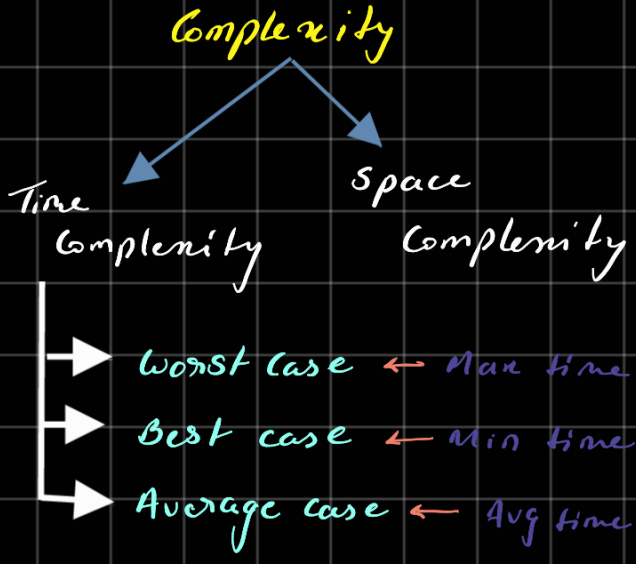
Design Analysis Algorithm ⟶

- **Algorithm**
  Procedure used for solving a problem or performing a computation.
  - Characteristics
    - Unambigous - Algorith should be straight, clear and to the point.
    - Feasibility - Should be feasible with the available resources.
    - Independent - Platform Independent.

- Statement → line of code.

- **Problem faced while creating Algorithm ⟶**
  1. Bug's can't be detected
  2. Non executable

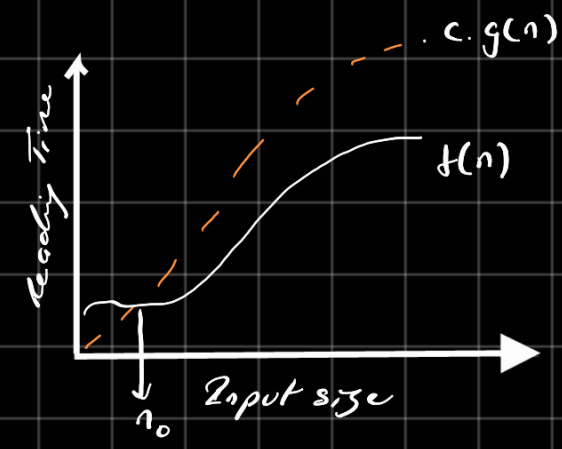


Asymptotic Analysis ⟶
- Space
- Speed

Execution time of an algorith depends on interaction set, processor speed hence this analysis is called Asymptotic Analysis.

Asymptotic Notation ⟶
Is used to describe the running time of an algorithm.
- big O
- big Theta (Θ)
- big Omega (Ω)

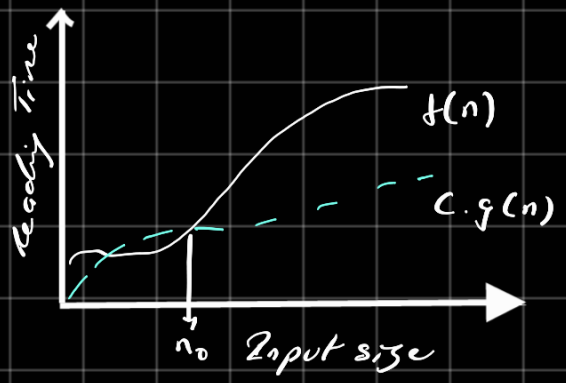
Big O →



$f(n) \leq c.g(n)$ for all $n > n_0$

$f(n)$ is $O\,g(n)$ if there exists c, n_0 state that $f(n) \leq c.g(n)$ for $n \geq n_0$

