04/01/19. # Design and Analysis of Algorithms

1. What is algorithm?
2. Analysis
3. Design

Introduction to Algorithms

Instructors

Origin:

'Algor' — Latin

Al-khawarizmi — systematically studied linear / eqn

↓ Father of Algebra

Latin translation of name →

Sorting problem: Input: Sequence of n

Output: A permutation reordering $a_1, a_2 \ldots a_n$

ST $a_1' < a_2' \ldots a_n'$

Resources Alg. requires

CPU time — Memory

1. Input
2. Output
3. Finiteness property → computational mode

Assumption under RAM model:

1. Under hypothetical Theory of computation all operations take constant time. → but not possible as '*' > '+'.
2. Execution is sequential, no parallelism
   ↳ infinite memory space.
3. Random access to memory
4. Infinite supply of memory.

RAM –

i/p → [Alg] → o/p

[Finite time]

Finite space irrespective of memory location.

Random access means any element part of it takes same time to access.

Ex: a[2], a[100], a[100000]

↓

operation all take same time

→ Finite means – Naot ~~steps~~ ~~finite~~ finite performed under ram model of computational model.

→ Looping operations not considered as single but repetition of n operations.

4. Definiteness: No ambiguity in steps.

5. Effectiveness:

→ Should not combine all steps

→ Operation performed in finite time and memory.

07/01/18.

Asymptotic Analysis:

If 'n' is input size,

$$m = \lfloor \log_2 n \rfloor + 1$$

input – in binary

T(n) – running time.

i.e., $n = 2^k$, $K \geq 0$

↳ since L] - can be taken

$$n = 2^{m-1}$$

If $T(n) = n-2$: $\boxed{T(m) = 2^{m-1} - 2}$

Order of growth, $O(n) = 2^n$ – exponential time alg.

Polynomial – good algorithm

Inputs (n) – n inputs → max. size
Each occupies $(\log_2(a_i)+1)$ of all inputs (a_i – input)
(binary rep.)

Total space = $n \times (\log_2(a_i)+1)$

→ Any base >1, $\log c = [\log_2 A] + 1$

$T(n) = n^2$ (if)

$T(n) = (cn)^2 = c^2 n^2$ – Polynomial (Quadratic)

→ **Problem complexity ≤ Algorithm complexity.**

Ex. For a sorting prob.,

prob. comp. = n (checking all elements)

algos. comp = n^2 (bubble sort - suppose)

Sorting algorithms — 1. Comparision based

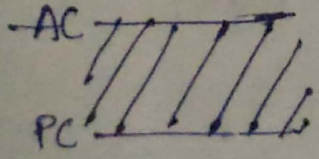
2. No comparision.
(Counting sort, Radix sort etc.)

→ It is proved that a comparision based sorting problem cannot be done less than nlogn.

Prob. complexity for comparision based = nlogn

→ If PC = AC then best algorithm

If there is a diff between PC and AC, scope for better algo.



Outcomes:

→ Running time, Time complexity, Space complexity.

→ Understand algo. design techniques

→ Understand how complex a problem is.

→ Classify problems based on complexity.

Asymptotic Notations:

1. O (Big O):

$O(g(n)) = \{f(n) : \text{OOG } f(n) \leq \text{OOG } g(n)\}$

$O(n^2) = \{n, n^2, n\log n, 1, 2, 92n^2+7n+40\}$

$3^{n+1} = O(3^n)$ ✗ $\quad 2^{n+1} = O(2^n)$ ✓ $\quad 2^n = O(n!)$ ✓

~~$3^{n+1} = O(3^n)$~~ $\quad \sum_{i=1}^{n} i^2 = O(n^3)$ ✓ $\quad n! = O(n^n)$ ✓

$(n+1)! = O(n!)$ ✗ $\quad (\log n)^{27356} = O(n^{0.001})$

$\log n! = O(n \log n)$ ✓ $\quad$ Polylogarithmic $\quad$ Polynomial

$n^{96} = O(3^n)$ ✓ (good)

$n^{96} = O((0.3)^n)$ ✗

OOG: Exponential > Polynomial > Polylogarithmic

08/01/19

Analysis of algorithm – Asymptotic analysis

Order of growth – always function of input size

2. Ω (Big Ω) (omega)

$\Omega(g(n)) = \{f(n) : \text{OOG } f(n) \geq \text{OOG } g(n)\}$

Ex: $\Omega(n^2) = \{n^3, 6n^3+7n+4, 3^n, 2^{n+5}, n!, n^n, \frac{n(n+1)}{2}, \sum_{i=1}^{n} i^2 \ldots\}$

1) $\Omega(2^n) = 2^{n+4}$ 2) $2^{n-1} = \Omega(2^n)$ 3) $2^n = \Omega(n!)$ – ✗ correct for $n \leq 3$

4) $n^n = \Omega(n!)$

5) $\log n! = \Omega(n \log n)$ — Approximation:

$\log n! \approx \frac{1}{2}\log 2\pi n + n\log n - n\log e$ ↑ $\boxed{n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n}$ ↑

Both are same order of growth — Stirling approximation formula

6) $n^{555} \cdot \Omega(2^n)$ – False ✗

Polynomial ↳ Exponential

7) $n^{0.001} \cdot \Omega(\log n)^{999}$ – ✓

3. Θ (Theta)

$\Theta(g(n)) = \{f(n) : OOG\ f(n) = OOG\ g(n)\}$

Ex: 1. $\Theta(n^2) = \{n^2+6n+7,\ \sum_{i=1}^{n} i,\ (n-1)^2,\ (3n+2)^2,\ \frac{n(n+1)}{2},\ n^2+n+1, n\}$ $4^{\log_2 n}$

2. $2^{n+2} = \Theta(2^n)$ ✓

3. $2^{n-1} = \Theta(2^n)$ ✓

4. $2^{2n-1} = \Theta(2^n)$ ✗

5. $(n+1)! = \Theta(n!)$ ✗

6. $\log_{100} n = \Theta(\log n)$ – ✓

7. $\log n! = \Theta(n \log n)$ ✓

4. O (Little o / Small o):

$o(g(n)) = \{f(n) : OOG\ f(n) < OOG\ g(n)\}$

Ex: $o(n^2) = \{$ ~~$n(n+1)$~~ $\frac{n(n+1)}{2}$ ✗

$o(n^2) = 3n+6\sqrt{n}$ ✓

$o(n^2) = \{n\log n, (\log n)^2, \sum_{i=1}^{n} 1, \sum_{i=1}^{n} i \ldots\}$

~~$2^{n-4} = o(2^n)$~~ ✗

$(\log n)^{75} = o(n^{0.2})$ ✓

$2^n = o(n!)$ ✓

$n! = o(n^n)$ ✓

$n^{9999} = o(1.2^n)$ ✓ $(0.2)^n$ ✗

5. ω (Little ω)

$\omega(g(n)) = \{f(n) : OOG\ f(n) > OOG\ g(n)\}$

Ex: $\omega(n^2) = \{n^3, \sum_{i=1}^{n} i^2, \frac{n^2(n+1)}{2}, n^2\sqrt{n}, \{\sum_{i=1}^{n} i\log i \ldots\}$

$(n+1)! = \omega(n!)$; $n^{\log n} = \omega(n^{500})$ ✓ $n! = \omega(2^n)$ ✓

$n^{9999} = \omega(3.2^n)$ ✗

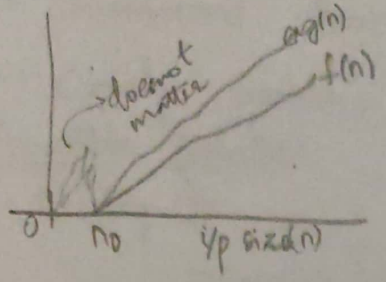
Formal definitions of asymptotic notations:

O: $f(n) = O(g(n))$ iff $\exists$ two constants 'c' and 'n_0' such that $f(n) \leq c * g(n)$ $\forall$ $n \geq n_0$

$n \in Z$

n-input size

↳ always integer



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

Proof: $\frac{1}{2}n^2 + \frac{1}{2}n \leq \frac{1}{c} * n^2 + 0$, $\forall n \geq \frac{0}{n_0}$

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

$+ \quad \frac{1}{2}n \leq \frac{1}{2}n^2$

$\frac{1}{2}n^2 + \frac{1}{2}n \leq 1n^2$

$c = 1; \; n_0 = 0$

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

2. $7n^4 + 90n^3 + 200n^2 + 900n + 1000 = O(n^4)$

Proof: $7n^4 + 90n^3 + 200n^2 + 900n + 1000 = \_\_\_ n^4$

$7n^4 = 7n^4$

$+ \quad 90n^3 \leq 90n^4$

$+ \quad 200n^2 \leq 200n^4$

$+ \quad 900n \leq 900n^4$

$+ \quad 1000 \leq 1000n^4$

$7n^4 + 90n^3 + 200n^2 + 900n + 1000 \leq 2197n^4, \; \forall n \geq 0$

Proved.

