0.9	
D_2	15	0.8	
D3	20	0.5	

$C = 105$

(definitely we should spend once for each device)

Remaining = $105 - \sum c_i$

$= 105 - 65 = 40$

$\frac{40}{30} = 1$

$$v_i = \left\lfloor \frac{C - \sum c_i}{c_i} \right\rfloor + 1 = \frac{40}{30} + 1 = 1 + 1$$

D_i	c_i	γ_i	v_i (max no. of copies)
D_1	30	0.9	$\lfloor \frac{40}{30} \rfloor + 1 = 2$
D_2	15	0.8	$\lfloor \frac{40}{15} \rfloor + 1 = 3$
D_3	20	0.5	$\lfloor \frac{40}{20} \rfloor + 1 = 3$

(R, C)

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

D_1

$S_1^1 = \{(0.9, 30)\}$

$S_2^1 = \{(0.99, 60)\}$

2 copies

$1 - (1-\gamma)^2 = 1 - (1-0.9)^2$

$= 1 - 0.01$

$= 0.99$

$S^1 = \{(0.9, 30), (0.99, 60)\}$

Consider D_2, (3 copies)

$S_1^2 = \{(0.9 \times 0.8, 45), (0.792, 75)\}$

$S_2^2 = \{(0.864, 60), (0.9504, 90)\}$ we can't purchase 3rd device

2 copies

$1 - (1-\gamma_2)^2 = 1 - (1-0.8)^2 = 0.96$

$S_3^2 = \{(0.8928, 75), (\nearrow, 105)\}$

$1 - (1-0.8)^3 = 0.992$ 45

$S^2 = \{(0.7245), (0.792, 75), (0.864$

$0.8928, 75)\}$

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

D_1

$S_1^1 = \{(0.9, 30)\}$

$S_2^1 = \{(0.99, 60)\}$

$S^1 = \{(0.9, 30), (0.99, 60)\}$

D_2

$S_1^2 = \{(0.72, 45), (0.792, 75)\}$

$S_2^2 = \{(0.864, 60), \cancel{(\ , 90)}\}$

$S_3^2 = \{(0.8928, 75), \cancel{(\ , 105)}\}$

$S^2 = \{(0.72, 45), \cancel{(0.792, 75)}, (0.864, 60), (0.8928, 75)\}$ Dominance Rule

D_3

$S_1^3 = \{(0.36, 65), (0.432, 80), (0.4464, 95)\}$

$(1-(1-0.5)^2) = 0.75$

$S_{\frac{2}{*}}^3 = \{(0.54, 85), (0.648, 100), \cancel{(\ , 115)}\}$

$D=2$

1 - 0.125

$S_3^3 = \{(0.63, 105), (\cancel{,120}), (\cancel{,135})\}$

$= \{(0.63, 105)\}$.

$S^3 = \{(0.\cancel{88}, 65), (0.432, 80),$

$(0.\cancel{4464}, 95), (0.54, 85),$

$(0.\cancel{63}, \cancel{105}), (0.648, 100)\}$

T_{8j} 0.648 $\{C_F$ 100 ..

No. of copies.

$D_3 = 2$

$D_2 = 2$

$D_1 = 1$

Longest Common Subsequence (LCS)

Memoization, Recursion and DP →

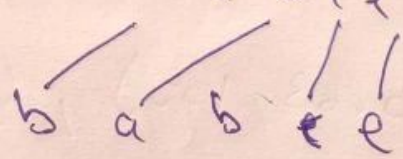
String 1 : a b c d e f g h i j

String 2 : c d g i

Q.

String 1 : a b d a c e

String 2 : b a b c e



LCS. [b a c e → 4]
[a b c e]

a b d a c e

b a b c e

Algorithm →

A | b | d | \0 |
0 1 2

B | a | b | c | d | \0 |
0 1 2 3 4

int LCS(i,j)

{ if (A[i] == '\0' || B[j] == '\0')

return 0;

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

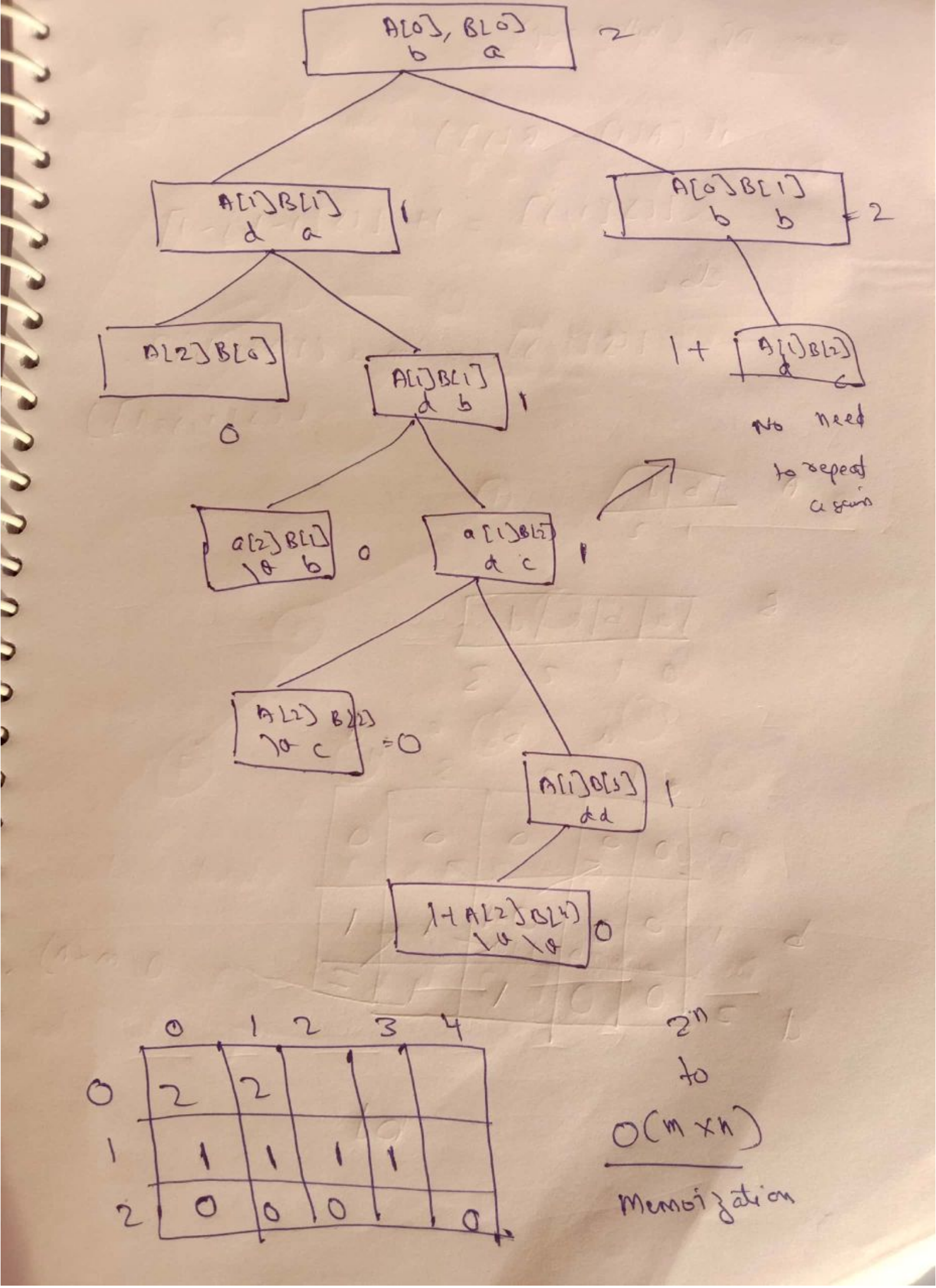
return 1 + LCS(i+1, j+1);

else

return max(LCS(i+1, j), LCS(i, j+1));

top down approach

⇒ Z



using DP, (bottom-up)

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

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

else

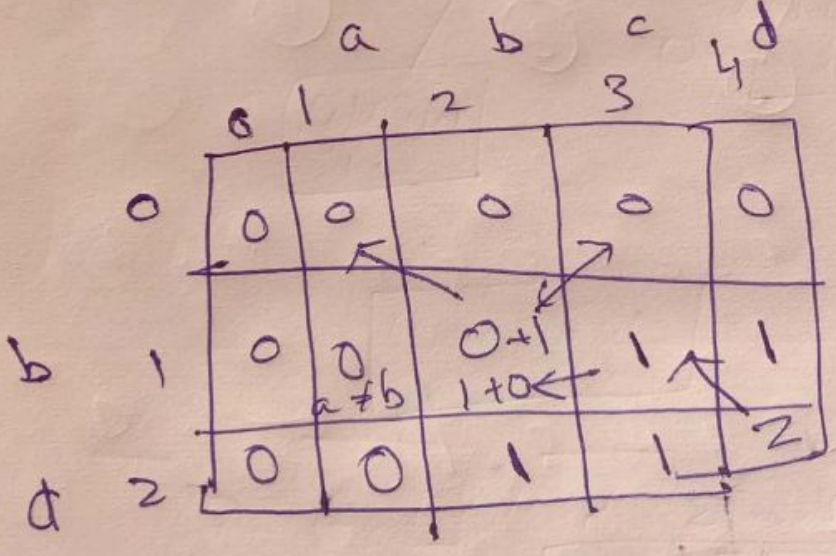
LCS[i,j] = max (LCS[i-1, j], LCS[i, j-1])

A

b	d
1	2

B

a	b	c	d
0	1	2	3



O(m*n).

bd

Q. str1 : Stone

str2 : longest

		l	o	n	g	e	s	t
	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
S 1	0	0	0	0	0	0	1	1
t 2	0	0	0	0	0	0	1	2
o 3	0	0	1	1	1	1	1	2
n 4	0	0	1	2	2	2	2	2
e 5	0	0	1	2	2	3	3	3

prev column

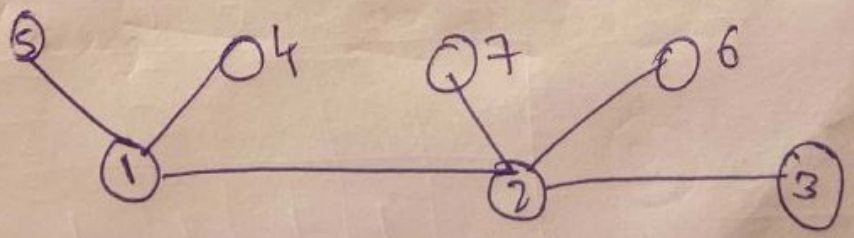
prev row

O(m x n)

0 ne

BFS and DFS →

Graph Traversals →

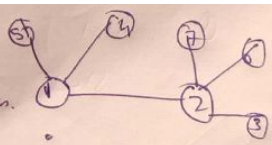


① Visiting a vertex

② Exploring a vertex (analyzing vertex neighbours)

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

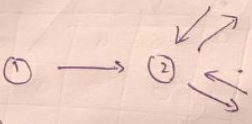
explore adjacent vertices in any order.

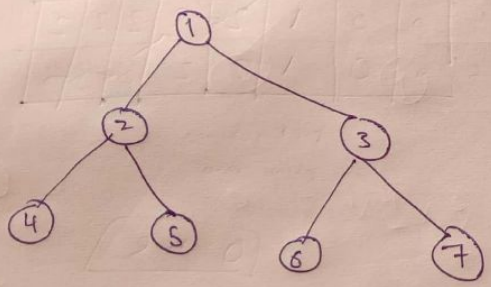


DFS : 1, 2, 3, 6, 7, 5, 4

Explore one by one.

Explore a vertex then goto another one





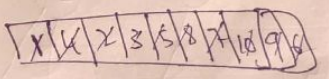
BFS → 1 2 3 4 5 6 7 level order

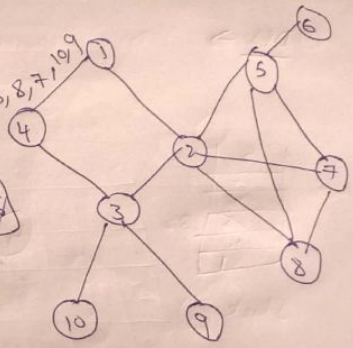
DFS → 1 2 4 5 3 6 7 preorder order.

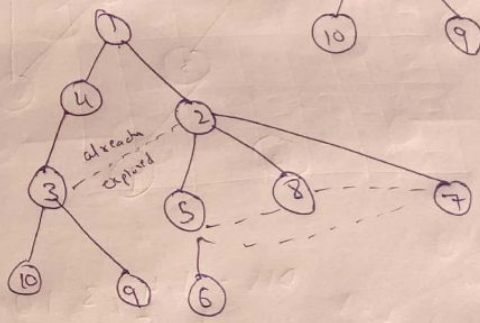
Q.

BFS → 1, 4, 2, 3, 5, 8, 7, 10, 9

Queue →







BFS Spanning Tree

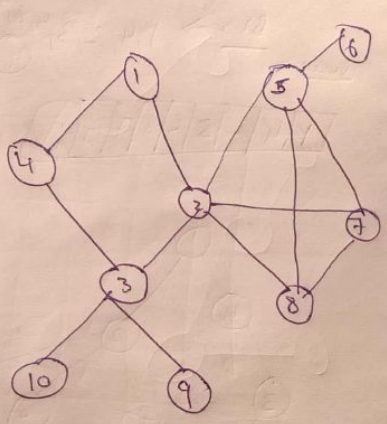
1) 1, 2, 4, 8, 5, 7, 3, 6, 10, 9

2) 5, 2, 8, 7, 6, 3, 1, 10, 9, 4

etc.

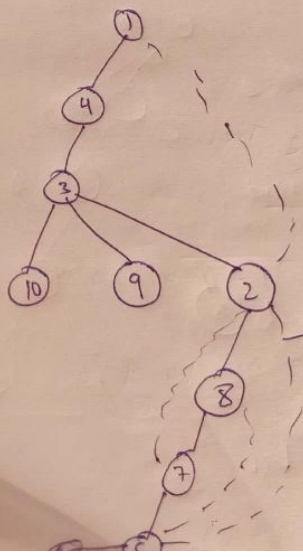
DFS →

Stack



DFS : 1, 4, 3, 10, 9, 2, 8, 7

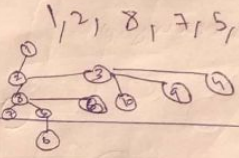
Once, stack a new vertex start exploring it



back Edges

DFS spanning tree

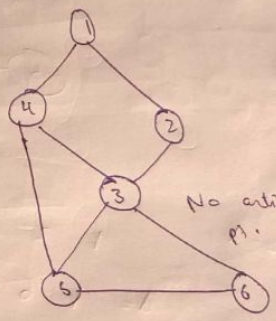
1)



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

Articulation Point

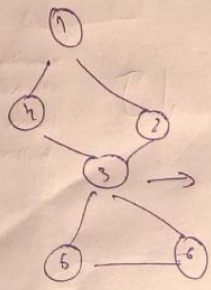
Biconnected components



While designing a N/w we don't need a articulation pt.