Omega notation (Ω) →



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

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

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

$f(n) \leq g(n)$ ✗

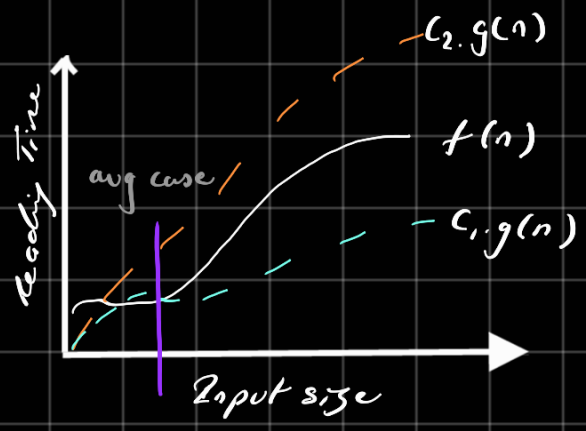
if there exists c, n_0 state that $f(n) \geq g(n)$ for all $n \geq n_0$

Theta notation (Θ) →

tidly bound

both upper and lower bound

avg, avg time bound



if there exists exists c_1, c_2, n_0 state that $c_1.g(n) \leq f(n) \leq c_2.g(n)$

Little O →

loose upper bound

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

Base case →

we want our program to end at some point. So we have to find a stopping point where the function will stop calling

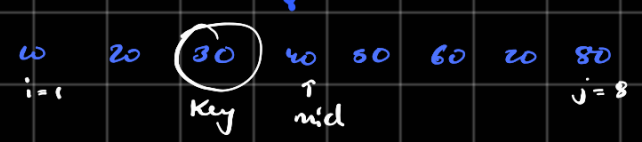
Little ω $\rightarrow$

loose lower bound

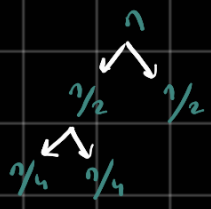
$f(n) \geq \omega\, g(n)$

Recurrance Relation $\rightarrow$ for expressing the run time of recurrance relation.

Binary search $\rightarrow$



$mid = \frac{i+j}{2}$



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

itself.

A base case is the condition where the last step is reached.

All cases should eventually reduce to base case.

without 'T'

$$T(n) = \begin{cases} a & \text{if } n=1 \\ 2T(n/2) + bn + c & \text{if } n>1 \end{cases}$$

base case.

worst case.

Substitute Method $\rightarrow$

$$\begin{aligned} T(n) &= T(n/2) + c \\ &= T(n/2^2) + 2c \\ &= T(n/2^3) + 3c \\ &\vdots \\ &= T(n/2^k) + kc \\ &= T(1) + kc \end{aligned}$$

$n = 2^k$

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

$\log n = k \log 2$

$$\begin{aligned} T(n) &= T(n/2) + c \\ T(n/2) &= T(n/4) + c \\ T(n/4) &= T(n/8) + c \end{aligned}$$

$$T(n) = 4T(n/2) + n^2 \quad -(1)$$

$$T(n/2) = 4T(n/4) + n^2/4 \quad -(2)$$

(2) $\rightarrow$ (1)

$$\begin{aligned} T(n) &= 4(4T(n/4) + n^2/4) + n^2 \\ &= 4^2 T(n/4) + 2n^2 \quad -(3) \end{aligned}$$

$$T(n/4) = 4T(n/8) + n^2/16 \quad -(4)$$

(4) $\rightarrow$ (3)

$$\begin{aligned} T(n) &= 4(4^2 T(n/8) + n^2/16) + 2n^2 \\ &= 4^3 T(n/8) + 3n^2 \end{aligned}$$

$$T(n) = 8T(n/2) + n^2 \quad -(1)$$

$$T(n/2) = 8T(n/4) + n^2/4 \quad -(2)$$

(2) $\rightarrow$ (1)

$$\begin{aligned} T(n) &= 8(8T(n/4) + n^2/4) + n^2 \\ &= 8^2 T(n/4) + 3n^2 \quad -(3) \end{aligned}$$

$$T(n/4) = 8T(n/8) + n^2/16 \quad -(4)$$

(4) $\rightarrow$ (3)

$$\begin{aligned} T(n) &= 8^2(8T(n/8) + n^2/16) + 3n^2 \\ &= 8^3 T(n/8) + 7n^2 \end{aligned}$$