Generalising:

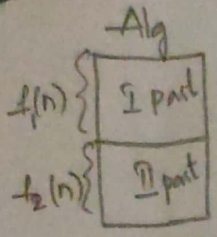
Theorem: If $f(n) = \sum_{i=0}^{m} a_i n^i$, then $f(n) = O(n^m)$

Proof: $f(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0 = \sum_{i=0}^{m} a_i n^i$

$\leq \sum_{i=0}^{m} |a_i| n^i$

$\leq n^m \sum_{i=0}^{m} |a_i| n^{i-m}$

11/01/19.



$f_1(n) = O(g_1(n))$

$f_2(n) = O(g_2(n))$

Total time complexity = $f_1(n) + f_2(n)$

= ~~$O(g_1(n))$~~ *

= $O(\max(g_1(n), g_2(n)))$

Theorem:

Statement: If $f_1(n) = O(g_1(n))$ & $f_2(n) = O(g_2(n))$, then prove that $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$.

Proof: Prove $f_1(n) + f_2(n) \leq \square * O(\max(g_1(n), g_2(n)))$

↳ constant $\quad \forall n \geq \square$ ↳ constant

$f_1(n) = O(g_1(n)) \Rightarrow f_1(n) \leq c_1 g_1(n) \quad \forall n \geq n_1$ — ①

$f_2(n) = O(g_2(n)) \Rightarrow f_2(n) \leq c_2 * g_2(n) \quad \forall n \geq n_2$ — ②

Equation ① + Equation ②,

$f_1(n) + f_2(n) \leq c_1 * g_1(n) + c_2 * g_2(n), \quad \forall n \geq \max(n_1, n_2)$

$c_3 = \max(c_1, c_2)$

$f_1(n) + f_2(n) \leq c_3(g_1(n) + g_2(n))$

If a, b are real numbers, $a + b \leq \max(a,b) + \max(a,b)$

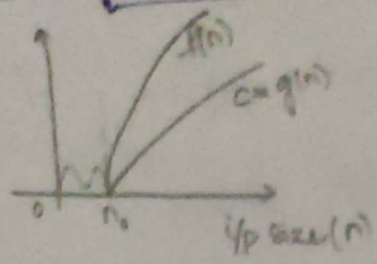
$f_1(n) + f_2(n) \leq c_3(2\max(g_1(n), g_2(n)))$

$f_1(n) + f_2(n) \leq \underbrace{2c_3}_{c}(\max(g_1(n), g_2(n)))$

$f_1(n) + f_2(n) \leq O(\max(g_1(n), g_2(n)))$

$n \geq \max(n_1, n_2)$

Ω: $f(n) = \Omega(g(n))$ iff $f(n) \geq c * g(n)$, $\forall n \geq n_0$



⟶ It only gives lowest bound (not tight)

Ex: 1. Show that $\frac{n(n+1)}{2} = \Omega(n^2)$

$$\frac{n(n+1)}{2} \geq \frac{1}{2}n^2, \quad n \geq 0$$

2. S.T $\frac{n(n-1)}{2} = \Omega(n^2)$

$$\frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{4}n^2$$

$$\frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \left(\frac{1}{2}n\right)\left(\frac{1}{2}n\right) \quad \forall n \geq 2$$

$$\frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{4}n^2$$

3. S.T $6n^2 - 5n - 20 = \Omega(n^2)$

~~$6n^2 \geq n^2$~~

~~$5n \geq 0$~~ $5n \leq 5n^2$

$-5n \geq -5n^2$

~~$-20 \geq 0$~~ $-20 \leq 20n^2$

$-20 \geq -20n^2$

$6n^2 - 5n - 20 \geq 5n^2$, $\forall n \geq 9$ ($c = 5$)

$n^2 - 5n - 20 \geq 0$

$n \geq \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

$n \geq \frac{5 \pm \sqrt{25 - 4(1)(-20)}}{2(1)} \geq 5 \pm 1$

4. S.T $\log n! = \Omega(n \log n)$

$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

$\log n!$

$n! < n^n$

$\log 1 + \log 2 + \ldots + \log(n/2) + \ldots + \log n > = \log\left(\frac{n}{2}\right) + \log\left(\frac{n}{2}+1\right) \ldots \log(n)$

(5) If $f(n) = a_m n^m + a_{m-1} \cdot n^{m-1} + \ldots + a_1 n + a_0$, then $f(n) = \Omega(n^m)$.

↳ $a_m > 0$

$f(n) = \Omega(n^m)$ — $a_m > 0$

$f(n) = \sum_{i=0}^{m} a_i \cdot n^i$

Θ: $\boxed{f(n) = \Theta(g(n))}$

To show $a = b$, $a \geq b$ and $a \leq b$.

show

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

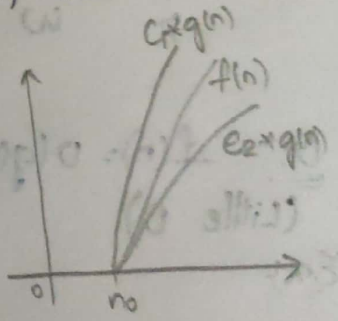
$f(n) = O(g(n))$ $\quad f(n) = \Omega(g(n))$

$f(n) \geq c_2 * g(n)$, $\forall n \geq n_2$

$f(n) \leq c_1 * g(n)$, $\forall n \geq n_1$

$f(n) = 0$ $\quad \boxed{c_2 g(n) \leq f(n) \leq c_1 (g(n)) \quad \forall n \geq n_0}$

If c_1, c_2, n_0 can be found, then $f(n)$ and $g(n)$ are of same order



Ex: 1. S.T $\frac{n(n-1)}{2} = \Theta(g(n)) = \Theta(n^2)$

$c_2 \times n^2 \leq \frac{1}{2}n^2 - \frac{1}{2}n \leq c_1 \times n^2$

$\frac{1}{4}n^2 \leq \frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2$

$n \geq 2$ $\qquad$ $n \geq 0$ $\Longrightarrow n_0 = 2 \Rightarrow n \geq 2$.

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

2. $f(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0 \longrightarrow \boxed{f(n) = \Theta(n^m)}$

only if $a_m > 0$.

3. Show that $\log n! = \Theta(n \log n)$

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

$\Rightarrow \log 1 + \log 2 + \cdots + \log n \le \log n + \log n + \log n + \cdots + \log n$

$\Rightarrow \boxed{\log n! \le n \log n} \Rightarrow \log n! = O(n \log n) \quad n_0 \ge 1$

$c_2 * g(n) \le \log n! \le c_1 * g(n)$

$\frac{1}{2} * n \log n \le \log n! \le 1 * n \log n$

$n_0 \ge 1 \qquad n_0 \ge 1$

$\Rightarrow \log n! = \Theta(n \log n)$

Scribing

a	b	f(n) g(n)	
$\le$	O	— ①	$f(n) \le c * g(n) \quad \forall n \ge n_0$
$\ge$	Ω	— ②	$f(n) \ge c * g(n) \quad \forall n \ge n_0$
$=$	Θ	— ③	$c_2 g(n) \le f(n) \le c_1 g(n) \quad \forall n \ge n_0$
$<$	o		
$>$	ω		

o: $f(n) = o(g(n))$ iff $\boxed{\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0}$

(Little o)

Ex:

1. Show that $4n^3 + 8n + 1 = o(n^5)$

$$\lim_{n \to \infty} \frac{4n^3 + 8n + 1}{n^5} = 0.$$

2. Show that $n! = o(2^n)$

$$\lim_{n \to \infty} \frac{n!}{2^n} *$$

$\boxed{2^n = o(n!)}$ ✓

$\lim_{n \to \infty} \frac{2^n}{n!} = 0.$ ✓

3. Show that $\boxed{n! = o(n^n)}$ ✓

4. Show that $(\log n)^{200} = O(n^{0.001})$

↳ polynomial

polylogarithm

NOTE:- $\lim_{n\to\infty} \frac{(\log n)^a}{n^b}$ where $b>0$ is zero. $\boxed{(\log n)^a = o(n^b)}$

$\lim_{n\to\infty} \frac{n^a}{b^n}$ where $b>1$ is zero. $\boxed{n^a = o(b^n)}\ \forall\ b>1$

ω: (Little omega)

$f(n) = \omega(g(n))$ iff $\boxed{\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty}$

Ex:

1. $3n^3+5 = \omega(n^2)$

$\lim_{n\to\infty} \frac{3n^3+5}{n^2} = \infty$.