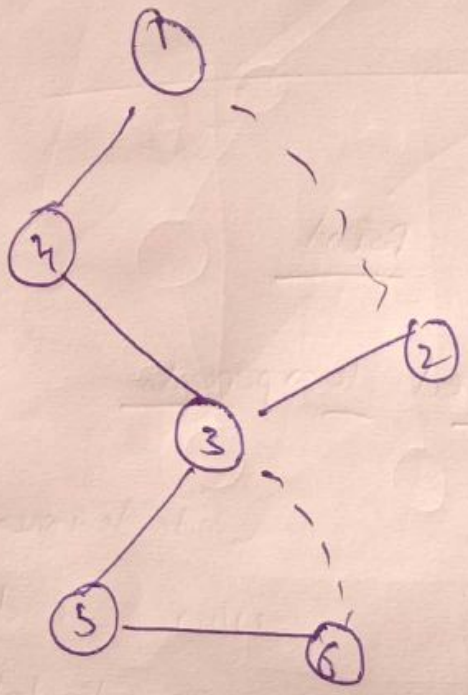
We prefer articulation

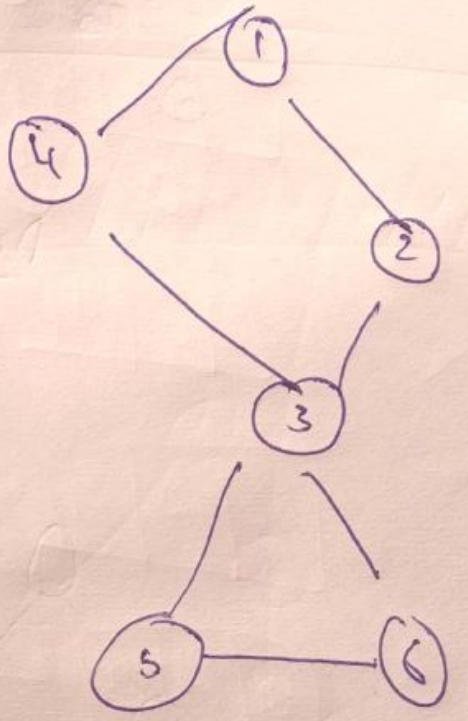
No articulation pt. pt. should'nt be first in graph

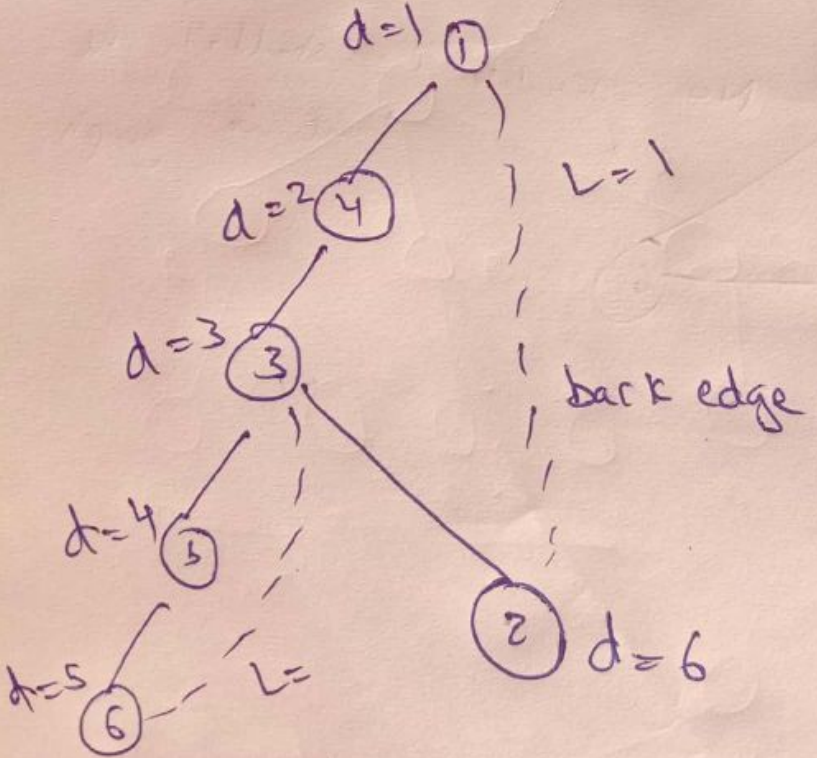


→ articulation pt.

① Prepare DFS Spanning pl.







	1	2	3	4	5	6
d=	1	6	3	2	4	5
L=	1	1	1	1	3	3

Lowest number reachable from bottom

parent child
u, v

parent child

$L[v] \geq d[v]$ u, v

u is articulation pt except root

3 4

1 2 parent u child v

$L[5]$ $d[3]$

3 ≥ 3

If any articulation pt is found, we add a edge to make graph 3 connected (3) articulation pt.

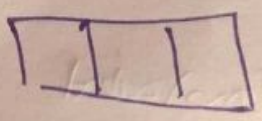
---

not for optimization ← Back Tracking

Brute force Approach → says that for any desired problem try out all possible soln & pick up desired one

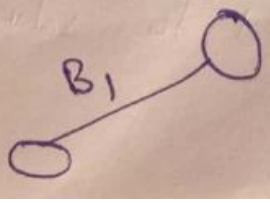
Back tracking is used when we have multiple solutions and we want all of those.

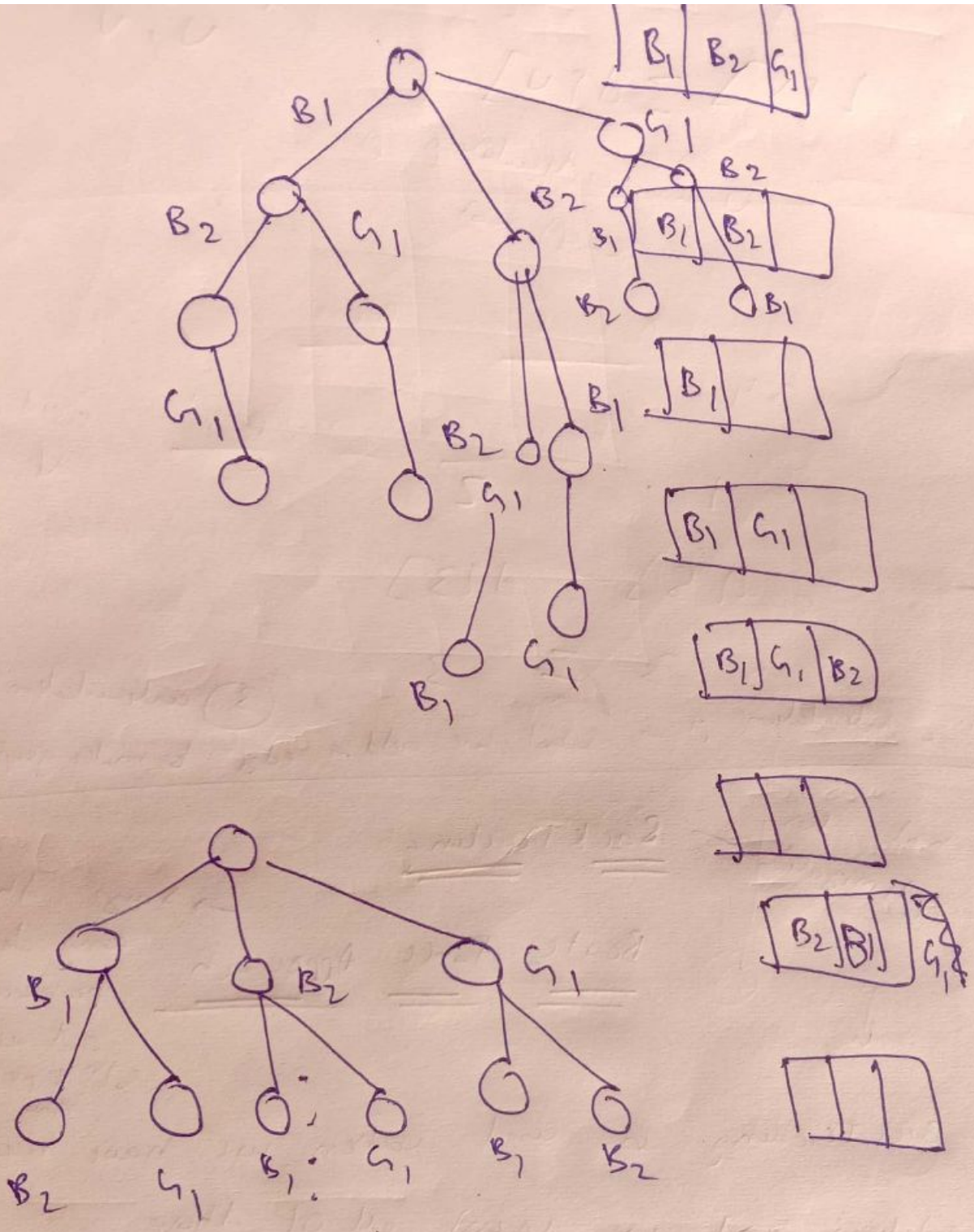
B_1 B_2 G_1



3!

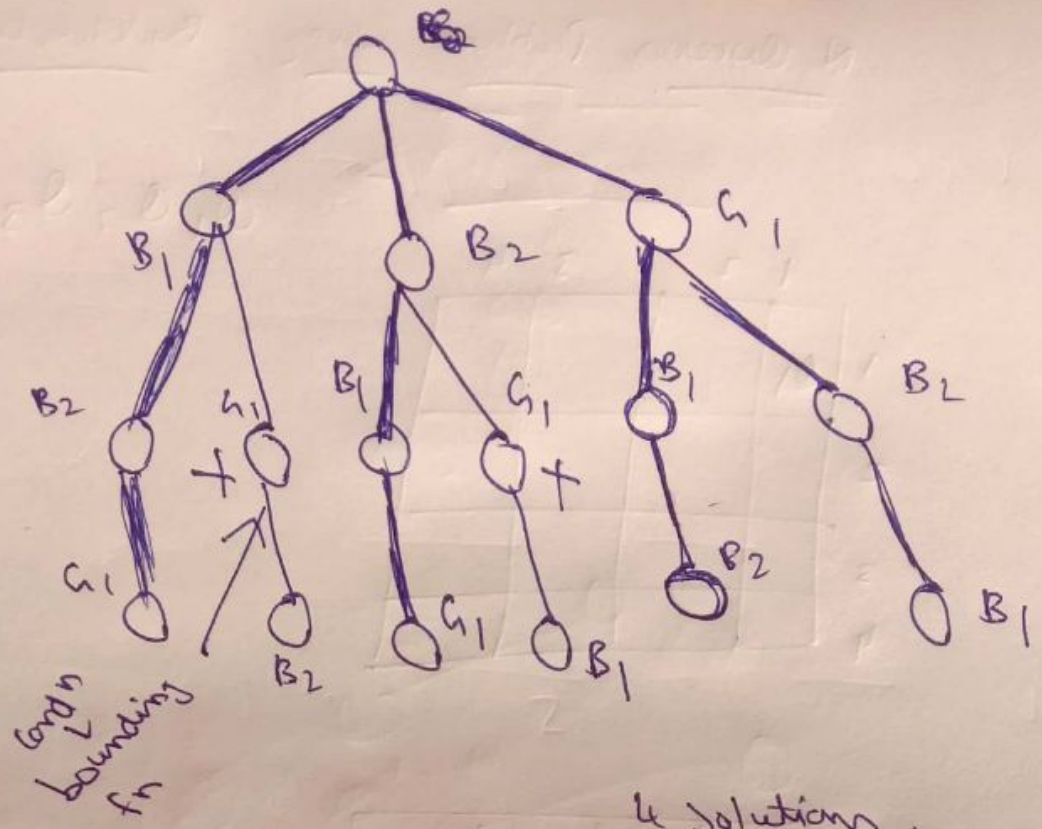
State Space Tree →





Constraint → Girl shouldn't be in middle

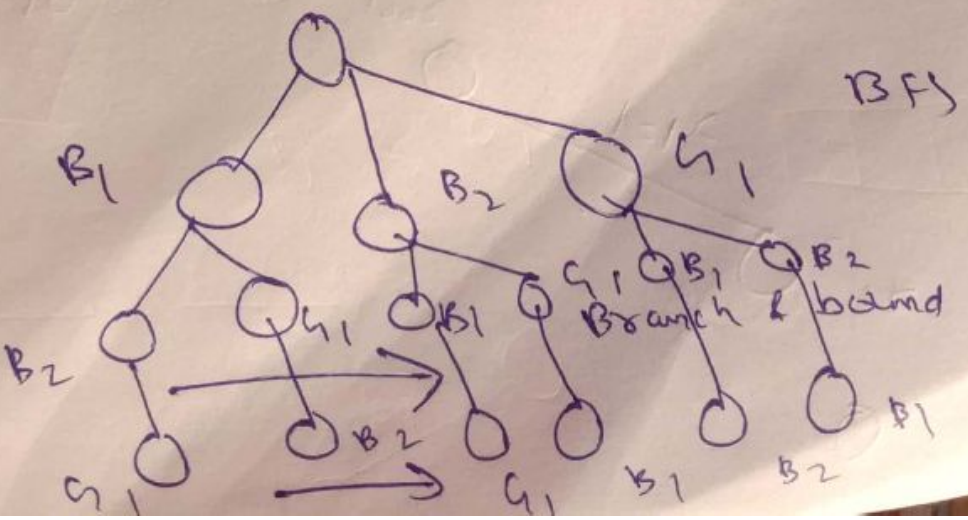
Backtracking used to solve problems consisting of constraints.



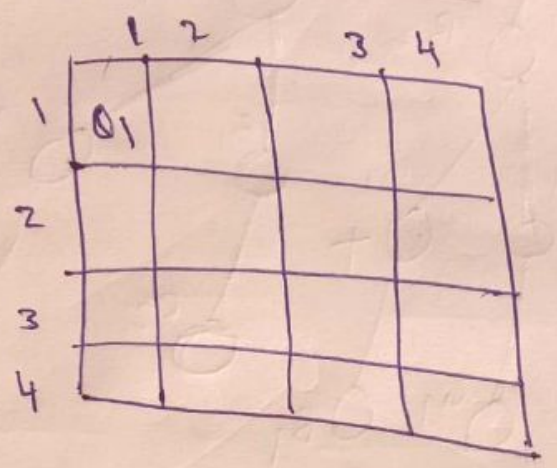
4 solutions.

Branch and bound → Both (brand and bound backtracking) follows brute force approach and generates state space tree

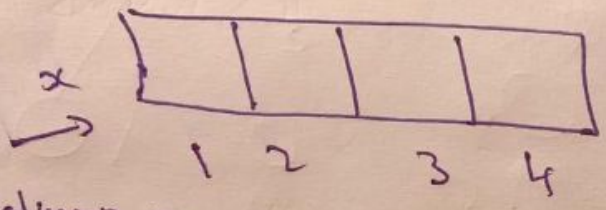
Back tracking follows DFS where branch and bound generates BFS.