$\Rightarrow 4^k T(n/2^k) + kn^2$

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

$n = 2^k$ —(i)

$\log n = k \log 2$ → taking on 'base 2'

$k = \log n$

$T(n) = 4^k T(n/2^k) + kn^2$

$= 4^k T(1) + n^2 \log n$

$\Rightarrow 4^{\log n} + n^2 \log n$

$\Rightarrow n^2 \log n$

$\Rightarrow 8^k T(n/2^k) + kn^2$

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

$n = 2^k$ —(i)

$\log n = k \log 2$

$k = \log n$

$T(n) = 8^k T(1) + n^2 \log n$

$= 8^k + n^2 \log n$

2^3

$n^3 + n^2 \log n$

$\Rightarrow n^3$

Master theorem ⟶

$$T(n) = aT(n/b) + f(n)$$

① $T(n) = O(n^{\log_b a - \epsilon})$ $\epsilon > 0$ — $-\epsilon$ to equate 'O' with 'Θ'. (e.g. O: Avg case)

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

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

② $T(n) = \Omega(n^{\log_b a + \epsilon})$ — $+\epsilon$ to equate Θ with Ω

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

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

③ $T(n) = \Theta(n^{\log_b a})$ $\quad f(n) = c.g(n)$

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

① ⟶ $T(n) = 8T(n/2) + \underbrace{n^2}_{f(n)}$

$\Rightarrow n^{\log_2 8}$

$\Rightarrow \underline{n^3}$ $g(n)$ $\quad f(n) \leq c.g(n)$

$T(n) = \Theta(n^3)$ $\quad O(n)$

② ⟶ $T(n) = 4T(n/2) + \underbrace{n^2}_{f(n)}$

$\Rightarrow n^{\log_2 4}$

$\Rightarrow \underline{n^2} \times \log n$ $\quad c.g(n) = f(n)$

$T(n) = \Theta(n^2 \log n)$ $\quad \Theta(n^{\log_b a} \log n)$

array →

static

declaration data type array name [size] ;

initialization data type array name [size] = { 1, 2, 3 ... }

Operation's →

- insertion
- deletion
- search

Types of Array →

- 1-D array
- 2-D array

Strack → LIFO

Is a linear data structure.

Overflow isFull()

Underflow isEmpty()

Linked list →

- Singly
- Doubly
- Circular

Queue → FIFO

- Enque
- Deque

Types of Queue →

- Circular Queue
- Priority Queue
- De queue (Double ended Queue)

Heap → ACBT

max min

(All must comple binary tree)

Heapify meathod →

Heap sort → Inplace

unstable

Hash function →

Red - Black →

Count sort →

Redix sort →

Bucket sort →

Masks Theorams →

$$T(n) = aT(n/b) + f(n)$$

$a \geq 1$

$b > 1$

$n^{\log_b^a}$

if $f(n) > n^{\log_b a} = \Theta(f(n))$

if $f(n) < n^{\log_b a} = \Theta(n^{\log_b^a})$

if $f(n) = n^{\log_b a} = \Theta(f(n) \cdot \log n)$

ex → $T(n) = 2T(n/2) + n$

$a = 2$
$b = 2$ } can solve by master theorem

$n \log$

$T(n) = 9T(n/3) + T(n^2)$

$a = 9$
$b = 3$ } ✓

$n^{\log_b a} \Rightarrow n^{\log_3 9}$

$\Rightarrow n^{\log_3 3^2}$

$\Rightarrow n^2$

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

Sobsitute Method →

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

$T(n) = T(n-1) + \log n$ —①

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

$= T(n-2) + \log(n-1)$ —②

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

1, 2, 3, 4, - - - . n-2, n-1, n.