Other notations:

1. $f(n) = O(f(n))$ ✓ 2. $f(n) = O(g(n))$ then $g(n) = \frac{\Omega(f(n))}{\omega}$ ✓ X

Ω ✓
Θ ✓
O X
ω X

Ω O, ω, Θ — Reflexive, transitive

O, ω

BST — O(height of BST)

AVL — Θ(height of AVL)

Height worst case: $\Theta(n)$ in BST

$\frac{\Theta(\log n)}{O(\log n)}$ in AVL.

Transitive — Θ, O, Ω, o, ω

Reflexive — Θ, O, Ω

Symmetry — Θ.

Transpose symmetry — $f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$

$f(n) = o(g(n))$ iff $g(n) = \omega(f(n))$

→ If $f(n) = O(h(n))$; $g(n) = O(h(n))$ → $f(n) + g(n) = O(h(n))$

$\log(an) = O(\log_2(n))$ $a \neq 1$

Analysis of Algorithms:

Algorithm Linear Search (A, n, key)

1. i ← 1
2. while i ≤ n and A[i] ≠ key
3. i ← i+1
4. if i ≤ n
5. print "Key found"
6. else
7. print "Key not found"

Steps:

→ Decide the parameter indicating input size.

2. Basic Operation: (Main operation in the program)
3. T(n) – no.of operations basic operation is performed.

Here, n – input size

Basic operation: Comparision.

→ Based on the input size, T(n) can't be found since it depends on the values of the input. (order in which the values are arranged.) (instance)

If T(n) is only function of input size, it is direct

If not ! 1. Min (Best Case)

2. Max ~~Generally in Σ, find OOG~~ (Worst case)

3. Average

4. Generally in Σ, find OOG

5. Represent running time using asymptotic notation.

For recurrence: T(n) - changes
(Same procedure)

Solution for Linear Search algo:

1. n
2. Comparision
3. T(n) = depends on instance and input
4. Min (Best Case Running time):

$T(n) = 1 \Rightarrow T(n) = n^0$

OOG of $T(n) = n^0$

5. Best case running time = $O(1) / O(\log n) \ldots$

$= \Omega(1) \ldots$

$= \Theta(1) \ldots$

4. Max (Worst Case Running time):

Key present at last (or) not present.

$T(n) = n$

5. Worst Case Running time = $O(n) / O(n^2) \ldots$

$= \Omega(n) / \Omega(1) \ldots$

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

=

4. Average Case Running time:

Type of input matters.

Assuming uniform distribution on input.

1. Search is successful with probability p $(0 \le p \le 1)$
2. Each position is equally likely,

X: Running time: 1, 2, 3 . . . k, . . . n.

Probability at position ?: $P(\frac{1}{n})$ $P(\frac{1}{n})$ . . . . $(\frac{p}{n})$

Failure case: $(1-p)n$

22/01/19

	1	2	3	4	5	...	i	...	n	n+1 not present
X:	$\frac{p}{n}$	$\frac{p}{n}$							$\frac{p}{n}$	$1-p$

$$\text{Expectation}(X) = \sum_{i=1}^{n} i\frac{p}{n} + (1-p)(n)$$

$$= \frac{p}{n}\left(\frac{n(n+1)}{2}\right) + (1-p)n$$

$$= \frac{pn}{2} + \frac{p}{2} + n - pn$$

$$= \frac{p}{2} + n - \frac{pn}{2}$$

$$E(X) = \frac{p(n+1)}{2} + n - \frac{pn}{2} = \frac{p(1-n)}{2} + n$$

Average case DDG = $O(n)$

$= \Theta(n)$

$= \Omega(n)$

→ Average case RT is close to worst case in general.

NOTE: Avg. case RT ≅ Best case RT (Quicksort)

→ Best Case RT: Ω preferred

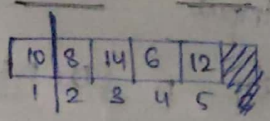
└ since avg. time (Ω) = BC RT(Ω) = WC RT(Ω)

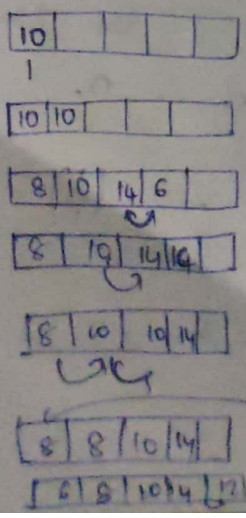
→ Worst Case RT: O preferred

└ true for avg. case and best case.

Sorting Algorithms:

1. Insertion sort:





Algorithm: (A, n)

1. for j ← 2 to n
2. key = a[j]
3. i ← j-1
4. while i ≥ 1 and a[i] > key
5. a[i+1] ← A[i]
6. i ← i-1
7. A[i+1] = key

Correctness:

↳ To prove that the algorithm is correct.

Loop invariant method:

✓Something is not ~~keep~~ changing for loop.

↳ should be found. (irrespective of loop)

- A[1 . . . . j-1] – sorted
- A[1] ≤ A[2] ≤ . . . . ≤ A[j-1]

↳ To prove this 3 steps (Similar to mathematical induction)

1. Initialization
2. Maintanence
3. Termination.

1. Initialisation : Before entering loop, checking sorted or not

j = 2 – loop starts

A[1 . . 2-1] – sorted

Before loop, only one element ⟹ sorted.

2. Maintanence:
(Hypothesis + Induction)

Before k^{th} iteration, if loop statement is true (sort)

then after one iteration, the elements are in sorted order.

k^{th} iteration: A[1 . . . . k-1] – sorted

⇒ $(k+1)^{th}$ iteration: A[1 . . . . k+1] – also sorted. ← to be proved.

From the algorithm, loop invariant is maintained.

3: Termination: (loop terminates – is checked)

For j = n+1

A[1 . . . . . (n+1)-1]

A[1 . . . . . n] = sorted.

→ Resources: Time
Space.

Running Time:

1. n — input
2. A[i] > key — basic operation
   └ Comparision
3. T(n)
   └ only depends on 'n' or depends on actual values.

④ Best-Case: $T(n) = \sum_{j=2}^{n} 1 = n-2+1 = n-1$

↳ elements are in sorted order

OOG $T(n) = n$

⑤ T(n) - BC RT of IS = $O(n)$
= $\Omega(n)$
= $\Theta(n)$

④ Worst-Case:

Descending order:

For j^{th} values, j-1 comparisions performed.

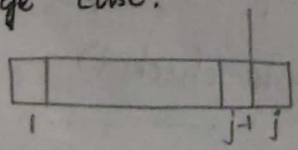
$$T(n) = \sum_{j=2}^{n}(j-1) = \frac{n(n-1)}{2}$$

OOG $T(n) = n^2$

25/01/19

RT of IS = $O(n^2)$ ✓
$O(\sqrt{n})$ ×
$\Omega(n)$ ✓
$\Omega(n^3)$ ×
$\Theta(n^2)$ ×

⑤ Average case:



Element in 'j' has equal chances to be moved to 1, 2 . . . j-1, j.

Pos. — Prob.

Probability to be at 'j' position is 1/j.

j-1 — 1/j ...

j-2 : 1/j

1 — 1/j

A(j) – avg. no. of comparisions req. to move from position 'j' to first (j-1) elements (sorted).

1	2	3	4	.....	j-1
1/j	1/j	1/j	1/j	.....	1/j

$$E[X] = \sum_{i=1}^{j} i \times 1/j = \frac{j(j+1)}{2} \times \frac{1}{j} = \frac{j+1}{2}$$

Avg. no. of comparisions

$$\sum_{j=2}^{n} \frac{j+1}{2} = \frac{1}{2}\left(\frac{n(n+1)}{2}\right) + \frac{1}{2}(n-2+1)$$

$$\therefore BOG = \Theta(n^2)$$

$$\cdot OOG = n^2$$

OOG: Avg. case = n^2

$\Rightarrow$ Avg. case RT = Worst case RT

Alg:

1. for j ∈ 2 to n
2. key ← A[i]
3. i ← j-1
4. while i ≥ 1 and A[i] > key
5. A[i+1] ← A[i]
6. i ← i-1
7. A[i+1] ← key

10	6	20	18
1	2	3	4

If (i<j) and A[i] > A[j]

then, inversion pair

(1,2), (3,4) – Inversion pairs

Min. no. of inversions: Zero (Sorted)

Max. no. of inversions: $^nC_2$ (Descending order)

40	30	20	10
1	2	3	4

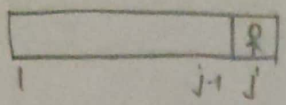
Relation b/w RT of Is and no. of inversions:

T(n) ↗ f(n) ↗

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

$$= O(\max\{n, f(n)\})$$

In insertion sort, instead of comparing an element with the sorted part linearly, binary search can be done.