N Queens Problem using Backtracking



$Q_1 Q_2 Q_3 Q_4$



Column no

Same

→ row ✗

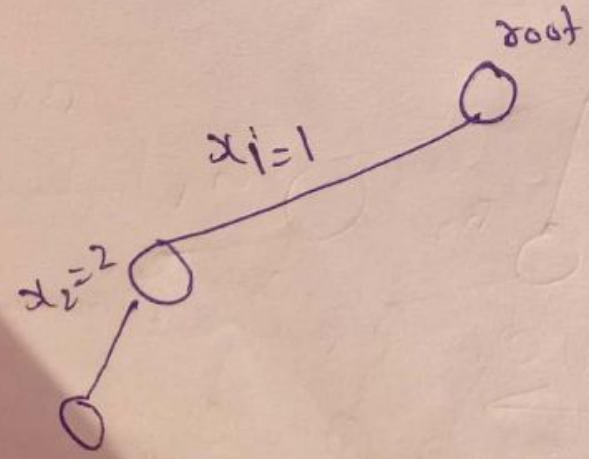
→ column ✗

→ diagonal

$^{16}C_4$

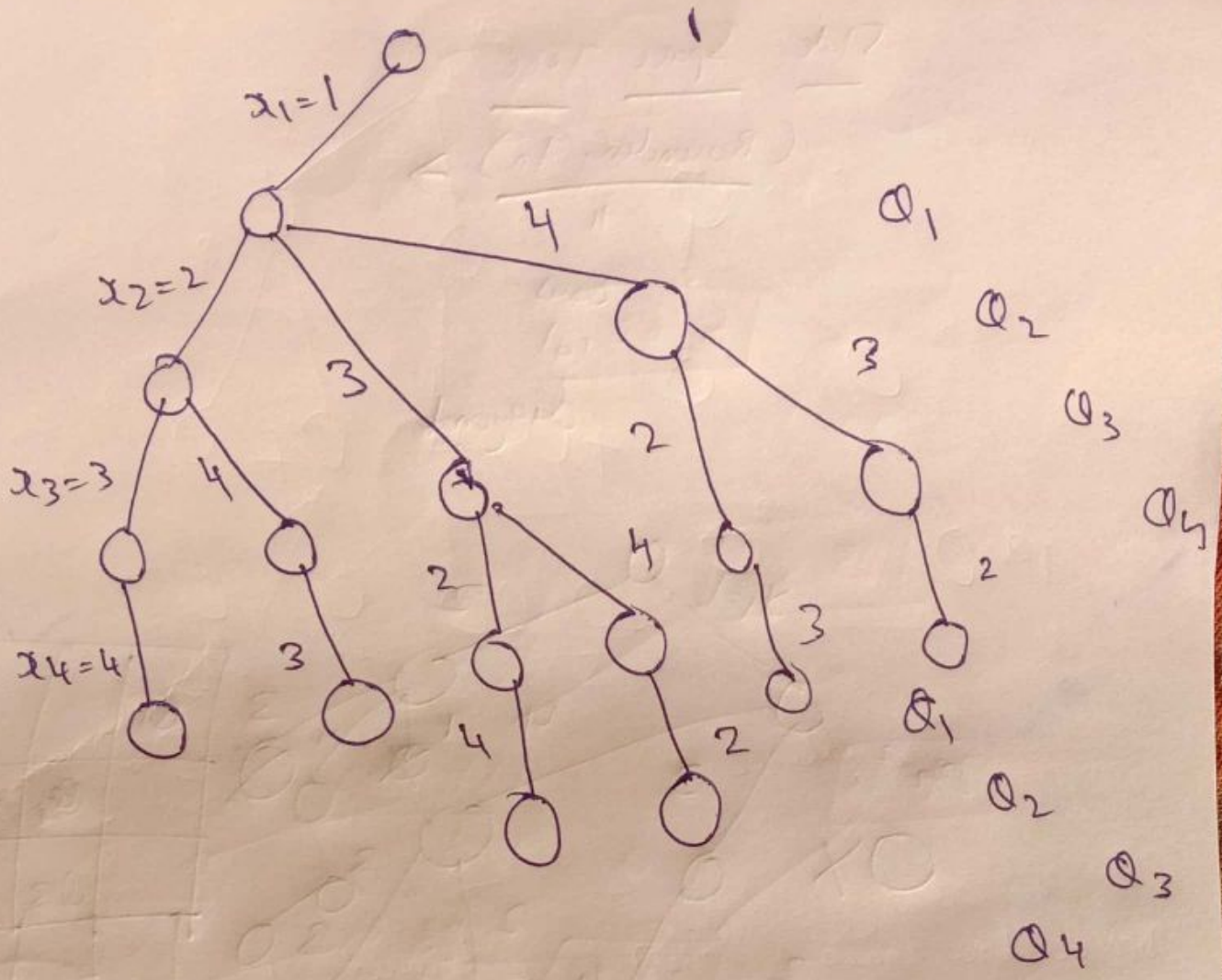
Avoid keeping Queens in same column & row,

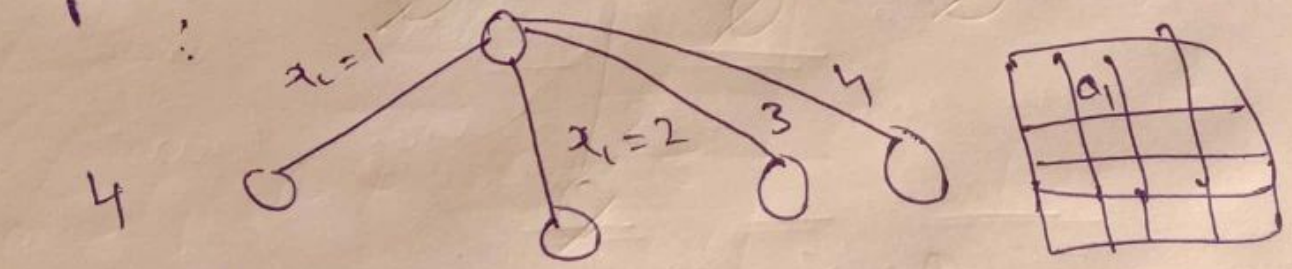
State Space tree



Q_1

Q_2





Allowing diagonals

$$3 \quad 1+4+4\times3+4\times3\times2+4\times3\times2\times1$$

$$= 1+\sum_{i=0}^{3}\left[\prod_{j=0}^{i}(4-j)\right]$$

$$= \underline{65}$$

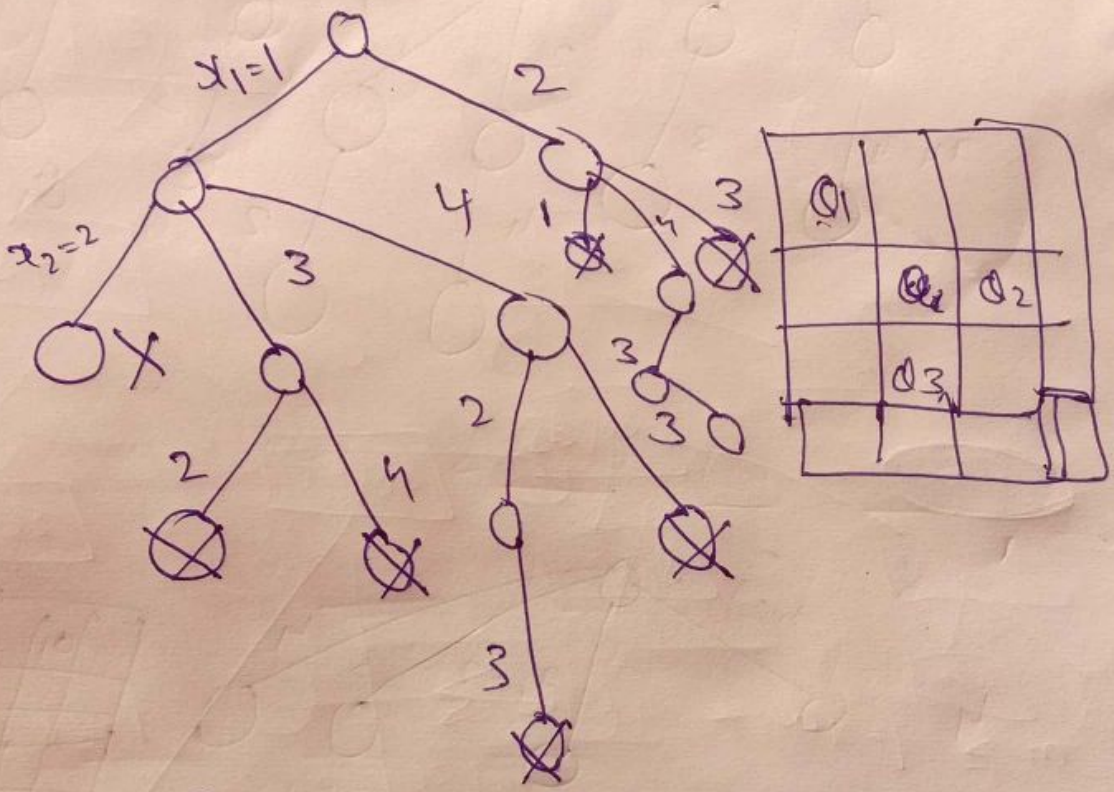
State Space Tree

(Bounding fn) →

row

col

diagonal

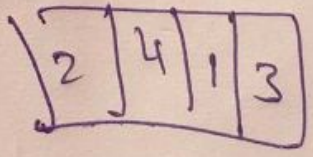


Q1

Q4

Q3

Q3



2 4 1 3 ✓

3 1 4 2

Soln

mirror

image

Q1

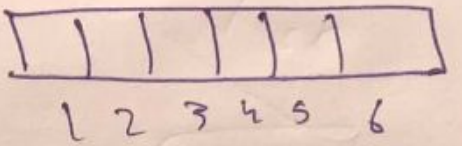
Q2

Q3

Q4

Sum of Subset → Backtracking

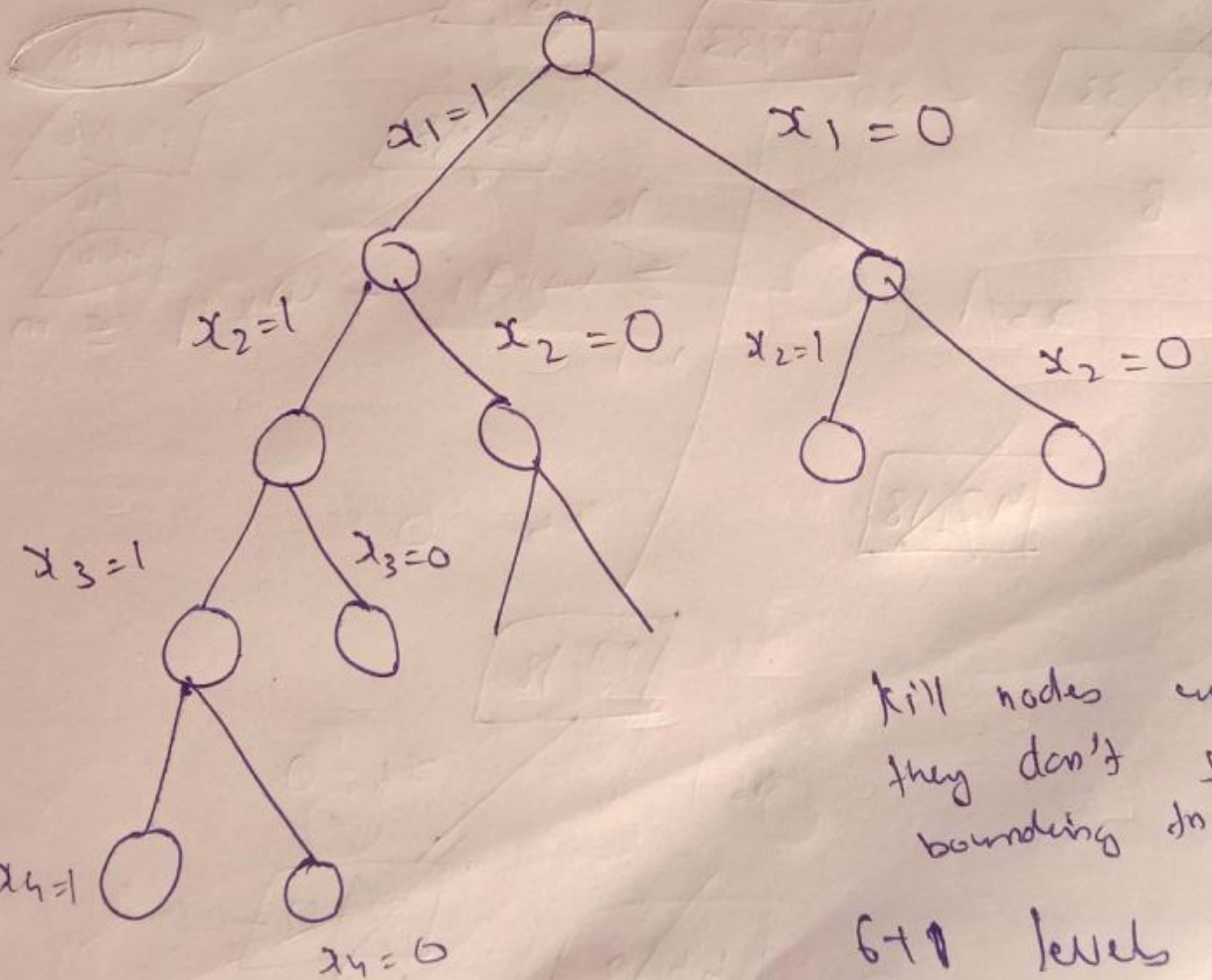
$w[1:6] = \{5, 10, 12, 13, 15, 18\}$ (1 2 3 4 5 6)

$n = 6 \quad m = 30 \quad x$ 

$x_i = 0/1$

Sum is exactly = 30

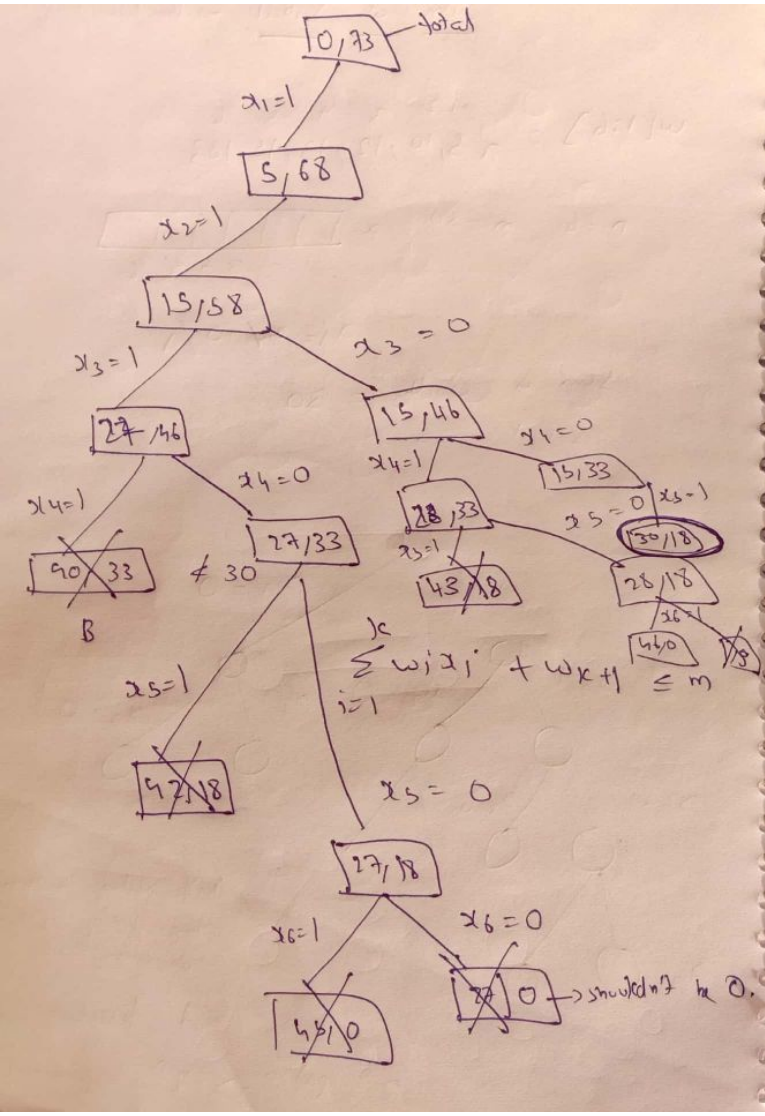
Subset = 30



kill nodes when they don't satisfy bounding fn.

6+1 levels

2^6 paths $\Rightarrow 2^n$ exponential.



$$\sum_{i=1}^{k} w_i x_i + \sum_{j=k+1}^{n} w_j > m$$

$27 + 0 \neq 30.$

1	1	0	0	1	0

No. than one soln possible.

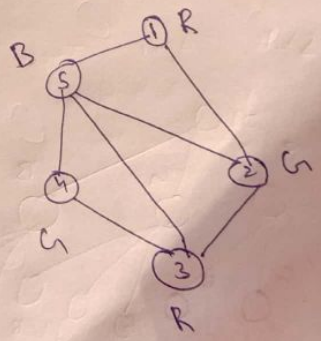
Recursive algorithms.

DFS

$$\sum_{i=1}^{k} w_i x_i + w_{k+1} \leq m$$

Graph Coloring Problem

Backtracking →



m = 3

R, G, B

1 2 3 4 5

R G R G B

More than one soln possible

ways to

① Color a graph

② m Coloring decision Problem

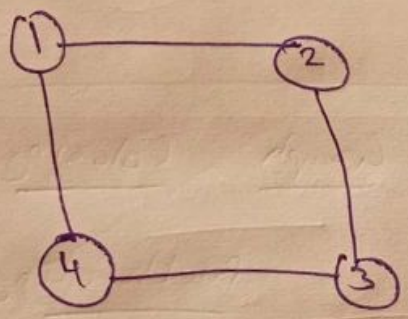
③ m Coloring optimization Problem

Chromatic number

graph can be colored or not with those vertices

min no. of colors required.