$= T(n-3) + \log(n-2)$ — ③

$T(n-3) = T((n-3)-1) + \log(n-3)$

$= T(n-4) + \log(n-3)$ — ④

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

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

$\vdots$

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

Unit – 2

Divide and Conqure →

Quick sort →

10, 20, 30, 40, 50, 60

60, 50, 40, 30, 20, 10

Merge sort →

inplace, unstable

Greedy Algorithm →

i) Knapsack Problem →

Job sequencing →

	J_1	J_2	J_3	J_4
Profit →	50	15	10	25
Deadline →	2	1	2	1

Duration for every Job → 1

→ feasible solution } the one with most profit.

50 + 10 = 60

→ optimal solution } the one with most profit and upcoming deadline.

$25 + 50 = 75$

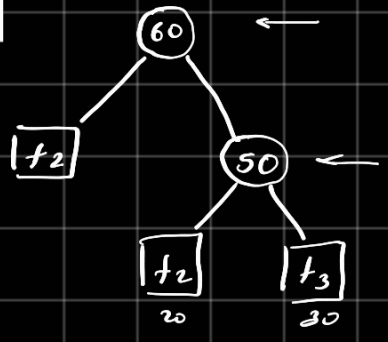
① Arrange in Decreasing order of profit

	J_1	J_4	J_2	J_3
Profit	50	25	15	10
Deadline	2	1	1	2

Optimal Merge Pattern

t_1 t_2 t_3

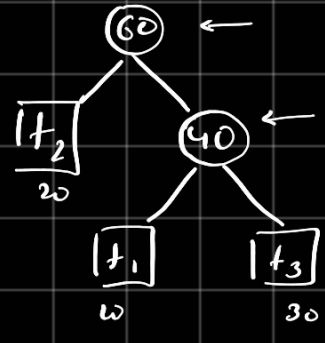
10 20 30



Total moment ⇒ 50 + 60

⇒ 110

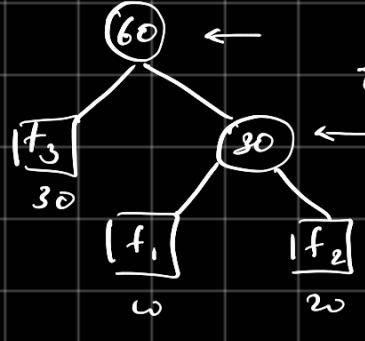
* Most optimal



Total moment ⇒ 60 + 40

⇒ 100

* feasable solution



Total moment ⇒ 60 + 30

⇒ 90

* feasable solution.