$\sum_{j=2}^{n} \log(j-1)$

$= \log(n-1)!$ $\quad \log(n-1)! = O((n+1)^{n-1})$

$= O(n \log n)$

GCD:

$x, y \in \mathbb{Z}$

$gcd(x,y) = gcd(|x|, |y|)$

$gcd(x,0) = x$

Euclids algorithm (300 BC):

↓

GCD finding

→ It is based on:

$$\boxed{gcd(x,y) = gcd(y, x \bmod y)}$$

$gcd(x,0) = x$

Dividend

Divisor ← $y)x$ $(\frac{x}{y}$

$x \bmod y$

$x = y(\frac{x}{y}) + x \bmod y$

$$\boxed{x \bmod y = x - y(\frac{x}{y})}$$

Algorithm:

1. if $y > 0$ then ans $= x$
2. else recu Euclid GCB $(y, x \bmod y)$

Correctness:

Prove $gcd(x,y) = gcd(y, x \bmod y)$

(prove 1. $gcd(y, x \bmod y)$ divides $gcd(x,y)$)

2. Vice versa

Show That: $gcd(x,y) \mid gcd(y, x \bmod y)$

Proof: Let $d = gcd(x,y)$

$d|x \Leftarrow x/d$ and $y/d \Rightarrow d|x$

$d|x$ and $d|y \Rightarrow d|(x+y)$

$d|ax+by$ → any such eqn

d divides $x - y \cdot \lfloor \frac{x}{y} \rfloor$

$\Rightarrow$ d divides $x \bmod y$.

If $d|x$ and $d|y$ then $d|\gcd(x,y)$.

$\Rightarrow$ $d|y$ and $d|x \bmod y$ then $d|\gcd(y, x \bmod y)$

$\Rightarrow$ $\gcd(x,y) = d$

$\Rightarrow$ $\gcd(x,y) | \gcd(y, x \bmod y)$ (divides)

$x \bmod y = x - y\lfloor \frac{x}{y} \rfloor$

② To show $GCD(y, x \bmod y) \mid GCD(x, y)$

Let $d = GCD(y, x \bmod y)$

$\Rightarrow$ d divides y and d divides $x \% y$.

$x = y\lfloor \frac{x}{y} \rfloor + x \bmod y$.

$\Rightarrow$ d divides 'y' and d divides $x \% y$

$\Rightarrow$ d divides x $(d|x)$

$\Rightarrow$ $d|x$ and $d|y$ $\Rightarrow d | \gcd(x,y)$

$\rightarrow$ $\gcd(y, x \bmod y) \mid \gcd(x,y)$

Euclids GCD algorithm:

$n = \lfloor \log_2 x \rfloor + 1 + \lfloor \log_2 y \rfloor + 1$

Running time:

1. x and y — input (2) $n = \lfloor \log_2 x \rfloor + 1 + \lfloor \log_2 y \rfloor + 1$
2. Division
3. T(n) — depends on input and the values.

Best case: $T(n) = O(1)$ (144, 89)

Worst case: $T(n) = O(n)$ $O(\log y)$ $O(y)$ $O(x)$

1845: Theorem — Lame's (to find WC RT):

Let k = no. of steps euclidian algorithm for the i/p x, y where $x > y$. Then $y \geq f_k$, k^{th} fibonacci number.

Proof: GCD(21, 35) — Suppose

$(x > y)$ = GCD(35, 21) — $(x_{k+1}, y_{k+1}) = 35, 21$

= GCD(21, 14) — $(x_k, y_k) = 21, 14$

= GCD(14, 7) — $(x_{k-1}, y_{k-1}) = 14, 7$

= GCD(7, 0) — Three consecutive pairs.

= 7

Claim: If (x_{k+1}, y_{k+1}), (x_k, y_k), (x_{k-1}, y_{k-1}) are three consecutive pair of euclidian algorithm for finding GCD of x, y. where $x > y$, then $y_{k+1} \geq y_k + y_{k-1}$ — ①.

Proof of claim:

$x_k = y_k * \left\lfloor \frac{x_k}{y_k} \right\rfloor + x_k \bmod y_k$ (from alg.)

$x_k = y_k * \left\lfloor \frac{x_k}{y_k} \right\rfloor + y_{k+1}$

$x_k = y_{k+1}$

$x > y \Rightarrow \left\lfloor \frac{x_k}{y_k} \right\rfloor \geq 1$

$y_{k+1} \geq y_k + y_{k-1}$

Based on mathematical Induction on 'k':

Base Case: $k=0$: $y \geq F_0$; $y \geq 0$ ✓

↳ Zeroth fibonacci — 0

First fibonacci — 1

Inductive hypothesis:

If no.of (#) steps $\leq k$ then,

$$y_k \geq F_k$$

Inductive step:

$gcd(x_{k+1}, y_{k+1})$ — denotes 'k+1' steps are req. to find GCD.

$gcd(x_k, y_k)$ — denotes 'k' steps

$$y_{k+1} \geq y_k + y_{k-1}$$

(From hypothesis $y_k \geq F_k$; $y_{k-1} \geq f_{k-1}$

$$y_{k+1} \geq F_{k-1} + f_k$$

$$\boxed{y_{k+1} \geq F_{k+1}}$$

Hence proved

$$y \geq \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^k - \left(\frac{1-\sqrt{5}}{2}\right)^k\right]$$

$1 - \left(\frac{1+\sqrt{5}}{2}\right) = -0.618$

GOLDEN RATIO = 1.618

For large values of k;

$$y \geq \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^k$$

$$\left(\frac{1+\sqrt{5}}{2}\right)^k \leq \sqrt{5}\, y$$

$$k = \log_{\frac{1+\sqrt{5}}{2}}(y\sqrt{5}) = \log_{\frac{1+\sqrt{5}}{2}} y + \frac{1}{2}\log_{\frac{1+\sqrt{5}}{2}} 5$$

k – Running time

$$k = \log_2 y * \log_{\frac{1+\sqrt{5}}{2}} 2 + \log_{\frac{1+\sqrt{5}}{2}} \sqrt{5}$$

$$k \leq C \log_2 y \qquad C \geq \log_{\frac{1+\sqrt{5}}{2}} 2$$

$$\boxed{k = O(\log y)} \qquad n = \text{---} + \log_2 y$$

$$\boxed{k = O(n)}$$ — Linear time.

Worst case input: Consecutive fibonacci.

Chapter 31.2

Algorithm: Find max(A, n)

1. max ← -1
2. for i ← 1 to n
3. if A[i] > max
4. max ← A[i]
5. .

1. n
2. comparision
3. T(n) — depends only on input size.
4. $T(n) = \sum_{i=1}^{n} 1 = n$
5. $T(n) = \Theta(n)$

Calculate no.of times max ← A[i] is performed.

1. n
2. Assignment
3. T(n) — no.of times operation performed.
4. Depends on 'n' and actual values.

Best Case: 4 3 2 1 | 4 2 3 1 0

$O(1) = \Theta(1) = \Omega(1)$

Worst Case: $T(n) = \Theta(n)$

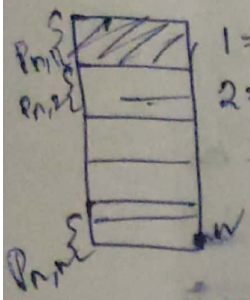
$= O(n)$

$= \Omega(n)$

4 5 6 7

AVERAGE CASE :-

Total no. of inputs = n! (permutations)



1 => T(n) = (n-1)!
2 => T(n) [< T(n) < n

$\sigma_1, \sigma_2, \ldots, \sigma_n$ → last element

$1 \le i \le n$

$\sigma_i \in \{1, 2, \ldots n\}$

$i \neq j ; \sigma_i \neq \sigma_j$

$S(n,k)$ — no. of permutation of n inputs where no. of assignments in permutation is k.

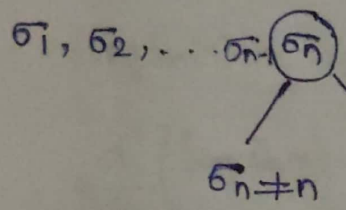
All permutations are equally likely.

$P_{n,k} = \frac{S_{n,k}}{n!}$

Let 'x' be no. of times the assignment operation is performed.