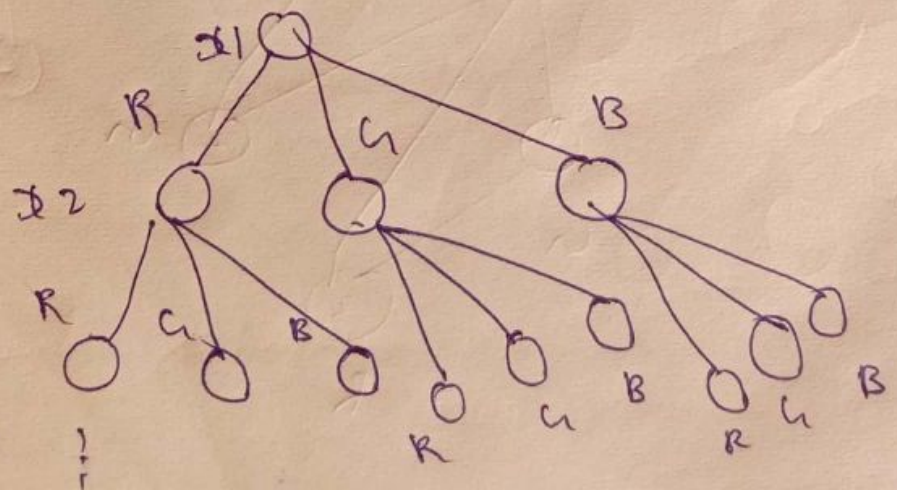
Q.



m = 3

{R, G, B}

State Space Tree



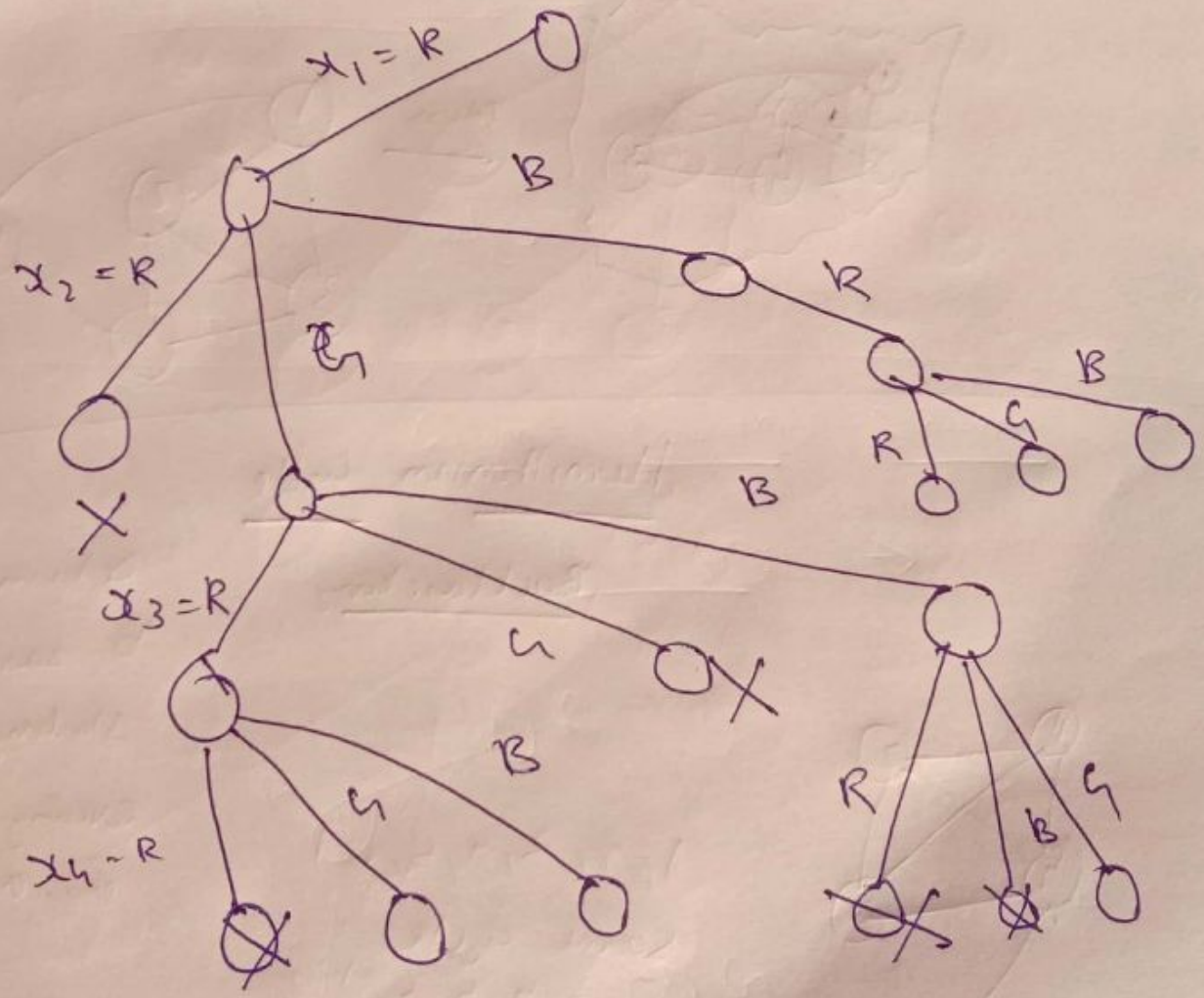
All possible cases without checking cond^n

$1 + 3 + 9 + 27 + 3^4 = |2|$

C^{n+1} $\quad 3^{n+1}$ $\quad m=3$

exponential.

using backtracking,



R G R G

R G R B

R G B G

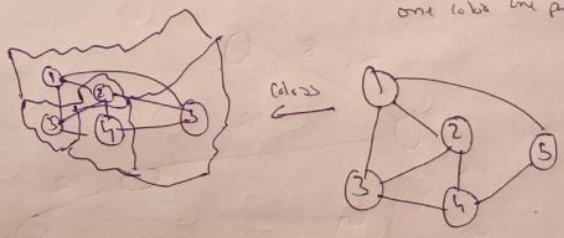
R B R G

R B R B

Map Coloring → two adjacent colors are similar

Minimum no. of colors should be used. Map can be converted into Graph

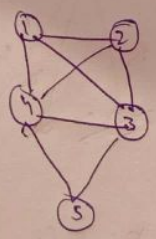
ancient times one color one pro.



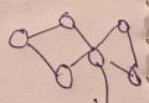
Hamiltonian Cycle

Backtracking

Return to same starting pt. Covering all pts. only once.

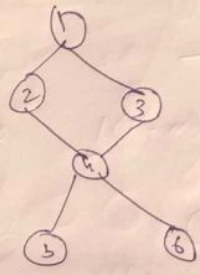


1, 2, 3, 4, 5, 6, 1
same cycle
2, 3, 4, 5, 6, 1, 2



articulation pt ✗

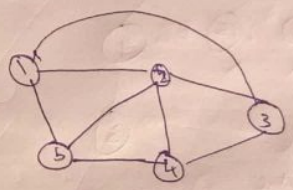
Graphs having articulation pt. cannot have hamiltonian cycle.



Pendant

Hamiltonian Not Possible

Q.



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

x | 0 | 0 | 0 | 0 | 0 |
1 2 3 4 5

(1,2) → 1
1+1 = 2
already in

(1,3) → 3

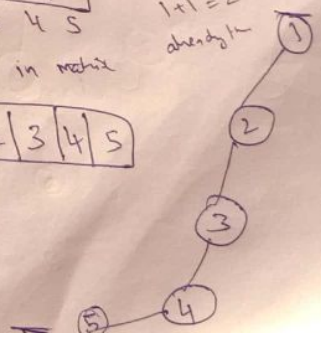
1,1 ⇒ 1,2 check in matrix

1,1 ⇐ | 1 | 2 | 3 | 4 | 5 |

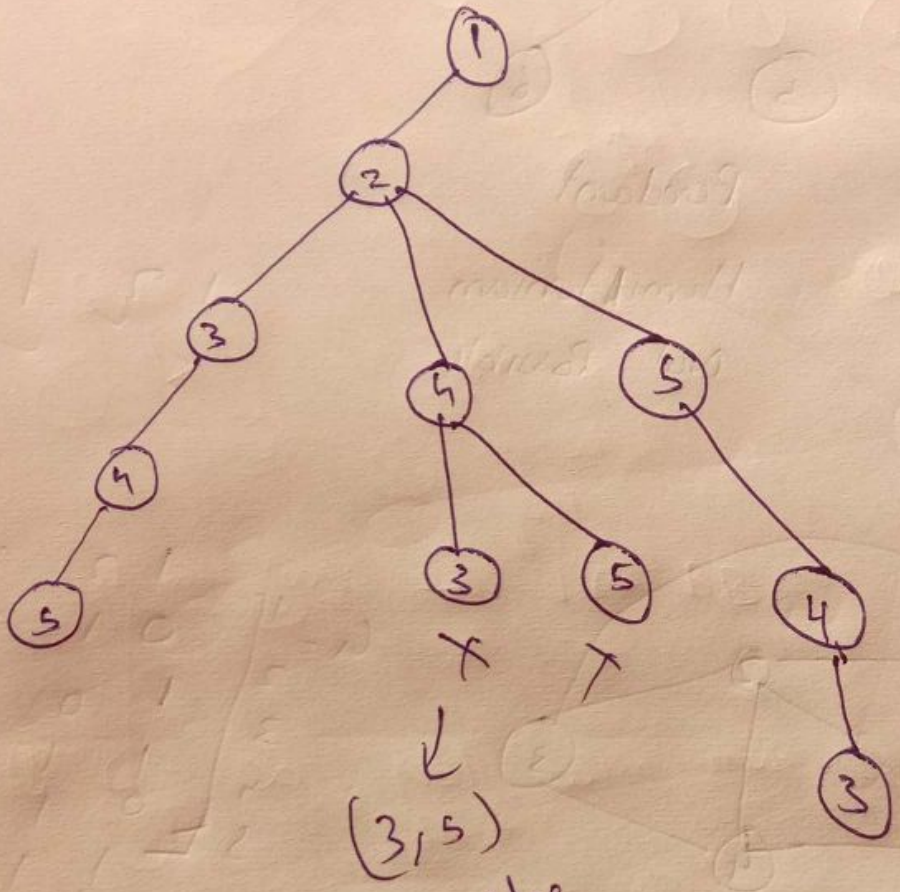
already there

1,2

1 2 3 4 5



4! (n-1)! $O(n^n)$



1 2 3 4 5 1

1 2 5 4 3 1

Branch And Bound →

(Also uses state space tree)

(Used for solving optimization problem)

(only minimization)

Job Sequencing →

Jobs = { J_1, J_2, J_3, J_4 }

P = { 10, 5, 8, 3 }

d = { 1, 2, 1, 2 }

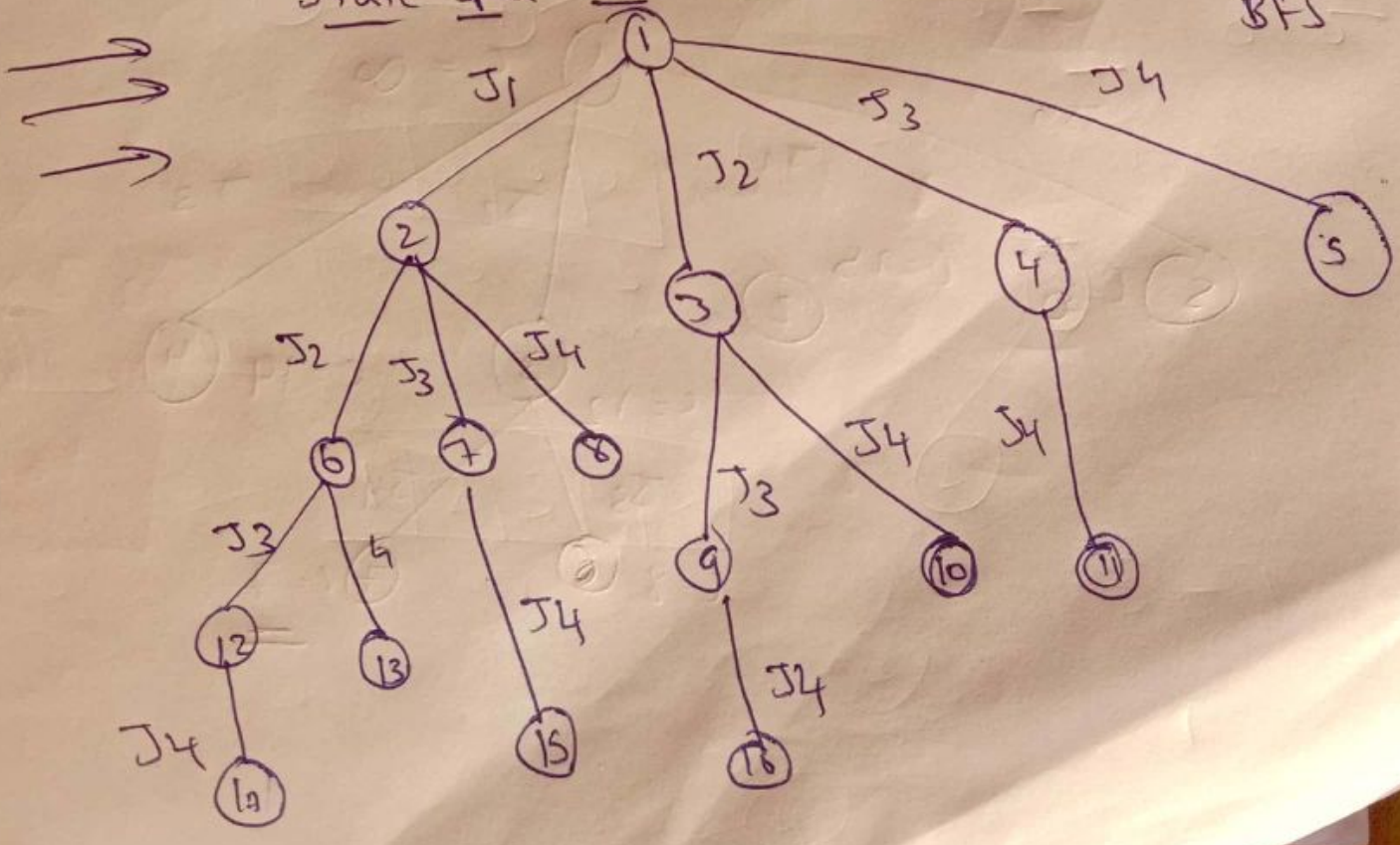
Backtracking
DFS
Branch & Bound
BFS

- S_1 = { J_1, J_4 } variable sized.