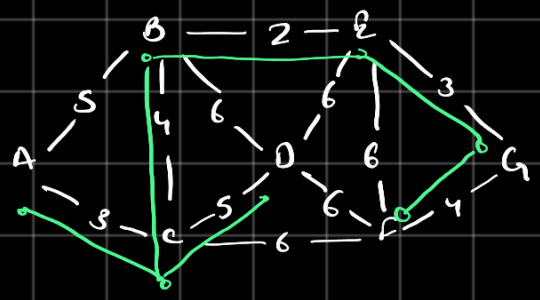
t_1	t_2	t_3	t_4	t_5	t_6
2	4	6	8	10	12

Spanning Tree →

Kruskikal Algorithm → Non continous

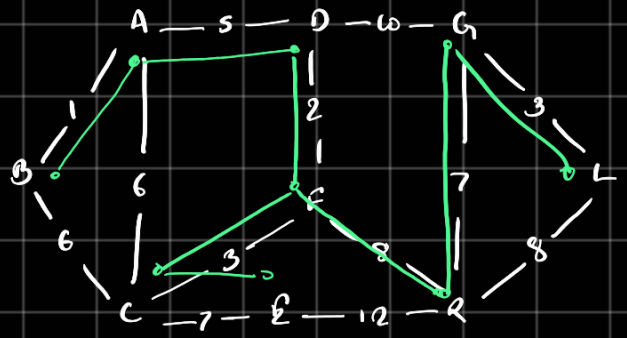
Prim's Algorithm → In flow

Kruskikal Algorithm →



Min cost ⇒ 21

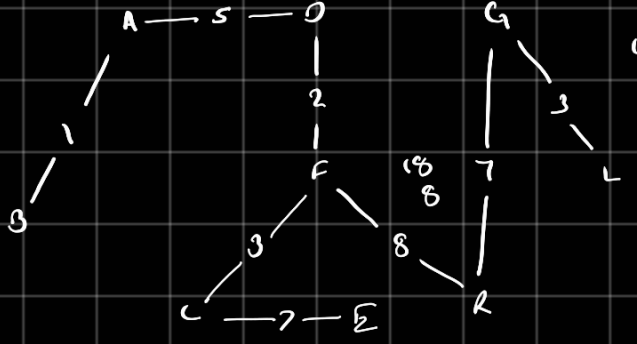
Prim's Algorithm →



6 ✓

7 —

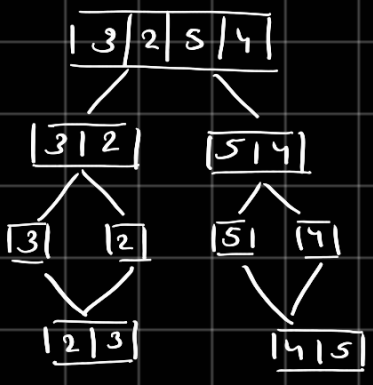
Using Kruskikal



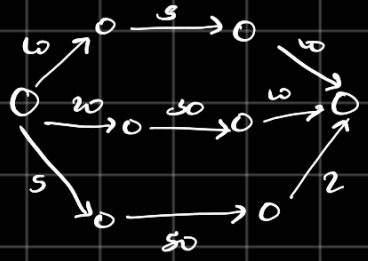
10

36

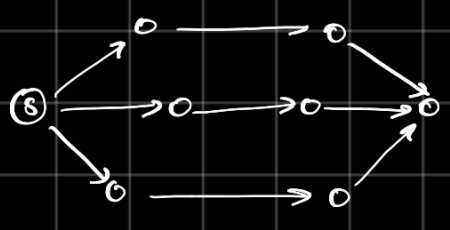
Divide and conquer



Greedy Approach



Dynamic Programming

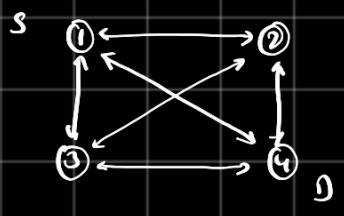


S – 1 – 2 – D =

S – 3 – 4 – D =

S – 5 – 6 – D =

Travelling Salesman Problem →



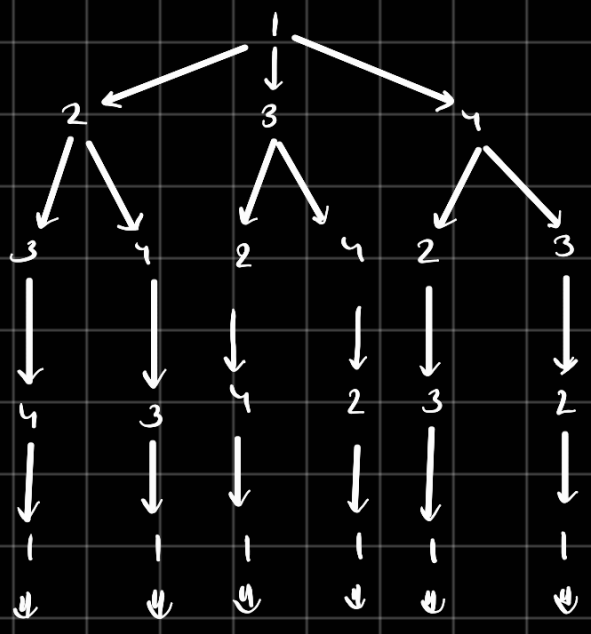
	1	2	3	4
1	0	10	15	20
2	5	0	25	10
3	15	30	0	5
4	15	10	20	0