Expectation of X: $\sum_{k=1}^{n} k \, P_{n,k} \quad \sum_{k=1}^{n} k \frac{S_{n,k}}{n!}$

$$E(x) = \frac{1}{n!} \sum_{k=1}^{n} k \cdot S_{n,k} \quad \text{—①}$$



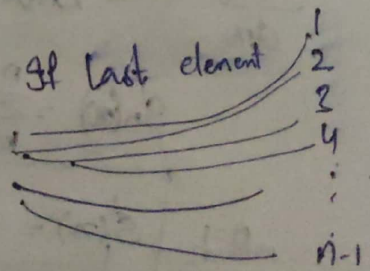
$\sigma_n = n$ — k assignments happened,

$\sigma_1, \sigma_2 \ldots \sigma_{n-1}$ — k-1 assignments happened — $S_{n-1,k-1}$

$\sigma_n \neq n$

$\sigma_1 \; \sigma_2 \ldots \sigma_{n-1}$

k assign. operations. $S_{n-1,k}$



$(n-1) S_{n-1,k}$

$$S_{n,k} = S_{n-1,k-1} + (n-1) S_{n-1,k} \quad \text{—②}$$

Generating functions:

$a_0, a_1, a_2 \ldots\ldots$ – sequence

↓

polynomial $a_0x^0 + a_1x^1 + a_2x^2 + a_3x^3 + \ldots\ldots + \infty$

↓

Generating func. for sequence $a_0, a_1, \ldots a_n$

Ex:- 1, 1, 2, 3, 5, 8, 13, 21 . .

$G(x) = 1x^0 + 1x^1 + 2x^2 + 3x^3 + 5x^4 + 8x^5 + 13x^6 + 21x^7$

$G(x) = 2 + 3x^3 + 4x^5 + \ldots\ldots$

2, 0, 0, 3, 0, 4

$$S_{n,k} = S_{n-1,k-1} + (n-1)\,S_{n-1,k}$$

$S_{n,0}\ S_{n,1}\ S_{n,2} \ldots\ldots$

$G(x) = S_{n,0}x^0 + S_{n,1}x + S_{n,2}x^2 + \ldots\ldots + S_{n,n}x^n + S_{n,n+1}x^{n+1} + \ldots$

$$G(x) = \sum_{k=0}^{\infty} S_{n,k}x^k$$

$S_{n,0} = 0$ ∴ $S_{n,k} = 0$ where $k > n$

$$\Rightarrow G(x) = \sum_{k=1}^{n} S_{n,k}x^k$$

$$S_n(x) = \sum_{k=1}^{n} S_{n,k}x^k \quad \text{—(3)}$$

Take derivative

$$\frac{d(S_n(x))}{dx} = S_{n,k}\sum_{k=1}^{n} S_{n,k}\,k(x^{k-1})$$

$$S'_n(x) = \sum_{k=1}^{n} S_{n,k}\,k\,(x^{k-1})$$

$n=1$ $$S'_1(x) = \sum_{k=1}^{n} S_{n,k}\,k \quad \text{—(4)}$$

$$S_{n,k} = S_{n-1,k-1} + (n-1)\,S_{n-1,k}$$

↓

Multiply with x^k and $\sum_{k=1}$

$$\sum_{k=1} S_{n,k}\cdot x^k = \sum_{k=1}(S_{n-1,k-1})x^k + \sum_{k=1}(n-1)x^k \cdot S_{n-1,k}$$

$S_n(x) = x\,S_{n-1}(x) + (n-1)\,S_{n-1}(x)$

$S_n(x) = (x+n-1)\,S_{n-1}(x)$ — (5)

$S_1(x) = x$

$\Rightarrow S_2(x) = (x+2-1)S_1(x) = (x+1)(x)$

$S_3(x) = (x+3-1)S_2(x) = (x+2)(x+1)(x)$

$S_4(x) = (x+4-1)S_3(x) = (x+3)(x+2)(x+1)(x)$

$S_n(x) = (x+n-1)(x+n-2)\ldots(x+1)(x)$

$$\boxed{S_n(x) = (x)(x+1)\ldots(x+n-1)} \quad \text{— (6)}$$

$S_n(1) = n!$ $\quad ((1)(2)\ldots(n))$

$$E(x) = \sum_{k=1}^{n} \frac{k}{n_k} S_{n,k}$$

Equation (7) $\quad \boxed{E(x) = \dfrac{S_n'(1)}{S_n(1)}}$ — (7)

$$S_n(x) = \prod_{j=0}^{n-1} (x+j)$$

$\log S_n(x) = \log x + \log(x+1) + \log(x+2) + \ldots + \log(x+n-1)$

$\dfrac{d}{dx}\{\log S_n x\} = \dfrac{d}{dn}\{\log x + \log(x+1)\}$

$\dfrac{S_n'(x)}{S_n(x)} = \dfrac{1}{x} + \dfrac{1}{x+1} + \ldots + \dfrac{1}{x+n-1}$

$x = 1 \quad \dfrac{S_n'(1)}{S_n(1)} = 1 + \dfrac{1}{2} + \ldots + \dfrac{1}{1+n-1}$

$= 1 + \frac{1}{2} + \ldots + \frac{1}{n}$

n^{th} Harmonic number.

$E(x) = \dfrac{S_n'(1)}{S_n(1)} = \dfrac{S_n'(1)}{S_n(1)} = H(n)$

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

The operation $\max \leftarrow A[i]$ — performed is $O(n)$.

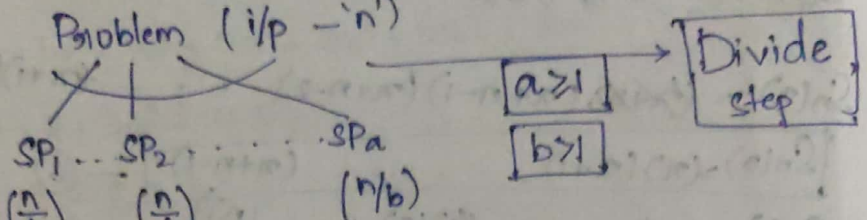
28/01/19

1. Show that $\sum_{i=1}^{n} 1/i = \Theta(\log n)$

Designing Algorithms:

Given a problem (input size - 'n')

→ Problem is divided into subproblems (less i/p size and same type)

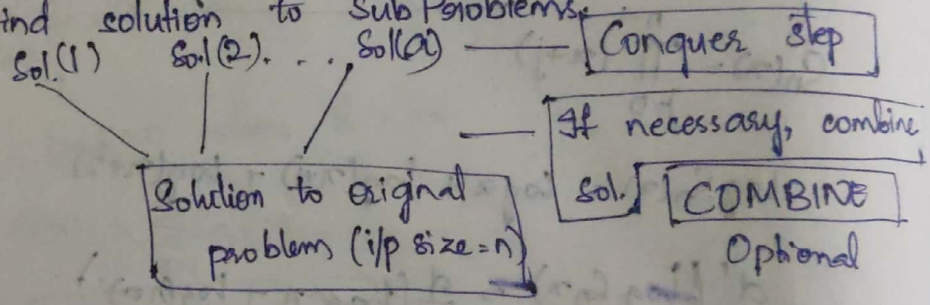


Assumption: All SP are of same size.

If $SP_1 \ldots SP_a$ is not small, again divided into 'a'. with i/p - $\frac{n}{b^2}$ .. and so on.

→ $\frac{n}{b}$ - small or not, decided based on problem.

→ Next find solution to Sub Problems.



→ This method is called Design and Conquer Algorithm.

Ex: Merge Sort, Quicksort.

→ In general, problem (n - i/p size), then

Running time = T(n)

D(n) & C(n)

D(n) — Divide Time

C(n) — Combine time.

Conquer = $aT(n/b)$

$T(n) = D(n) + aT(n/b) + C(n)$ $n>1$; $a\geq 1$, $b>1$

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

$f(n) = D(n) + C(n)$; $f(n) \geq 0$ (or) $f(n) > 0$ definitely non-negative

$$T(n) = aT(n/b) + f(n), \text{ if } n > \square$$
$$= g(n), \text{ if } n \leq \square$$

General Divide and Conquer Recurrence.

If SP are not of same size, then

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

$\Rightarrow T(n) \leq 2T(n/2) + n^2$ - general

$T(n) \geq 2T(n/3) + n^2$ - general

Ex: $T(n) = 4T(n/2) + n^2$ - Here $a > b$

$T(n) = T(n/2) + n^2$ — $a < b$ No relation b/w 'a' and 'b'.

→ There are methods to find TOG for a given recurrence relation. (doesnot actually solve the problem).

Binary Search:

$D(n) = \Theta(1)$

$C(n) = 0$ (No work)

(Sol. of SP = Sol. of Original Problem).

Conquer - $1.T(n/2)$

$a = 1$ — One SP solved (check key with middle element and solve only one SP)

$b = 2$ — size = $n/2$. (Left or right)

$T(n) = 1T(n/2) + \Theta(1)$; $n \geq 2$.

$$T(n) = T(n/2) + \Theta(1) \; ; \; n > 1$$

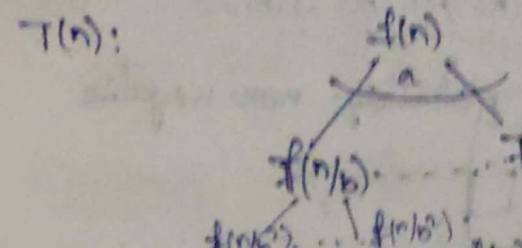
For $n = 1$: $T(n) = 1$ (Key found or not found).