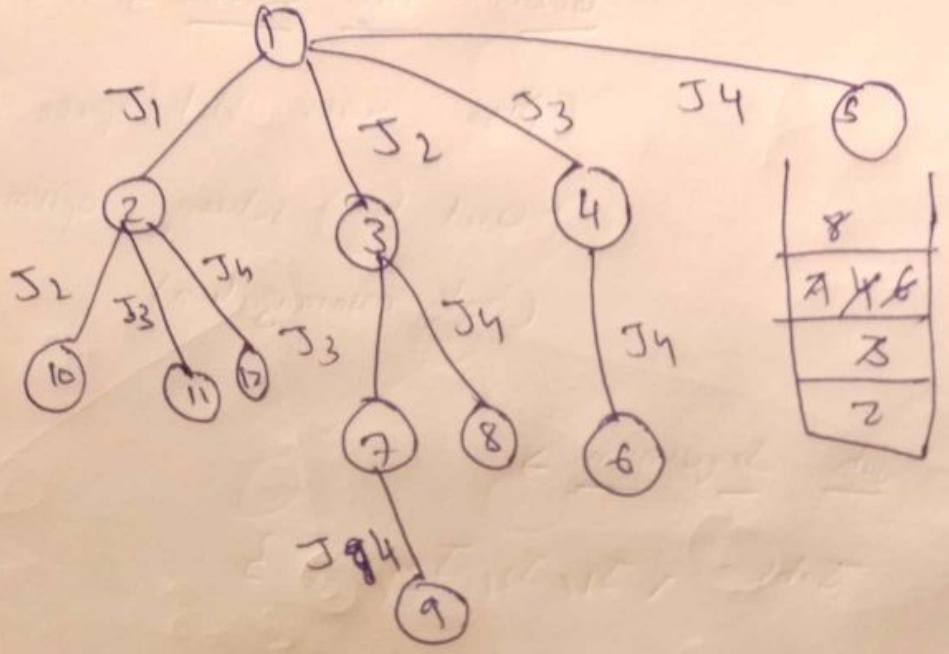
S = { 1, 0, 0, 1 } fixed sized

State Space Tree for solutions →

BFS



2nd method



Stack → LIFO

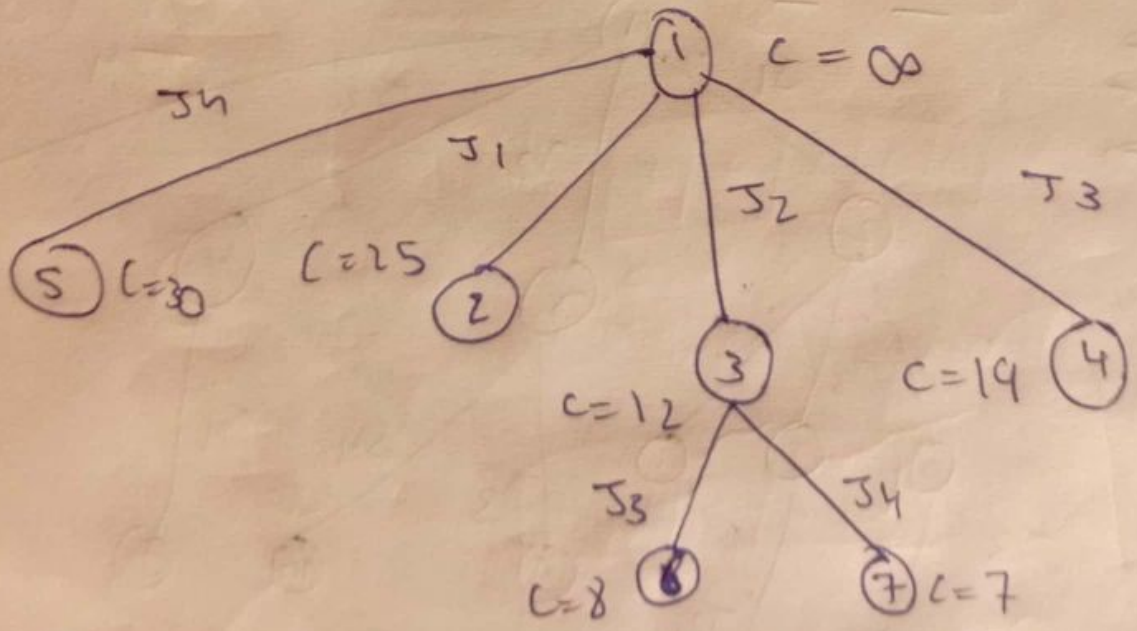
BFS — Once a node is selected explore it

Queue → FIFO

Least cost → LC - BB

BFS only

(1) $C = \infty$ Node with min. cost should be explored first



Job Sequencing with deadline →

It can also be solved using greedy method and dynamic programming.

Soln using Branch And Bound →

(only for minimization)

Profit → Penalty.

Jobs	1	2	3	4
Penalty	5	10	6	3
deadline time	1	3	2	1
	1	2	1	1

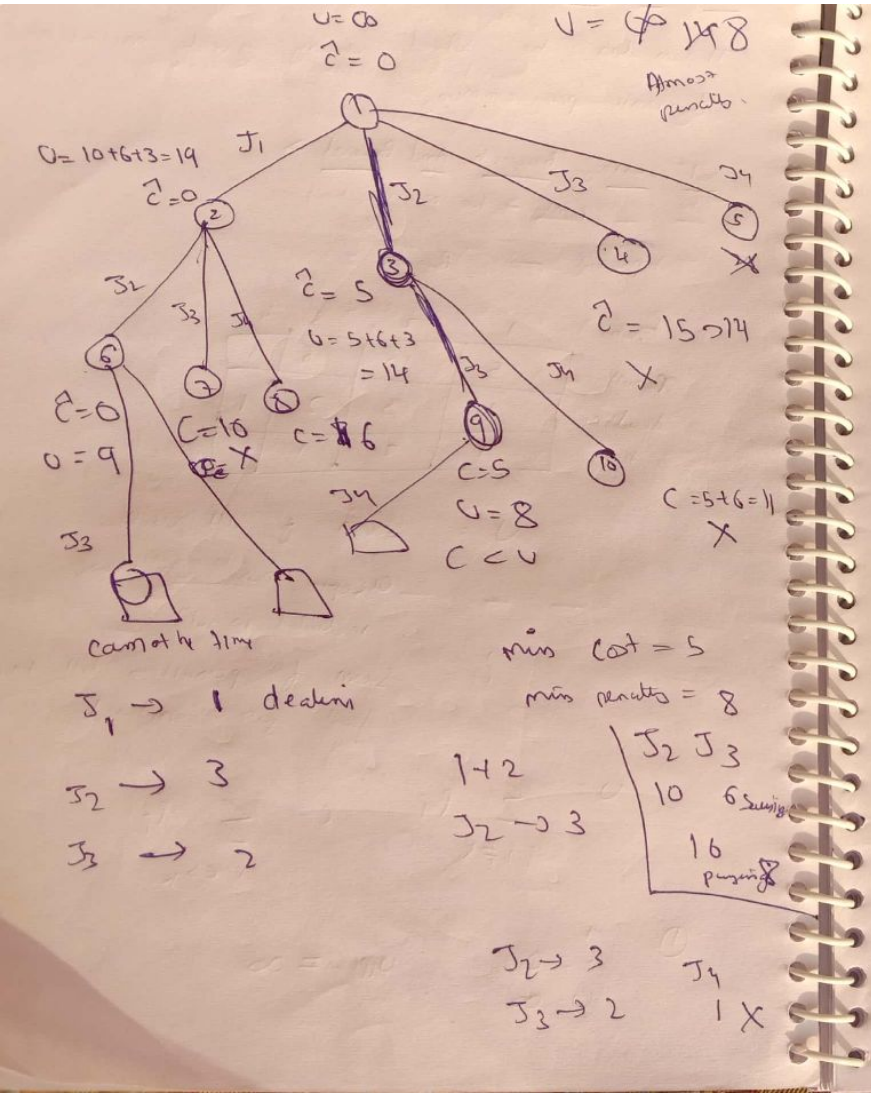
U = Sum of all penalties except those which are included in soln

$\hat{c}$ = Sum of penalties till the last job considered.

$$U = \sum_{i \notin S} P_i \qquad c = \sum_{i \in S_k} P_i$$

(1) $\hat{c} = 0$ upper = ∞

$U = \infty$



0/1 Knapsack Problem

	1	2	3	4
Profit	10	10	12	18
weight	2	4	6	9

Total wt. $\leq$ m = 15
n = 4.

Take complete object

Maximization

Greedy method, DP, Back tracking

Branch & bound

LC - BB

$$U = \sum_{i=1}^{n} p_i x_i \leq m$$

$$C = \sum_{i=1}^{n} p_i x_i$$

$S = \{x_1, x_2, x_3\} = \{1, 0, 1, 0\}$

Upper = ∞

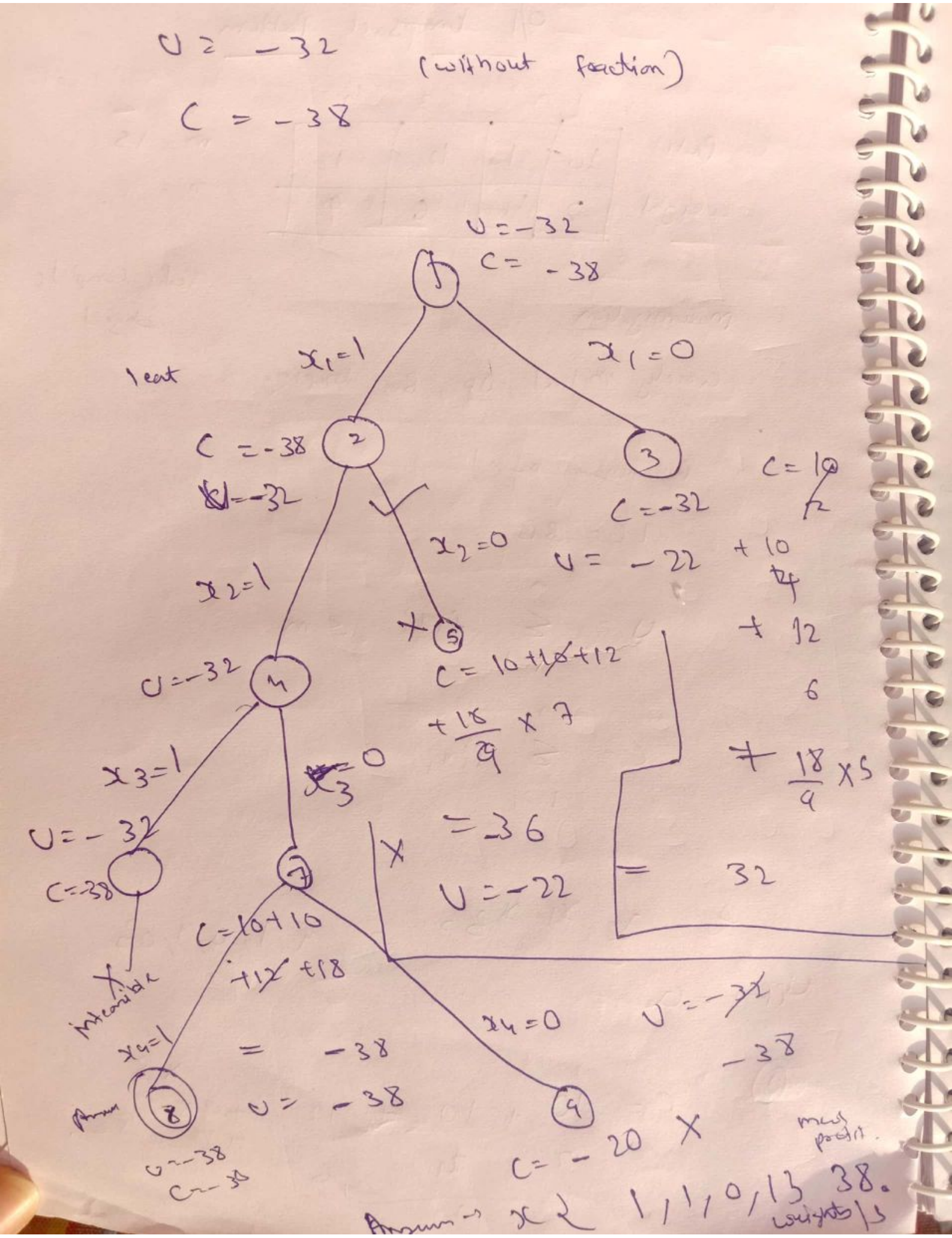
① $C = 10 + 10 + 12$ (x3) $+18 = 38$

2 4 6 9

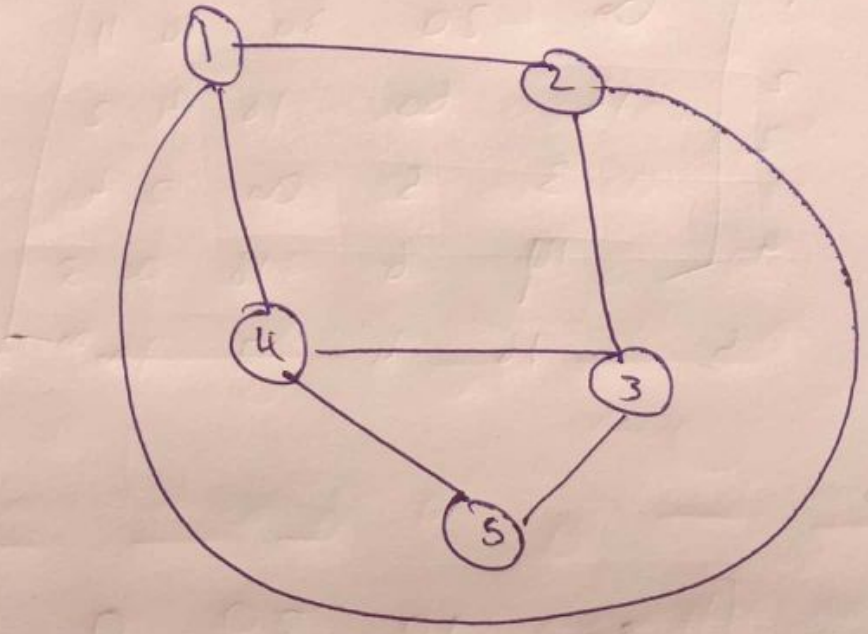
15 - 12 = 3 more kg.

$U \geq -32$ (without fraction)

$C = -38$



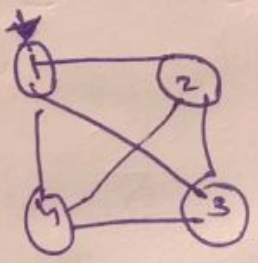
Traveling Sales person Branch n Bound



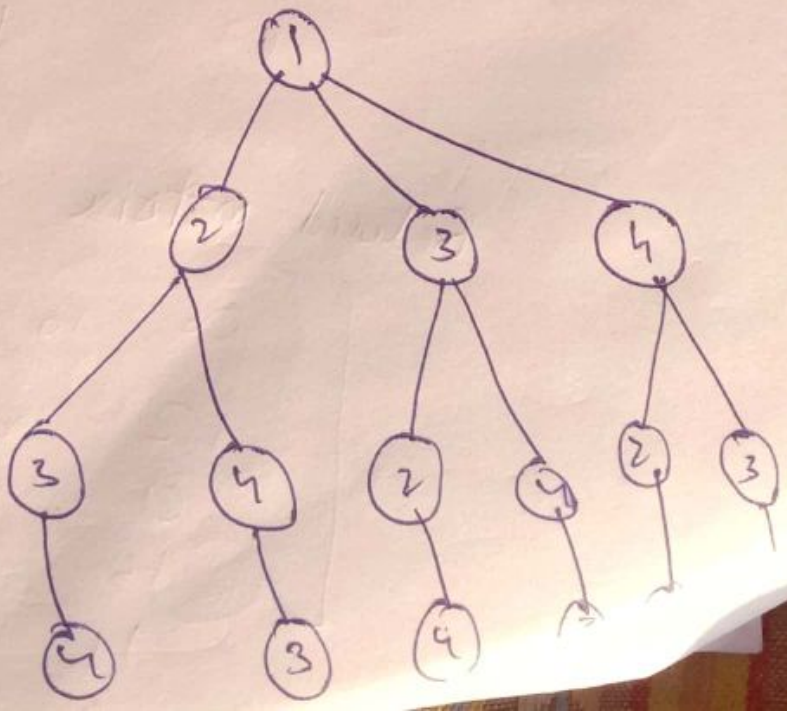
	1	2	3	4	5
1	∞	10	30	10	11
2	15	∞	16	4	2
3	3	5	∞	2	4
4	19	6	18	∞	3
5	16	4	7	16	∞

Cost adjacency matrix

Q.



State Space Tree showing all possible tours.



	1	2	3	4	5	
1	∞	20	30	10	11	10
2	15	∞	16	4	2	2
3	3	5	∞	2	4	2
4	19	6	18	∞	3	3
5	16	4	7	16	∞	4

21

⇒

∞	10	20	0	1	10
13	∞	14	2	0	2
1	3	∞	0	2	2
16	3	15	∞	0	3
12	0	3	13	∞	7
					21

1 0 3 0 0

✓ ✓

>0 >0

Reduced matrix

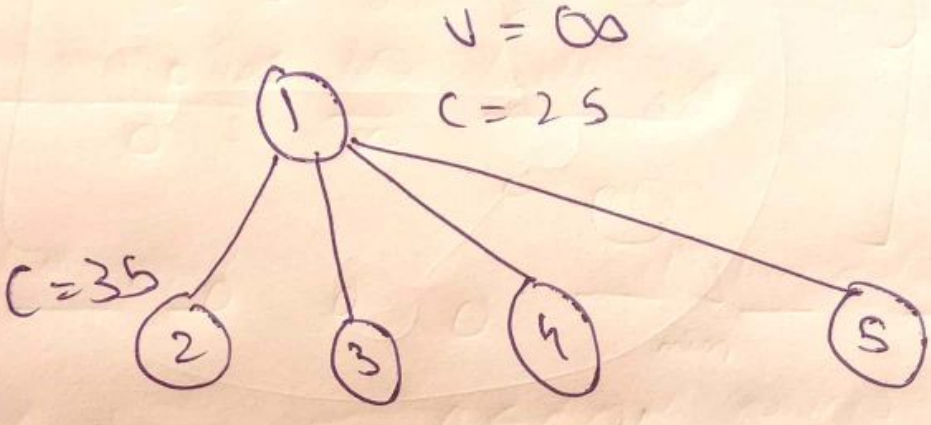
⇒

∞	10	17	0	1	
12	∞	11	2	0	21+4
0	3	∞	0	2	=25
15	3	12	∞	0	Cost
11	0	0	13	∞	of
					induction

we founded min distance.