Greedy

1 → 10
2 → 10
4 → 20
3 → 15
1

55

Broot force method →



NP complete problem

0/1 Knapsak Problem →

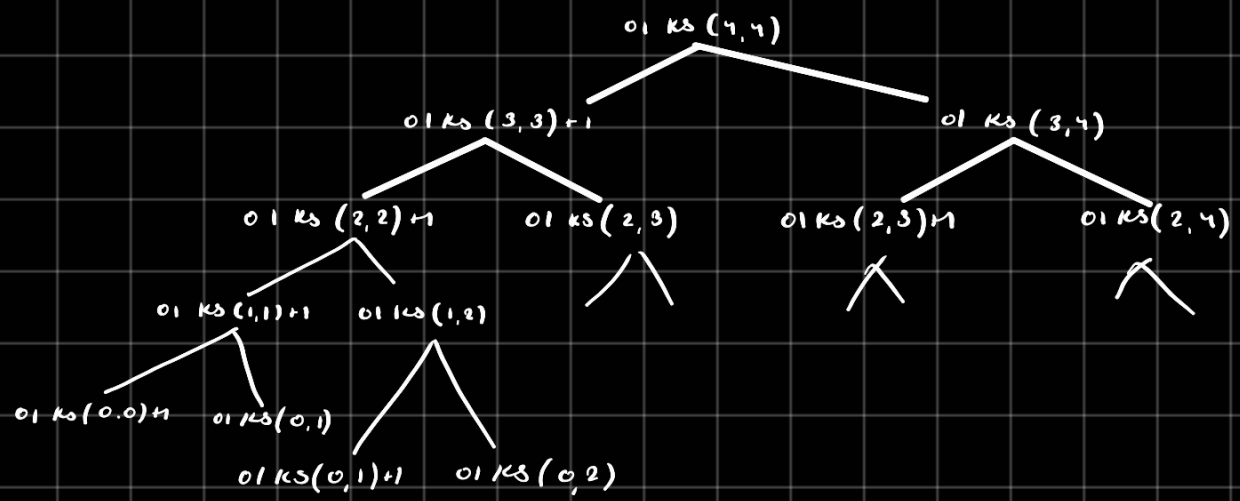
	Obj 1	Obj 2	Obj 3
weight	2	4	8
Profit	20	25	60

n = object
M = Total weight
Pn = Profit
wn = object weight

0 → object not selected
1 → object accepted

$$\begin{cases} 0/1\ KS(n-1), (M-w_n) + P_n \\ 0/1\ KS(n-1), M \end{cases}$$

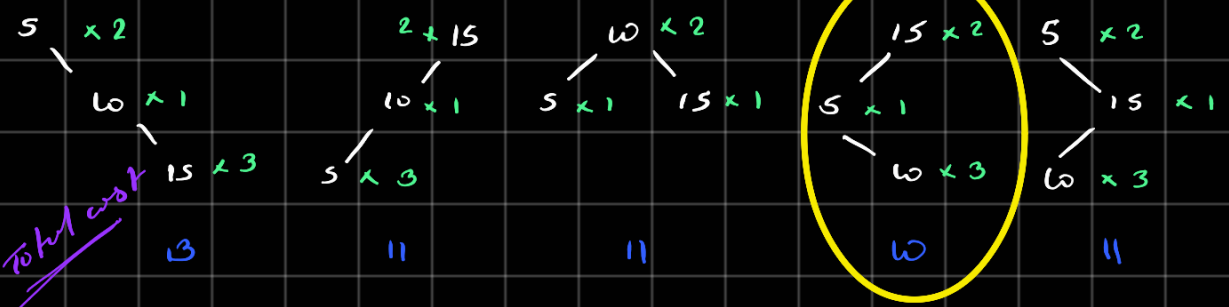
	Obj 1	Obj 2	Obj 3	Obj 4
Weight	1	1	1	1
Profit	1	1	1	1



- Optimal Binary Search Tree →

  key – 5, 10, 15

  frequency – 2, 1, 3



Most cost efficient.