Precisely — $T(n) \leq T(\lceil n/2 \rceil) + \Theta(1)$

Generally, $\lceil\ \rceil$ (ceiling) ; $\lfloor\ \rfloor$ (floor) no impact.

Without (or) with - same TOG.

$T(n)$:

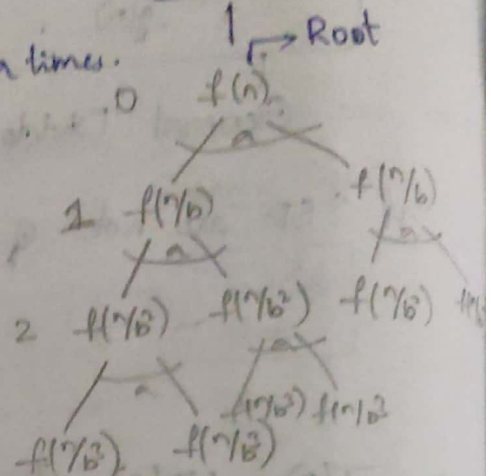


$T(n) = aT(n/b) + f(n)$

$T(n/b) = aT(n/b^2) + f(n/b)$

$T(n/b^2) = aT(n/b^3) + f(n/b^2)$

Recursion Tree



Complete a-ary tree.
(a^i nodes at level i)

Level No.		Amt. of Work done
(Root)	0	$f(n)$
	1	$a \cdot f(n/b)$
	2	$a^2 \cdot f(n/b^2)$
	3	$a^3 \cdot f(n/b^3)$
	⋮	
	i	$a^i f(n/b^i)$

$T(n/b^i) =$

(we stop at $n = 1$)

$\log_b n$ $T(i)$

30/01/19

At $\frac{n}{b^i} = 1$ it becomes $T(i)$

$\boxed{i = \log_b n}$

~~Level~~ ~~Amount of work done~~ work.

Last level — $\log_b n$. $a^{\log_b n}\, \Theta(i)$. $T(i) = \Theta(1)$

~~$T(n) = \sum_{j=0}^{(\log_b n - 1)}$~~

$$T(n) = \sum_{j=0}^{(\log_b n - 1)} a^j f(n/b^j) + \Theta(n^{\log_b a})$$

Last level.

↳ Simplication so that $T(n) = f(n)$

$$T(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j) + \Theta(n^{\log_b a})$$

Let $g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$ $\quad a \geq 1 ; b > 1 ; f(n)$ -tve fn. $n \geq b^k \quad k > 0$.

$\Rightarrow T(n) = g(n) + \Theta(n^{\log_b a}) \quad \text{---} \quad (1)$

Case-1: $f(n) = O(n^{\log_b a - \varepsilon})$, for some constant $\varepsilon > 0$.

$\rightarrow$ $f(n)$ OOG smaller than OOG of $n^{\log_b a - \varepsilon}$ (since $\varepsilon > 0$)

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j \, O\left(\left(\frac{n}{b^j}\right)^{\log_b a - \varepsilon}\right)$$

$$g(n) = O\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \varepsilon}\right)$$

$$g(n) = O\left(n^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b n - 1} a^j \times (b^j)^{-(\log_b a - \varepsilon)}\right)$$

$$g(n) = O\left(n^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b n - 1} a^j \times a^{(b^j)(-\log_b a)} \cdot (b^j)^{\varepsilon}\right)$$

$$g(n) = O\left(n^{\log_b a - \varepsilon} \sum_{j=0}^{(\log_b n - 1)} a^j \times a^{-j} \times b^{j\varepsilon}\right)$$

$$g(n) = O\left(n^{\log_b a - \varepsilon} \sum_{j=0}^{(\log_b n - 1)} (b^{\varepsilon})^j\right)$$

$\frac{a(1-r^n)}{1-r}$ $\quad a = 1 ; r = b^{\varepsilon}$

$$g(n) = O\left(n^{\log_b a - \varepsilon} \cdot \left(\frac{1(1-b^{\varepsilon})^{\log_b n \cdot \varepsilon}}{1-b^{\varepsilon}}\right)\right)$$

$$g(n) = O\left(n^{\log_b a - \varepsilon} \times \frac{n^{\varepsilon} \times b^{-\varepsilon}}{b^{\varepsilon} - 1} \cdot \frac{1 - n^{\varepsilon}}{1 - b^{\varepsilon}}\right)$$

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

$\rightarrow$ The workk done at leaf nodes dominates the entire work.

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

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

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

Case-2: If $f(n) = \Theta(n^{\log_b a})$

$$g(n) = \sum_{j=0}^{(\log_b n - 1)} a^j \Theta\left(\left(\frac{n}{b^j}\right)^{\log_b a}\right)$$

$$g(n) = \Theta\left[\sum_{j=0}^{\log_b n - 1} a^j \cdot \frac{n^{\log_b a}}{b^{j \log_b a}}\right]$$

$$= \Theta\left[n^{\log_b a} \sum_{j=0}^{\log_b n - 1} 1\right]$$

$$= \Theta\left[n^{\log_b a} \times (\log_b n - 0 + 1)\right]$$

$$g(n) = \Theta\left[(\log_b n)\, n^{\log_b a}\right]$$

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

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

Case 3: $f(n) = \Omega(n^{\log_b a + \varepsilon})$, for some $\varepsilon > 0$.

and $\boxed{af(n/b) \leq cf(n)}$ for some constant $\boxed{c < 1}$

$$f(n/b) \leq \left(\frac{c}{a}\right) f(n)$$

$$f(n/b^2) \leq \left(\frac{c}{a}\right) f(n/b) \leq \left(\frac{c}{a}\right)^2 f(n)$$

$$f(n/b^j) \leq \left(\frac{c}{a}\right)^j f(n) \quad \text{—— (3)}$$

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

$$\leq \sum_{j=0}^{(\log_b n - 1)} a^j \left(\frac{c}{a}\right)^j f(n)$$

Regularity Condition

$$= f(n) \sum_{j=0}^{\log_b n - 1} c^j$$

$$\leq f(n) \sum_{j=0}^{\infty} c^j$$

$$g(n) = f(n)\left(\frac{1}{1-c}\right)$$

$$g(n) = O(f(n)) \qquad f(n) = \Omega(n^{\log_b a})$$

$$\hookrightarrow n^{\log_b a + \varepsilon}$$

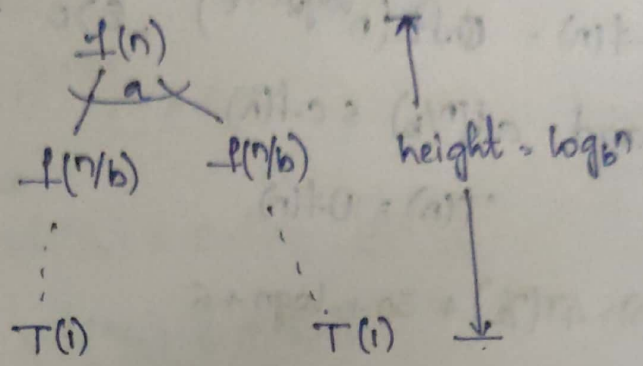
→ $g(n) = \Theta(f(n))$

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

→ These are **Master cases** (or) **Master Theorem**

Case 3: Regularity Condition.

Regurgence Tree:



$f(n)$ – woork done in dividing the problem.

Case-1:
Woork at last level (leaf nodes) = $T(1) = \Theta(a^{\log_b n})$ dominates all other levels.

Case 2: Same woork done at all levels.

Case 3: Woork done at ropot node dominates all levels.

→ These three cases donot solve all problems of this form.

01/02/19.

$f(n)$, a,b s.T $f(n) = O(n^{\log_b a - \varepsilon})$ – should fail

$f(n) = \Omega(n^{\log_b a + \varepsilon})$ – ✓

$f(n) : \frac{n^2}{\log n}$: $n^{\log_b a} = n^2$

1. An example of $f(n)$ S.T case 3: should be satisfied.

$f(n) = \Omega(n^{\log_b a + \varepsilon})$ – satisfies

$af(n/b) \leq c \cdot f(n)$ – doesnot satisfy.