Atleast cost should be 25.

Reduced cost = 25



=>

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	2	0
3	0	∞	∞	0	2
4	15	∞	12	∞	0
5	11	∞	0	12	∞

2 → 1 X

every row column must have 0

C(1,2) + r + r̂ (any other reduction)

10 + 25 + 0

= 35

1 → 3

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ (12) & 0 & \infty & 12 & 0 \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

min 0 0 0 0

||

$$\Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & 0 \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & 0 \end{bmatrix}$$

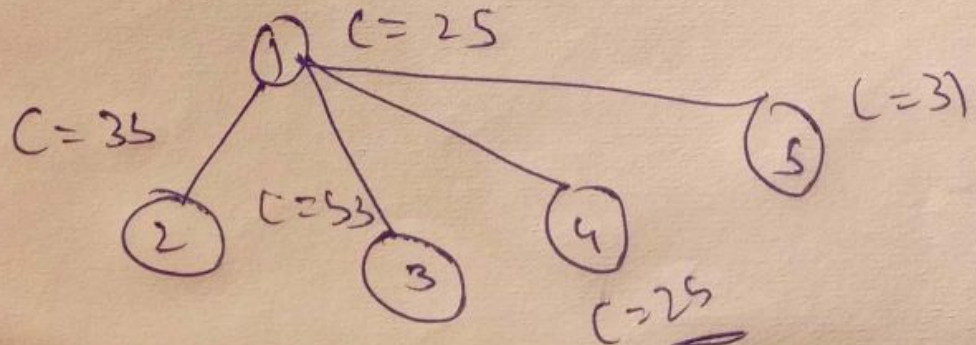
7

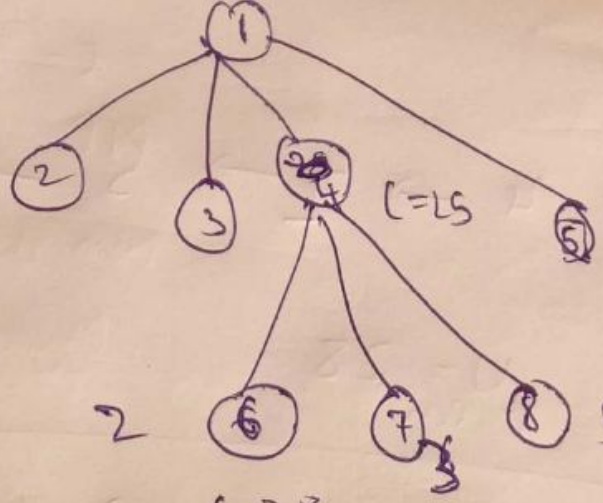
$c(1,3) + 8 + 0$

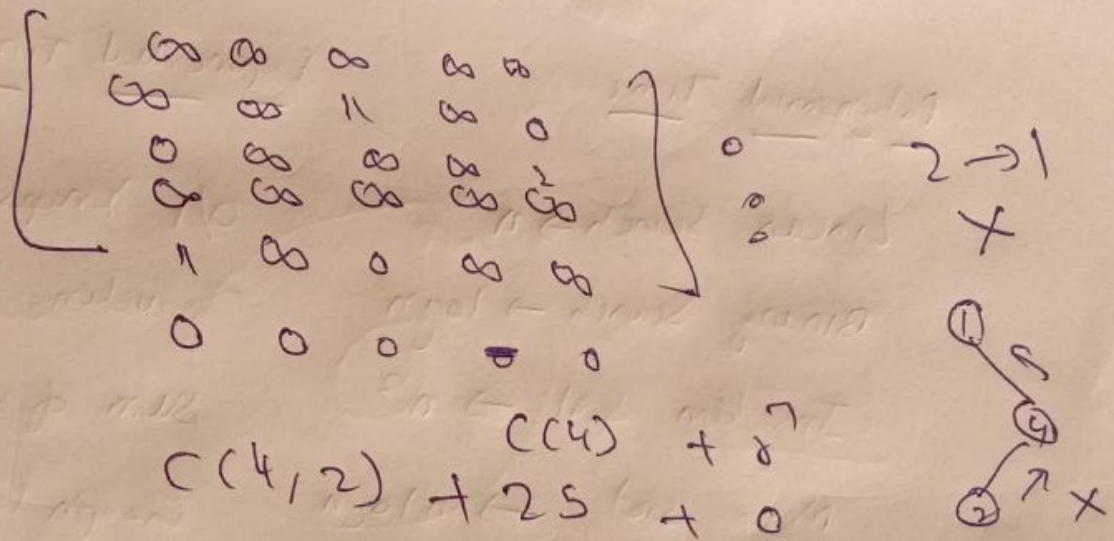
$= 17 + 25 + 11$

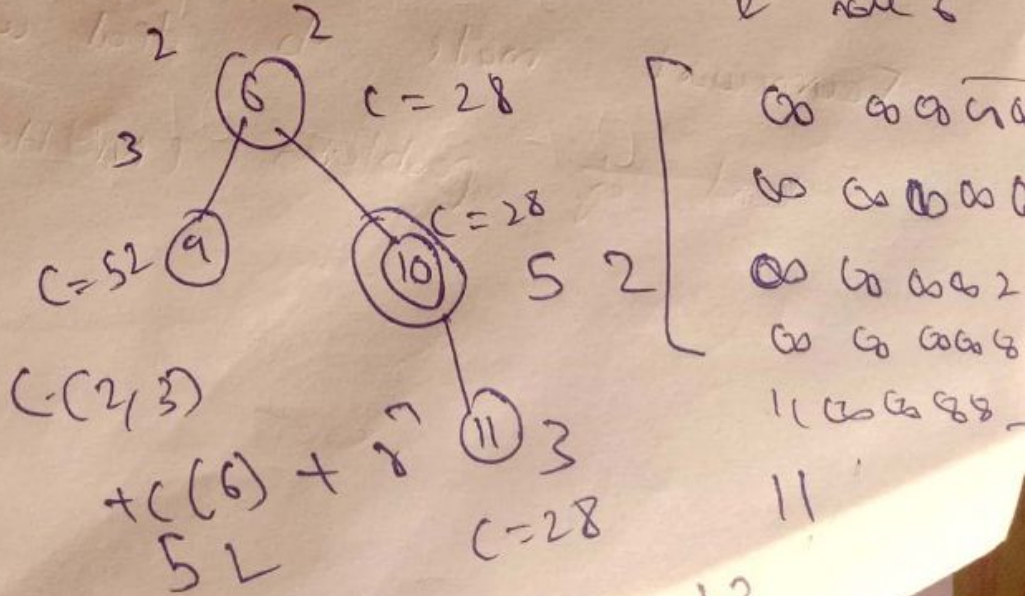
$= 53$

similarly,









C = 28

1 → 4 → 2 → 5 → 3 → 1

V = 28
PPS

NP-Hard and NP-Complete →

Polynomial Time

Linear Search → n

Binary Search → log n

Insertion Sort → n^2

Merge sort → n log n

Matrix Multiplication → n^3

Exponential Time

0/1 Knapsack → 2^n

Traveling SP → 2^n

Sum of subsets → 2^n

Graph Coloring → 2^n

Hamiltonian Cycle → 2^n

Very time consuming

Framework made to deal with this kind of problems (NP-Hard and NP-Complete)

Algorithms with which we work are generally deterministic. We know the working of such algorithms we work.

For non-deterministic algorithms, we don't know the working of algorithms we write

In non-deterministic, we preserve the work. So that in future someone can make the non-deterministic part deterministic.

What code the work should be preserved.

Non deterministic →

```
Algorithm NSearch (A, n, key)
{   j = choice();        ——— 1   we don't know the result of choice (omega)
    if (Key = A[j])
    {  write (j);
       success();        ——— 1
    }
    write (0);
    Failure();           ——— 1
}                                  O(1)
```

P

Deterministic
Algorithm
taking polynomical
time

Single source
Shortest Path
Huffman coding
etc.

NP

Non deterministic
Algorithm

2^n

0/1 knapsack
Travelling SP
Sum of subsets
Graph coloring
Hamiltonian cycle

Merge sort $(n \log n)$ NP $\Rightarrow$ P now

Bubble sort (n^2)

Today whatever are deterministic today
were earlier non deterministic.

If unable to solve atleast relate them
together for future.

(CNF) (Base formula)

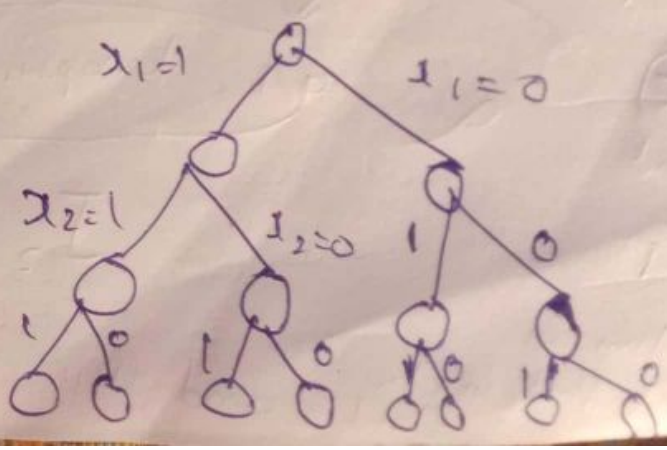
$x_i = \{x_1, x_2, x_3\}$

$CNF = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$

c_1 c_2

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

$2^3 \Rightarrow 2^n$



0/1 knapsack

$P = \{10, 8, 12\}$ $n = 3$

$w = \{5, 4, 3\}$ $m = 8$

$x_i = \{0/1, 0/1, 0/1\}$

2^3.

$n \rightarrow 2^n$

Exponential

Satisfiability $\rightarrow 2^n$ $\rightarrow$ NP - Hard

Base

0/1 knapsack

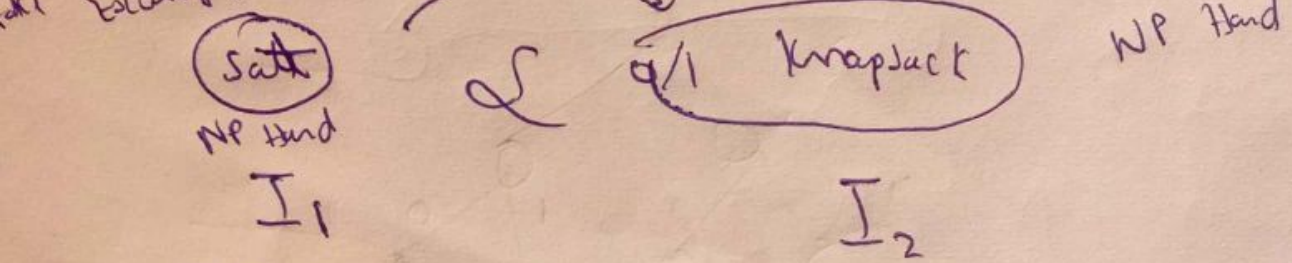
Traveling SP

2^n $\rightarrow$ Hard

:

Reduction $\rightarrow$ Poly

Example to prove



I_1 I_2

If this is solved that can be solved and vice-versa.

Sat $\propto$ L

if true NP Hard

Sat $\propto L_1$ $\quad L_1 \propto L_2$

NP Hard.

Sat $\propto L_2$

NP Hard NP Complete

↓ Reduced by Sat

Np Hard

↓

if we writes non deterministic algorithm it becomes NP complete.

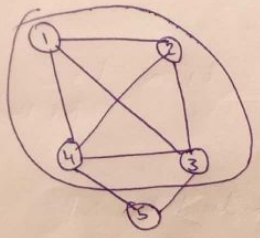
$P \subseteq NP$

$P = NP$

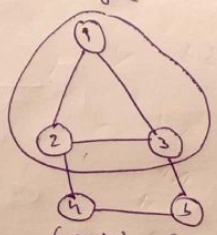
Cook's if Satisfiability will be in P only

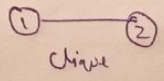
if $P = NP$

NP - Hard Graph Problems → Clique Decision Problem (CDP)



Clique





Clique

Complete Graph

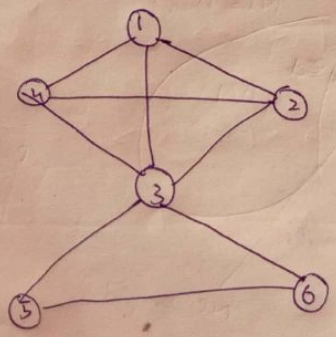
$|V| = n$

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

Clique →

Subgraph of a graph which is complete.

Q.



$k = 4$ max clique

$k = 3$

$k = 2$

Decision Problem → find if a graph is having a clique of size k.

Optimization Problem → find max clique in a graph

If decision problem is NP Hard optimization problem is also NP Hard.

$P = L_2$

Set CDP

$L_1 \propto L_2$

$I_1 \quad I_2$

x_1, x_2, x_3

$F = \underset{C_1}{(x_1 \vee x_2)} \wedge \underset{C_2}{(\bar{x}_1 \vee \bar{x}_2)} \wedge \underset{C_3}{(x_1 \vee x_3)}$

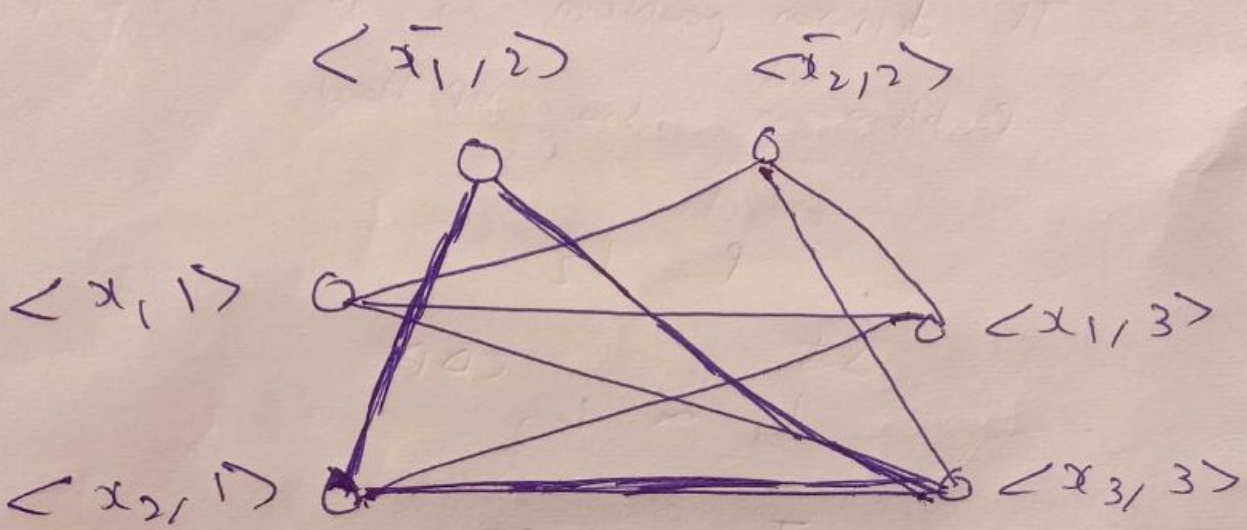
$F = \bigwedge_{i=1}^{k} C_i$

$\langle x_1, 1 \rangle$ $\langle \bar{x}_1, 0 \rangle$

$\langle x_2, 1 \rangle$ $\langle \bar{x}_2, 0 \rangle$

$V = \{ \langle a, i \rangle \mid a \in C_i \}$

$x_1 \not\rightarrow \bar{x}_1$



$E = \{ (\langle a, i \rangle, \langle b, j \rangle) \mid i \neq j \text{ and } b \neq \bar{a} \}$

$G = \{ \langle a, i \rangle \mid a \in C_i \}$

$k = 3$ x_1 x_2 x_3

0 1 1

$F = (0 \vee 1) \wedge (1 \vee 0) \wedge (0 \vee 1)$

$= 1$

Knuth - Morris - Pratt (KMP) Algorithm →

	1	2	3	4	5	6	7	8
String :	a	b	c	d	e	f	g	h

pattern : d e f

There is also a basic algorith (naive algorithm) but kmp is better than it.