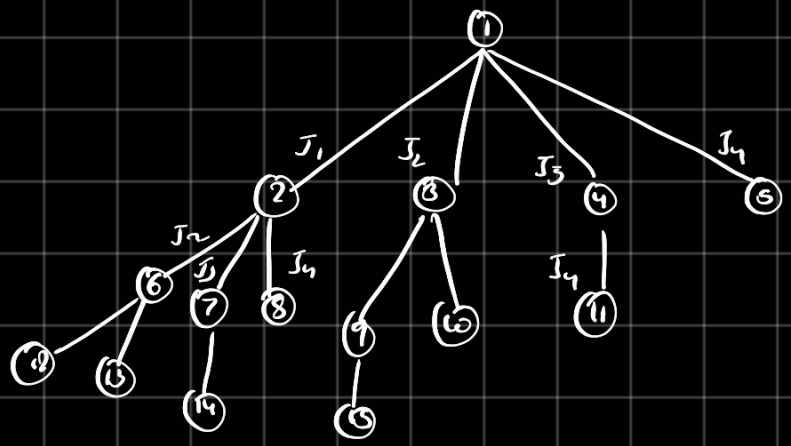
Backtracking →

Branch and Bound →

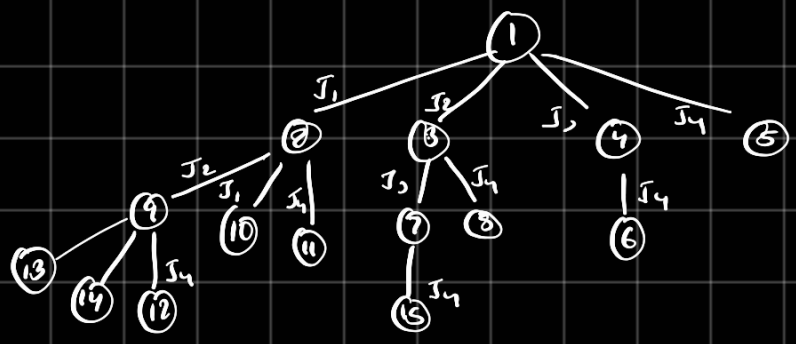
Job sequence	J_1	J_2	J_3	J_4
Profit				
Deadline				

State space tree →

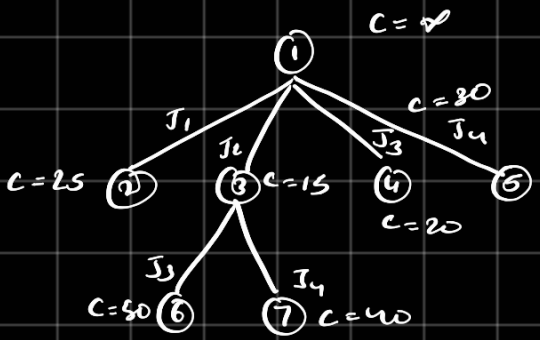
(1) Queue →

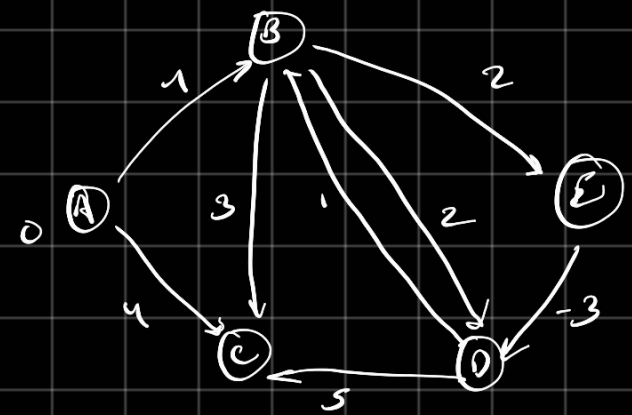


(2) Stack → (DFS)



(3) Least Cost (BB) →





A	B	C	D	E
0	-	-	-	-
0	(1)	4	-	-
0	1	2	1	(1)
0	1	2	(2)	1