Cases:

1. $f(n) = O(n^{\log_b a - \varepsilon})$ and $\varepsilon > 0$

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

2. $f(n) = \theta(n^{\log_b a})$

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

3. $f(n) = \Omega(n^{\log_b a + \varepsilon})$, $\varepsilon > 0$

   and $af(n/b) \leq cf(n)$

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

Ex: 1. $T(n) = 4T(n/2) + 3n + \log n + 6$

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

$\Rightarrow a = 4; b = 2; f(n) = 3n + \log n + 6$

$f(n)$	$n^{\log_b a}$
$3n + \log n + 6$	$n^{\log_2 4} = n^2$

$3n + \log n + 6 = O(n^{2-0.4})$ $\boxed{\varepsilon = 0.4}$

∴ By case 1 of master theorem,

$T(n) = \theta(n^{\log_b a}) = \theta(n^2)$

If $f(n) = n^2$, both have same OOG

~~$f(n) = \text{OOG}\ \theta(\log n^a\ n)$~~

$f(n) = \theta(n^{\log_b a}) = \theta(n^2)$

∴ By case 2,

$T(n) = \theta(n^{\log_b a} \cdot \log n) = \theta(n^2 \cdot \log n)$

~~If $f(n) >$~~

Ex: 2. $T(n) = T(n/2) + \text{constant}$

$a = 1;\ b = 2 \Rightarrow n^{\log_b a} = n^0 = 1$

$f(n) = \text{constant} = \Theta(\log n^{\log_b a})$

By case 2,

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

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

$a = 4;\ b = 2;\ f(n) = n^3 : n^{\log_b a} = n^2$

$n^3 = \Omega(n^{2+\varepsilon}),\ \varepsilon > 0$

Regularity condition:

$af(n/b) \leq c \cdot f(n)$ $\boxed{c<1}$

$\Rightarrow 4f(n/2) \leq c \cdot f(n)$

$\Rightarrow 4\left(\frac{n}{2}\right)^3 \leq c \cdot n^3$

$\Rightarrow 4/8 \leq c$ $\Rightarrow c \geq 1/2$ – true

$c<1$

$\boxed{T(n) = \Theta(n^3)}$

NOTE: If $f(n)$ is a polynomial and $f(n) = \Omega(n^{\log_b a + \varepsilon})$ satisfied, then regularity condition also satisfied.

4. $T(n) \leq 2T(n/2) + n$

$n^{\log_b a} = n^1 \quad f(n) = 1$

Case 2: solved.

But $T(n) \leq 2T(n/2) + n$

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

If $\boxed{T(n) = 2T(n/2) + n \Rightarrow T(n) = \Theta(n \log n)}$

5. $T(n) = \cancel{n \log n}\ nT(n/3) + 200$

Master Theorem not applicable ($\because$ a – not constant)

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

Master theorem – not applicable. ($-n^2 + n$ – '-ve' for large n)

7. $T(n) = 9T(n/3) + \left(\frac{n^2}{\log n}\right)$

Not applicable — ↳ not polynomial level smaller.

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

$T(n) \leq 2T\left(\frac{3}{4}n\right) + n$

$b = 4/3 \; ; \; a = 2.$

$a \geq 1 \; ; \; b > 1$

↳ $T(n) = O(n^{\log_{4/3} 2})$

$T(n) \geq 2T(n/3) + n \qquad n^{\log_3 2}$

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

↳ always true

But Θ bound cannot be found ($\because O \neq \Omega$)

→ To find Θ bound, other methods are possible. ($a \neq b$)

Θ bound — Asymptotically tight bound.

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

$n = 2^k, \; k \geq 0$

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

Let $s(k) = T(2^k)$

$\boxed{s(k) = s(k/2) + 1}$ — Binary Search Recurrence Relation

$s(k) = \Theta(\log k)$

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

$k = \log_2 n$

→ $\boxed{T(n) = \Theta(\log(\log_2 n))}$

By master theorem,

$\boxed{s(k) = 2s(k/2) + 1}$

$s(k) = \Theta(k)$

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

$\boxed{T(n) = \Theta(\log n)}$

10. $T(n) = 2T(\sqrt{n}) + \log n$ ⟵ Display -Alg.

$n = 2^k$, $k \geq 0$

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

$s(k) = 2s(k/2) + k$

$s(k) = \Theta(k \log k)$

$\hookrightarrow T(2^k) = \Theta(k \log k)$

~~$T(n) = \Theta(\log\log\log n)$~~

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

```
if (n = 1)
    print ("NITW")
    return
else
    for i ← 1 to log n
        print ("NITW")
display (√n)
Display (√n)
```

11. $T(n) = \sqrt{n}\,T(\sqrt{n}) + 50n$

~~$n = 2^k$, $k \geq 0$~~

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

Let $s(n) = \frac{T(n)}{n}$

Divide by n:

$\frac{T(n)}{n} = \frac{\sqrt{n}}{n} T(\sqrt{n}) + \frac{50(n)}{n} \Rightarrow \frac{T(n)}{n} = \frac{T(\sqrt{n})}{\sqrt{n}} + 50$

$s(n) = s(\sqrt{n}) + 50$

$s(n) = \Theta(\log(\log n))$

$\therefore T(n) = \Theta(n \log\log n)$

12. $T(n) = 9T(n/3) + n^2 \log n$

$n^{\log_b a}$ so n

doesnot fit cases (not polynomially more - for case 3)

NOTE: If $T(n) = \Theta(n^{\log_b a} (\log n)^k)$, $k \geq 0$

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

satisfies case-2: (Replaces also)

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

04/02/19.

→ Master theorem only gives asymptotic bounds of the problem of that sort ($T(n) = aT(n/b) + f(n)$)

Ex:

1. $T(n) = T(n/3) + T(2n/3) + n$, $n \geq ?$

$T(n) \leq 2T(2n/3) + n$

$f(n) = n$; $a = 2$; $b = 3/2$

$\log_n n^{\log_b a} \Rightarrow n^{\log_{3/2} 2}$

$\because n^{\log_{3/2} 2} > n$

$\Rightarrow n = O(n^{\log_{3/2} 2})$ – case I.

$\Rightarrow T(n) = \Theta(n^{\log_{3/2} 2})$ but here $T(n) \leq 2T(2n/3) + n$

$\Rightarrow T(n) = O(n^{\log_{3/2} 2})$

If $T(n) \geq 2T(n/3) + n$

then with case-III. $T(n) = \Omega(n)$

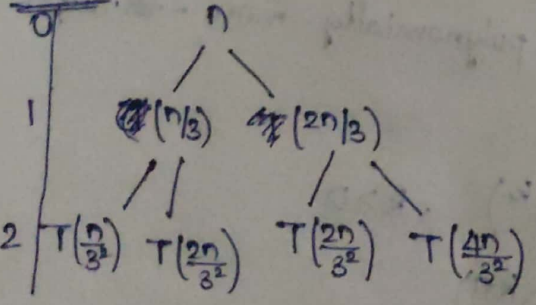
→ Tight bound (Θ) – not found.

$\log_{3/2} 2 = 1.709$

By general Recursion tree method,

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

Level No.



– Not complete binary tree.

At ith level, $T\left(\frac{n}{3^i}\right)$ $T\left(\frac{2n}{3^i}\right)$ $T\left(\frac{2n}{3^i}\right)$ $T\left(\frac{n}{\left(\frac{3}{2}\right)^i}\right)$

At $T\left(\frac{n}{3^i}\right)$ – $n = 3^i \Rightarrow \boxed{i = \log_3 n}$ –

at height i, leftmost becomes $T(1)$

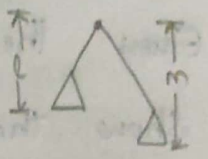
NOTE: At i^{th} level, exactly 2^i nodes should be there.

$T\left(\frac{n}{(3/2)^m}\right)$ at $n = (3/2)^m \Rightarrow \boxed{m = \log_{(3/2)} n}$

the rightmost becomes $T(1)$

$\boxed{m > l}$ – all other values lie in b/wo these values.

→ Height of this tree = $m = \log_{(3/2)} n$.



Level No.	Workdone
0	n
1	n
2	n
⋮	⋮
(upto l) l	n
⋮	O(n)
m	O(n)

Workdone = $\Theta(n)$ – upto level 'l'

After level 'l' – workdone is reduced.

Workdone at each level = $O(n)$ [∵ in general]

Total workdone = $(1+m)\,O(n)$

$= (1 + \log_{3/2} n)\,O(n)$

$= O(n) + O(n) \cdot \log n$

$\boxed{\text{Total workdone} = O(n \log n)}$

Previously, $T(n) = O(n^{\log_{3/2} 2})$

$T(n) = O(n \log n)$ — better than before.