Basic Algorithm →

e.g. :-

a b c d e f g h

d e f

→ a b c d e f g h

d e f

→ a b c d e f g h

d e f

→ a b c d e f g h

d e f

All are matching.

Q.

i → i

String: a b c d a b c a b c d f

Pattern: a b c d f

j → j

i ≠ j not matching

i

5

i_2 if not matched

if not matched i is shifted back

that's the drawback of basic algorithm

i.e. time wastage.

i

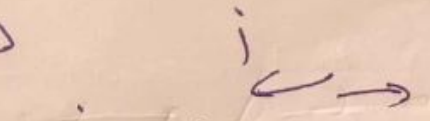
a b c d a b c a b c d f

a b c d f

1 2 3 4 5

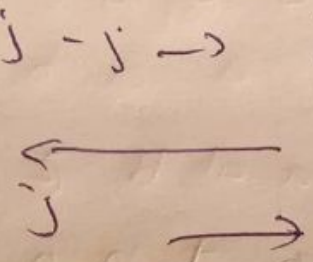
j

Worst Case →



n String: a a a a a a a b
1 2 3 4 5 6 7 8

pattern: a a a b
m 1 2 3 4

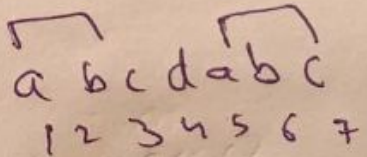


basic (Naive Algorithm)

O (m x n).

KMP →

Terminology :-



① prefix : a, ab, abc, abcd, ...

Suffix : c, bc, abc, dabc, ...

pi table

π or lps (longest prefix same with suffix)

P_1 : a b c d a b e a b f

0 0 0 0 1 2 0 1 2 0

P_2 :

a b c d e a b f a b c

0 0 0 0 0 1 2 0 1 2 3

P_3 : a a b c a d a a b e

0 1 0 0 1 0 1 2 0 0

P_4 : a a a a b a a c d

0 1 2 3 0 1 2 0

Q. String :

i

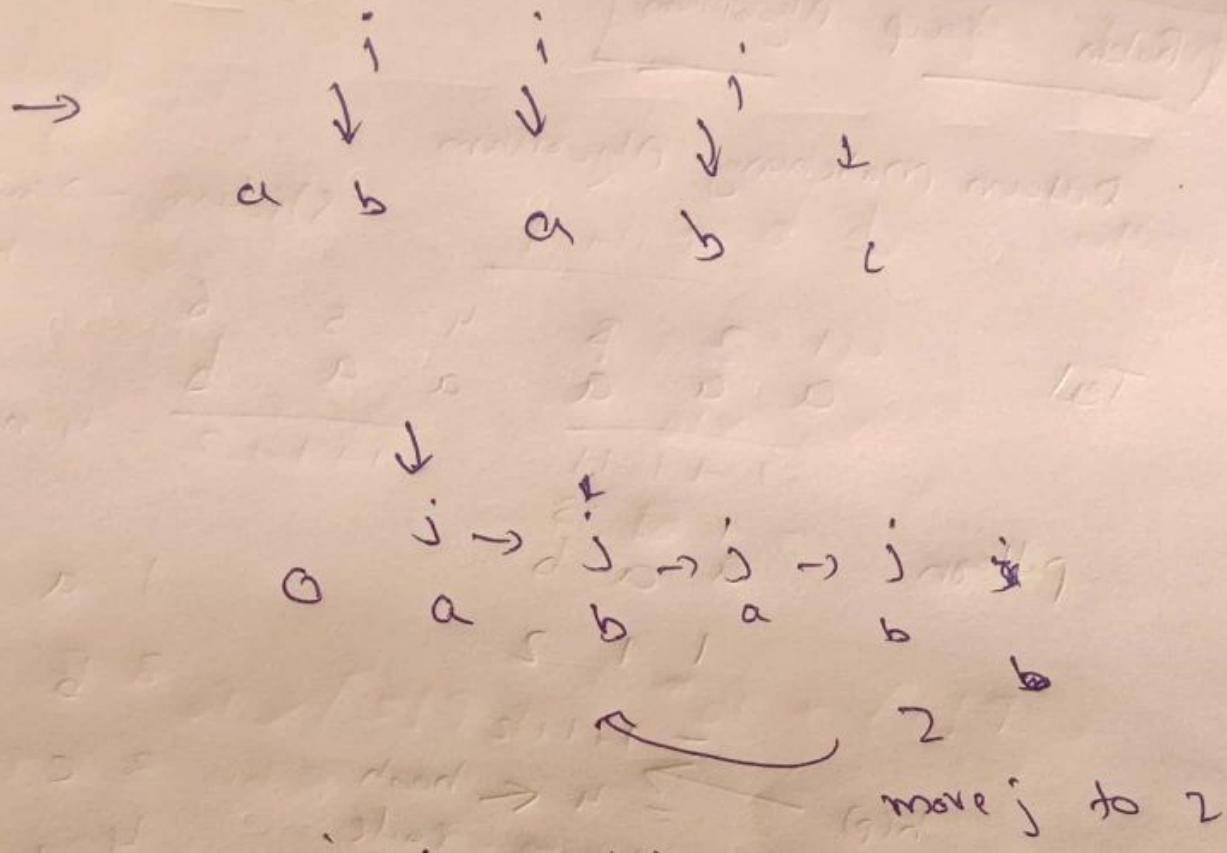
a b a b c a b c a b a b a b d

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

j pattern :

j → 0

	1	2	3	4	5
0	a	b	a	b	d
1	0	0	1	2	0



If beginning part of the string is appearing somewhere on the string don't again compare the strings from the start

i is never backtraced.

$O(m+n)$ ✓

Titable search

Rabin Karp Algorithm

Pattern matching Algorithm

1+1+1

Sliding -> rolling hash fn

Text

1	2	3	4	5	6
a	a	a	a	a	b

1+1+1 | 1+1+2 = 4 match

pattern

1	2	3
a	a	b

1 1 2

= 1+1+2

h(p) → = 4 ← hash code

1	a	97
2	b	98
3	c	99
4	d	100

1	2	3	4	5	6	7	8	9	10
a	b	c	d	e	f	g	h	i	j

m = 3

n = 6

Once we get a match confirm it we really got the pattern.

Q.

6 - 1 ~~+3~~ +4

~~get~~ = 9

Text

a	b	c	d	a	b	c	e

1+2+3 | 6+1 | 10

pattern

b	c	e

10

Text

1	2	3	4	5	6	7	8	9	10	11
c	c	a	c	c	a	a	e	d	b	a

$3 + 3 + 1 = 7 \Rightarrow c \neq d.$

Pattern

1	2	3
d	b	a

Using matching

$4 + 2 + 1 = 7$

7

c c a c c . . . . $c \neq d$

7

c c a c c $a \neq d$

3 matchings no result

Comparison second time

Spurious Hits →

when length is same but components are different.

in such cases

$O(mn)$

$O(n - m + 1)$ → No Spurious Hits

Avoid Spurious Hits,

10 alphabets

patter d b a

4 + 2 + 1

$\times 10^2$ $\times 10$

= 61.

$P[1] \times 10^{m-1} + P[2] \times 10^{m-2} + P[3] \times 10^{m-3}$

Avg time. $O(n-m+1)$

worst case $O(mn)$

Rabin Karp Fingerprint

Rolling hash fn.

= 400 + 20 + 1 = 421

using hash fn we are reducing chances of worst case

(for 27 alphabets 27^{m-1})

$10 \times [331 - 3 \times 10^2] + 3 \times 10^0 = 313$

c c a c c a a e d b a 421

Same or not

$3 \times 10^2 + 3 \times 10^1 + 1 \times 10^0$ $[133 - 100] \times 10 + 3 \times 10^0$

= 331 = 33

313 − 300 = 13 × 10 + 3

Also can apply % in rolling hash fn. exceeding int variable but intermediate

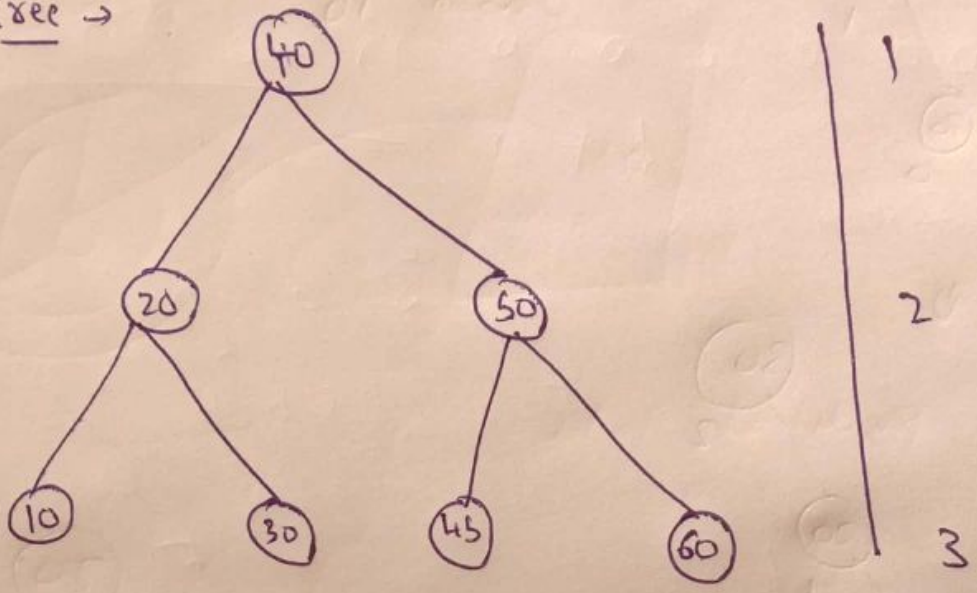
spurious hit

130 + 3 = 133

$[4 \times 10^2 + 2 \times 10^1 + 1 \times 10^0] \% 2^{31}$

AVL Trees

Binary Tree →



min log n

max n

left and right skew tree takes max. time

Rotations →

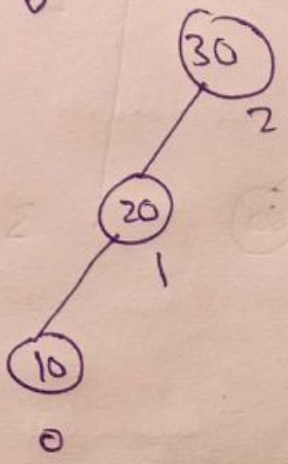
LL
RR
LR
RL

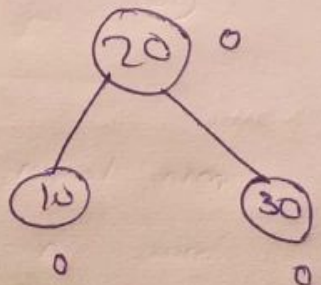
balance factor = height of left subtree − height of right subtree

$bf = h_l - h_r = 2, -1, 0, 1$

$|bf| = |h_l - h_r| \leq 1$ otherwise imbalanced.

Insert 10

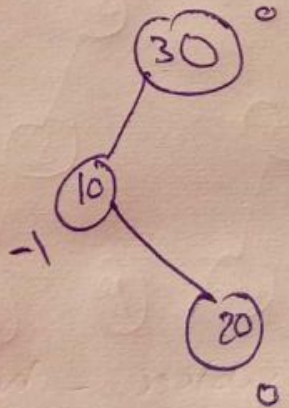


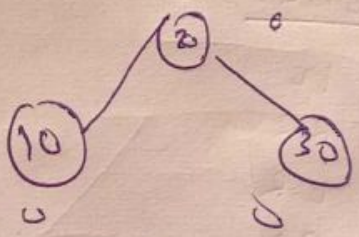


LL rotation

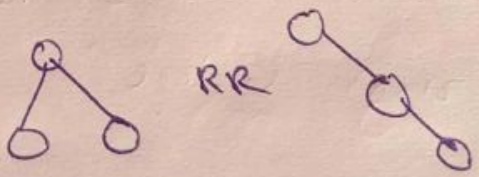
Imbalance due to insertion of left of left.

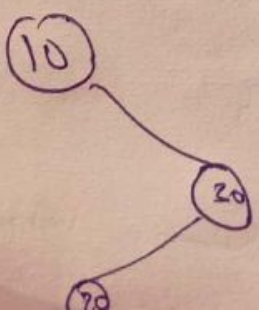
Insert 20





LR



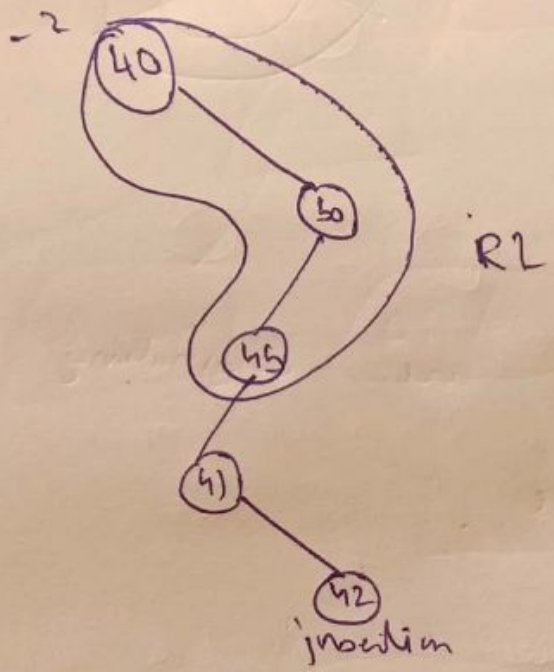


Height balanced BST → AVL

log n always

At most height → 1.44 $\log_2 n$

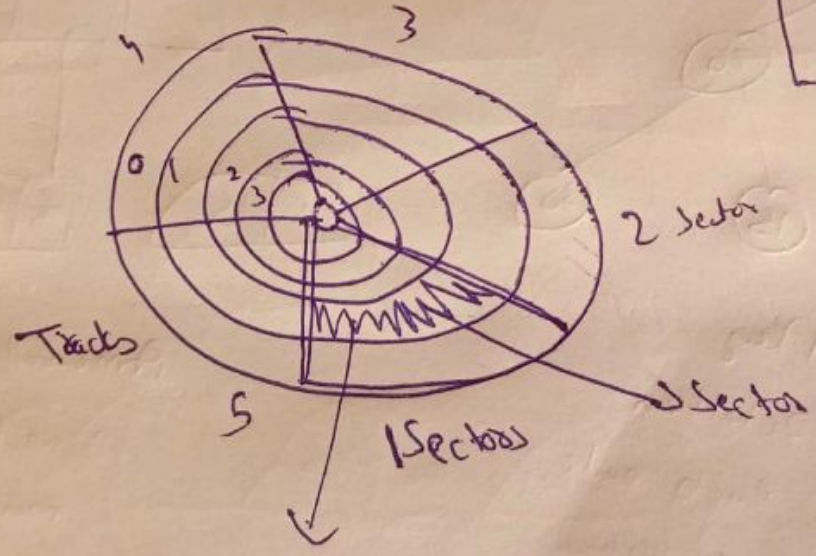
$\Theta(\log n)$



Red black is same as AVL

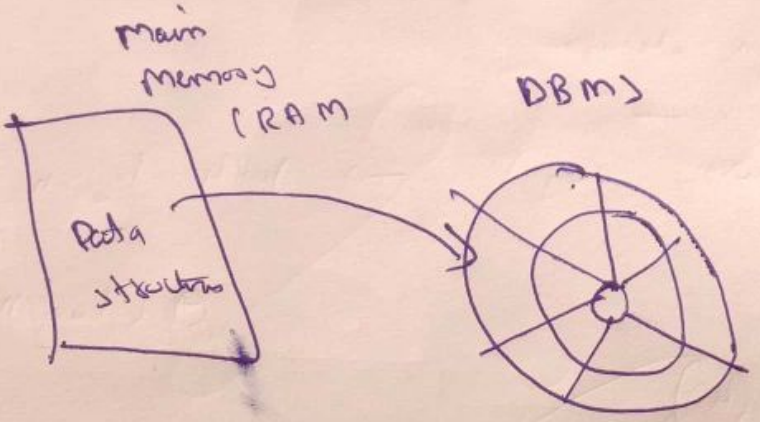
To avoid more frequent rotations of AVL rules, Red black trees are used instead of AVL. They are height balanced

B & B+ Trees →



Block intersection position.

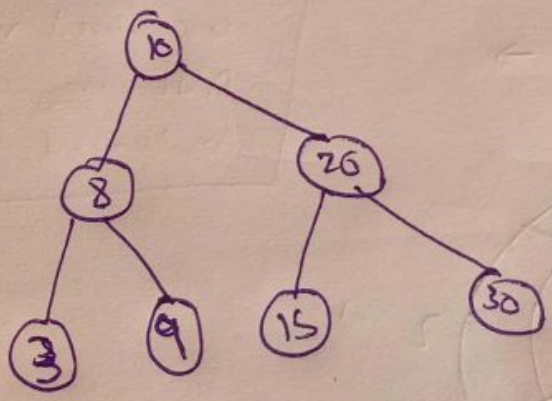
Track no. 3
Sector no. 3 → Block Address



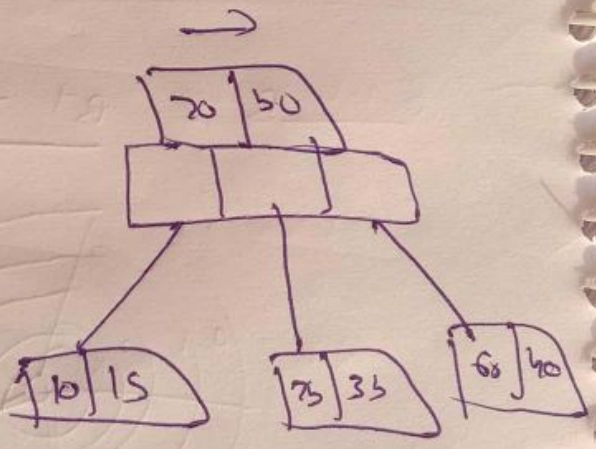
Multilevel Index

B and B+ Trees are originating from m-way search tree.

BST



1 key
2 children
each node
2 children



Amount of searching is more

$k_1 < k_2 < k_3 \ldots$

3-way search tree

m-way search tree → At most m keys.
m-1 keys.

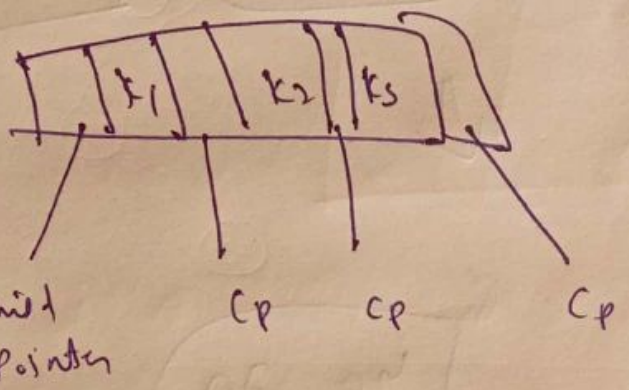
BST

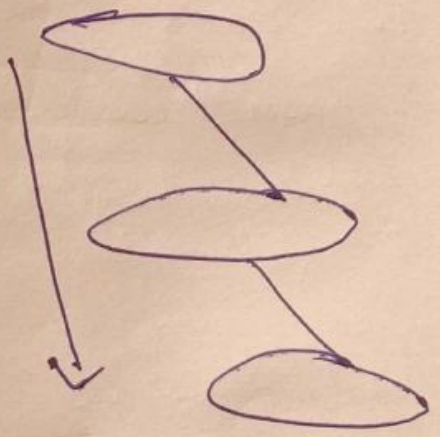
m-way search tree

4-way each node may have

4 children

3 keys





There must be some rules while creating m-way search trees.

B trees are m-way search trees with some rules

1. $\lceil m/2 \rceil$ children must be there

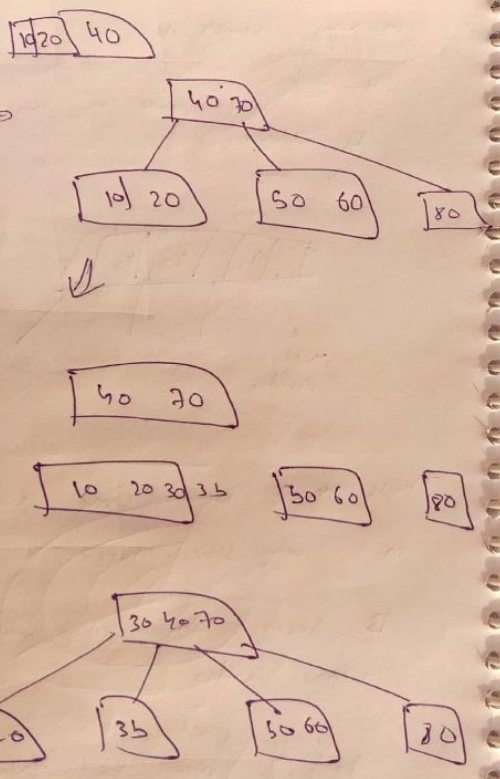
2. Root can have min 2 children

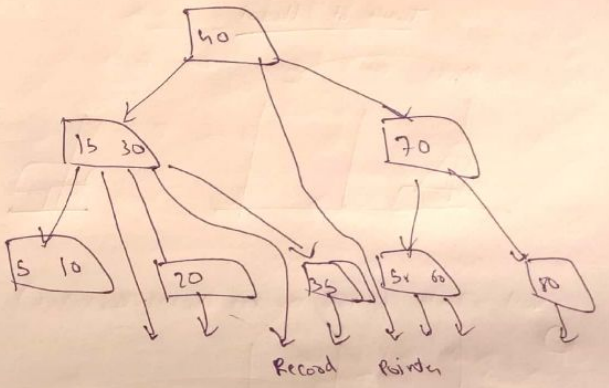
3. All leaf nodes must be at same level

4. Creation process is bottom up

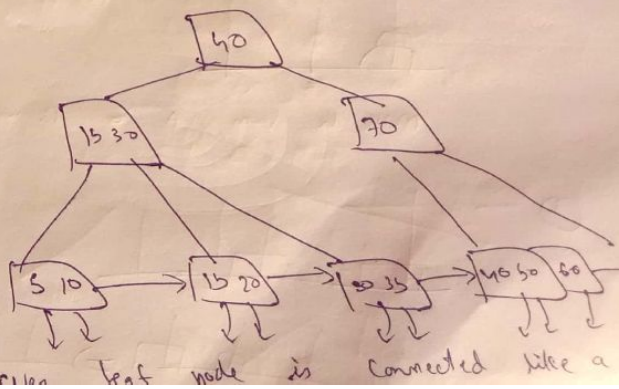
a. Keys 10 20 30 40 50 m=4

60 70 80 35 35



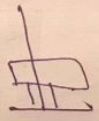


In BT tree, we don't have record pointer from every node. Only we have record pointer from leaf node



Every leaf node is connected like a

Tower of Hanoi



X

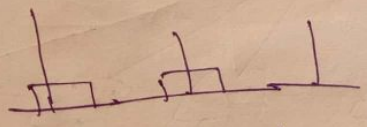
①

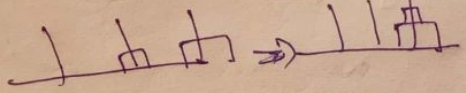
Move disc from A to C

②

Move disc from A to B using C

③





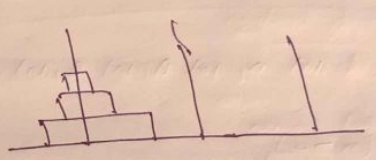
→

Move disc from A to B using C

Move disc from A to C

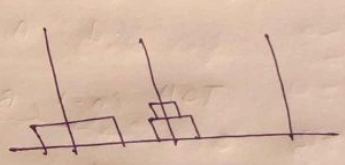
Move disc from B to C using A

Q. 3 disc

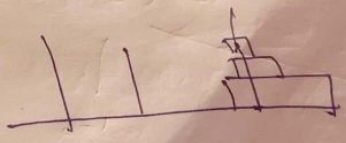


Move 2 disc from A to B using C

→



Move disc from A to C



Move 2 disc from B to C using A

A | B | C

Y move n-1 discs from A to B using C

Y move disc from A to C

Y move n-1 discs from B to C using A

```
void ToH ( int n, int A, int B, int C)
{
  if (n>0)
  {
    ToH(n-1, A, C, B);
    printf("move disc from %d to %d", A, C);
    ToH (n-1, B, A, C);
  }
}
```

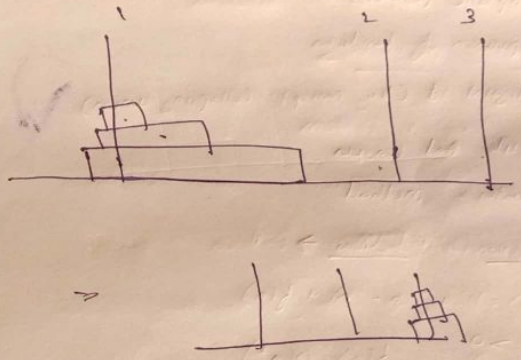
ToH (3, 1, 2, 3)

ToH (2, 1, 3, 2) | 1-3 | ToH (2, 1, 3)

ToH (1, 1, 2, 3) | 1-2 | ToH (1, 3, 1, 2) | ToH (1, 2, 3, 1) | 2-3 | ToH (1, 1, 2, 3)

(1-3) (1-2) (3-2) (1-3) (2-1)

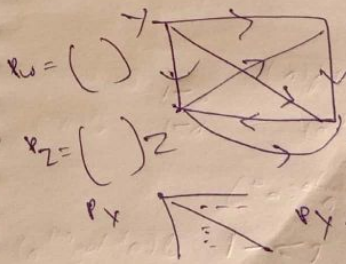
(2-3) (1-3)



Warshall Algorithm → O(n3)

used to find path matrix of a graph.

P[i,j] = P[i,j] or (P[i,k] and P[k,j])



	X	Y	Z	W
X	0	0	0	1
Y	1	0	1	1
Z	1	0	0	0
W	0	0	1	0

$P_0 = ( )$ $P_2 = ( )$ P_X $P_Y = ( )$

Contents Studied so far →

1. Algorithms
2. Frequency Count method
3. Time Functions
4. Asymptotic Notations
5. Comparison of functions
6. Disjoint Set (Key concept: Collapsing union)
7. Divide And Conquer.
8. Master's method

(a) Difference Functions →

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

$a > 0$
$b > 0$ $\quad f(n) = O(n^k)$
$k \geq 0$

$a \geq 1, \quad O(a^{n/b}\, n^k)$

$a = 1, \quad O(n f(n))$

$a < 1, \quad O(f(n))$

(b) Dividing Functions →

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

$f(n) = \Theta(n^k \log^p n)$ $\quad a \geq 1$, $b > 1$

(a) $\log_b a > k \quad O(n^{\log_b a})$

(b) $\log_b a = k$ i.e. $p > -1 \quad O(n^k \log^{p+1} n)$

(c) $\log_b a < k$ i.e. $p \geq 0 \quad O(n^k \log^p n)$

b.2 $p = -1$

$O(n^k \log\log n)$

b.3 $k < -1$

$\theta(n^k)$

(c) $\log_b a < k$

$p \geq 0 \quad O(n^k \log^p n)$

$p < 0 \quad n^k$

(9) Binary Search

(10) Heap Sort

- Complete binary tree - full binary tree
- Max Heap . Min Heap (nlogn)
- Heapify $O(n)$ (RTL)

(11) Priority Queue Max Heap Min Heap

(12) Merge Sort $O(m+n)$

Two way merge Sort $O(n \log n)$

(13) Insertion Sort $O(n^2)$

(14) Quick Sort Best case (nlogn) Worst case $O(n^2)$

(15) Strassen matrix multiplication (Divide and Conquer)
(7 multiplications $O(n^{2.81})$)

(16) Optimization Problems

16.1 Greedy Method

16.2 Dynamic Programming

16.3 Branch and Bound / Backtracking

17. Optimal and feasible solution.

18. Greedy methods →

a) Knapsack Problem

b) Job sequencing with Deadlines.

c) Optimal Merge Pattern $O(m+n)$

d) Huffman coding (Encoding / Decoding)

c) Minimum Cost Spanning trees

($|E|(|V|-1$ - no. of cycles)

d) Prim's and Kruskal.

$O(n^2)$ $O(EV)$ min $O(n \log n)$

min Heap

e) Single Source Shortest Path → Dijkstra

$O(n^2)$

19. Dynamic Programming →

Memoization & tabular method

① Multistage Graph $(O(n^2))$ ←

② All Pairs Shortest Path →
Floyd Warshall & Warshall $O(n^3)$

③ Matrix chain Multiplication. $O(n^3)$

$$c[i,j] = \min_{i \le k < j} \{ c[i,k] + c[k+1, j] \} + d_{i-1} * d_j * d_j$$

④ Bellman Ford → Single source Shortest Path

⑤ 0/1 Knapsack Problem

⑥ Optimal BST

⑦ Traveling Salesman Problem

⑧ Reliability Design

⑨ Longest Common Subsequence. 2^n to $O(m*n)$
$O(m \times n)$

Graph Traversals

- BFS & DFS preorder
  levelorder backedges
  State Space Tree
- Articulation pt. Constraints bounding fn

20. Back Tracking
21. Branch & Bound } Brute force Approach → DFS, BFS

22 Back Tracking

(a) N Queen Problem

(b) Sum of Subset

(c) Graph Colouring

(d) Hamiltonian Cycle

pendant.

23 Branch & Bound

(a) Job Sequencing with deadline

(b) 0/1 knapsack

(c) Travelling Sales person

24 NP Hard and NP Complete

Polynomial

linear n

binary log n

insertion n^2

merge n log n

matrix X n^3

Exponential (2^n)

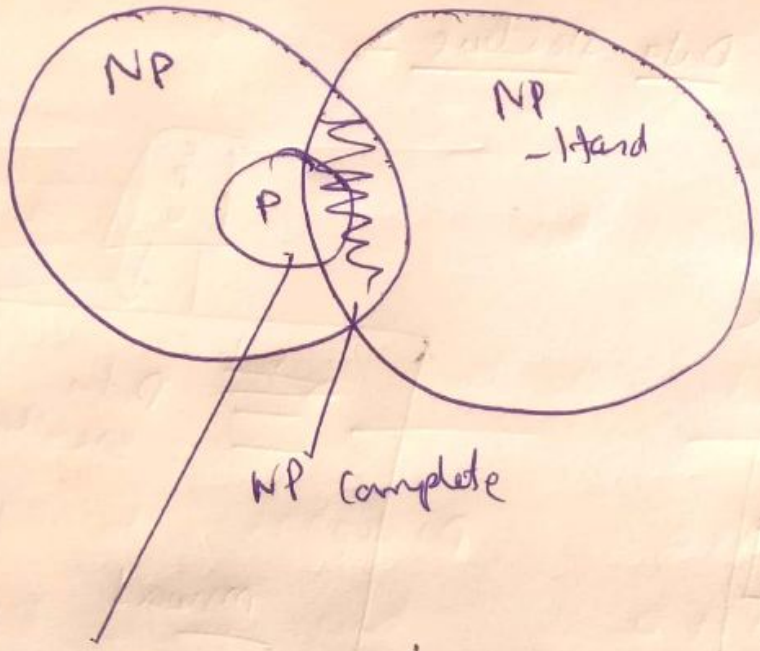
0/1

travelling

Sum of subset

Graph Colouring

Hamiltonian



P=NP (Cook's) Rule

25. NP Hard → Clique Decision Problem
Decision & Optimization Problem

26. Basic / Naive Algorithm

27. Knuth Morris Pratt Algo $O(m+n)$

28. Rabin Karp (Spurious Hits)

29 AVL

30 B B+

31. TOH