→ Optimising further,

$T(n) \geq \Theta(n)(1+l)$ ⊕

$T(n) \geq \Theta(n)(1 + \log_3 n)$

$T(n) \geq \Theta(n \log n)$

$\Rightarrow \boxed{T(n) = \Omega(n \log n)}$ — better

Previously, $T(n) = \Omega(n)$

$\therefore T(n) = O(n\log n)$ and $T(n) = \Omega(n\log n)$

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

→ Recursion tree gives us a small better interval for the co

Substitution method:

1. Guess the asymptotic solution.
2. Show that the guess is correct.

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

Guess: $T(n) = n\log n + n$

1. Base Step: $T(1) = 1(0) + 1 = 1$ ✓
2. Inductive Hypothesis:

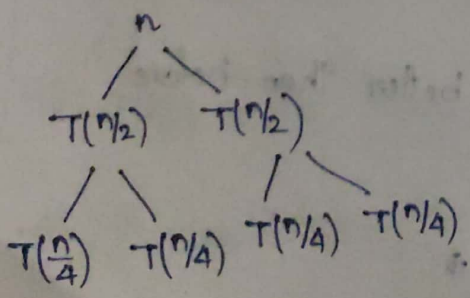
   for $k < n$, $T(k)$ is true.
3. Inductive step:

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

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

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

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



Workdone.

n

n

n

$1 = \frac{n}{2^i}$ - at ith level, $\Rightarrow 2^i = n \Rightarrow i = \log n$

Workdone = $n(i) = n\log n$

2. $T(n) = 2T(n/2) + \Theta(n)$, $n \geq ?$

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

$\Rightarrow T(n) \leq cn\log n \quad \forall\ n \geq ?$ c - constant

→ Base case - not needed.

Inductive Hyp: $k < n \quad T(k) = O(k \log k)$ – true

Inductive step $T(n) = 2T(n/2) + p_2(n) \quad \Theta(n) = pn$ ↓ constant.

$\Rightarrow T(n) \leq 2\left[c \cdot \frac{n}{2} \log \frac{n}{2}\right] + pn$

$\Rightarrow T(n) \leq cn\log n + pn - cn\log 2$

$\Rightarrow T(n) \leq cn\log n - cn + pn$

$\Rightarrow T(n) \leq cn\log n + n(p-c)$ ~~$n(p>0)$~~

$\Rightarrow T(n) \leq cn\log n + n(c-p)$

If $n(c-p) \geq 0$, then ~~$T(n) \leq O(n\log n)$~~

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

$\Rightarrow c \geq p.$

Guess: $T(n) = \Omega(n \log n)$

$\Rightarrow T(n) \geq cn\log n \quad \forall\ n \geq ?$

Ind. Hyp: $k < n \quad T(k) = \Omega(k \log k)$ – true

Ind. step: $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) \quad \Theta(n) = p(n)$

$T(n) \geq 2\left[c \cdot \frac{n}{2} \log\left(\frac{n}{2}\right)\right] + p(n)$

$T(n) \geq cn\log n + pn - cn\log 2$

$\Rightarrow T(n) \geq cn\log n + pn - cn$

$\Rightarrow T(n) \geq cn\log n + n(p-c) \quad p - c \geq 0$

$\Rightarrow T(n) = \Omega(cn\log n)$ ~~$c > p$~~ $p \geq c$

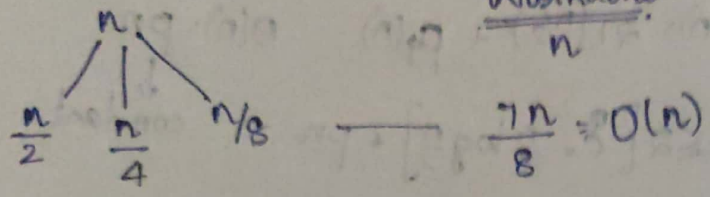
$\Rightarrow \boxed{T(n) = \Theta(n \log n)}$

3. $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + n$

$T(n) \leq 3T(\frac{n}{2}) + n$

$n^{\log_2 3} = n^{1.585}$

$T(n) = O(n^{1.585})$ – correct but not sufficient.



Workdone / n

$\frac{7n}{8} = O(n)$

Height $\frac{n}{2^i} = 1 \Rightarrow i = \log_2 n$.

Total workdone $= (\log_2 n) O(n)$

$= O(n \log n)$ – better.

Guess: $O(n) = T(n)$ ~~$O(n) \Rightarrow T(n) \leq cn$~~

Ind. Hyp: $k < n, \Rightarrow T(k) = O(k)$

Ind. step: $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + n$

$T(n) \leq c\frac{n}{2} + c\frac{n}{4} + c \cdot \frac{n}{8} + n$

$\Rightarrow T(n) \leq c(\frac{7n}{8}) + n$

$\Rightarrow T(n) \leq n(1 + \frac{7c}{8})$

$\Rightarrow T(n) \leq pn$

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

Find DOG for inorder, preorder, postorder of BST.

05/02/19.

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

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

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

$n! < n^n$

$\Rightarrow \boxed{\log n! < n \log n}$

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

Decrease and conquer: (is also divide and conquer)

$T(n) = T(n-4) + \log(n-2) + \log n$

$\Rightarrow T(n) = T(n-6) + \log(n-4) + \log(n-2) + \log n$

After 'i' steps,

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

at $n-2i=1 \Rightarrow 2i = n-1 \Rightarrow i = \frac{n-1}{2}$

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

at $n-2i=0 \Rightarrow i = n/2$

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

$T(0)$ → $\log 0$ not exist

↳ alternate terms

$T(n) \leq \sum_{i=0}^{n} \log i$ — all terms

$\Rightarrow T(n) \leq \log n!$

$\boxed{T(n) = O(n \log n)}$

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

Here

$T(n) = T(1) + \underbrace{\log(n-2i) \cdots}_{n/2 \text{ terms}}$

$\Rightarrow T(n) = \Omega(n/2) = \Omega(n)$

$\boxed{T(n) = \Omega(n)}$

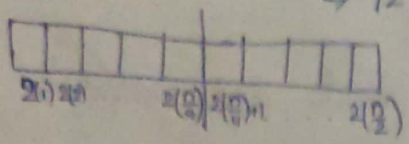
$T(n) = T(n-2) + \log n$

$T(n) = \log 1 + \log 3 + \log 5 + \ldots + \log n$ — if 'n' is odd

$T(n) = \log 2 + \log 4 + \log 6 + \ldots + \log n$ — if 'n' is even.

$$T(n) = \sum_{i=1}^{n/2} \log 2i$$

↳ $n/2$ terms



$$T(n) \geq \sum_{i=1}^{n/4} \log(n/4)$$

$$T(n) \geq \frac{n}{4} \log \frac{n}{4}$$

$$T(n) \geq \frac{1}{4}[n\log n - n\log 4]$$

$$T(n) \geq n\log n$$

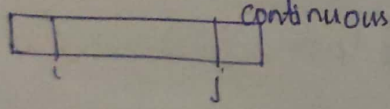
$$\boxed{T(n) = \Omega(n\log n)}$$

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

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

Maximum Subarray Sum problem:

↓ continuous



$1 \leq i \leq n$; $1 \leq j \leq n$ $\sum_{l=i}^{j} A[l]$ is maximum.

Input: An array $A[1 \ldots n]$ of n elements.

Output: Two indices, $1 \leq i \leq n$, $1 \leq j \leq n$ s.t $\sum_{l=i}^{j} A[l]$ is max.

Eg:1.

10	6	20	27	12	6	2
1	2	3	4	5	6	7

— i=1; j=7

↳ entire subarray.

2.

-10	-5	-20	-30	-12	-6	-2

↳ Largest only 1 element

3. | -12 | 20 | 46 | -50 | -26 | 4 | -30 | 2 |
1 2 3 4 5 6 7 8

| 20 | 46 |
2 3

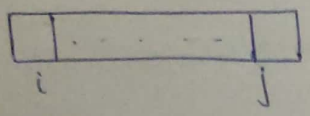
4. | -60 | 4 | -20 | 10 | -40 | 12 | 36 | 44 |
1 2 3 4 5 6 7 8

| 12 | 36 | 44 |
6 7 8

5. | -10 | 25 | 30 | -12 | 28 | -40 | -6 |
1 2 3 4 5 6 7

| 25 | 34 | -12 | 28 |

Brute force method:



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

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

i	j	
1	1,2,3 . . n	(n)
2	2,3 . . . n	(n-1)
3	3,4, . . . n	(n-2)
:		
n-1	n-1, n	(2)
n	n	(1)

} nC_2+n

1. | 10 | 5 | -100 | -200 | 7 | 8 |

(multiple instances)

both answers

For a particular i, 'n' possibilities of 'j' at maximum.

max. time to find sum = (n)

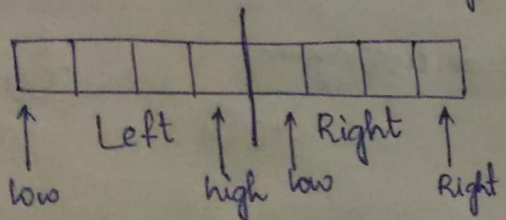
time taken = O(n)

There are nC_2 possibilities $\Rightarrow T(n) = nC_2\, O(n) + n\,O(n)$

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

→ For a better algorithm,

divide the array into two subarrays.



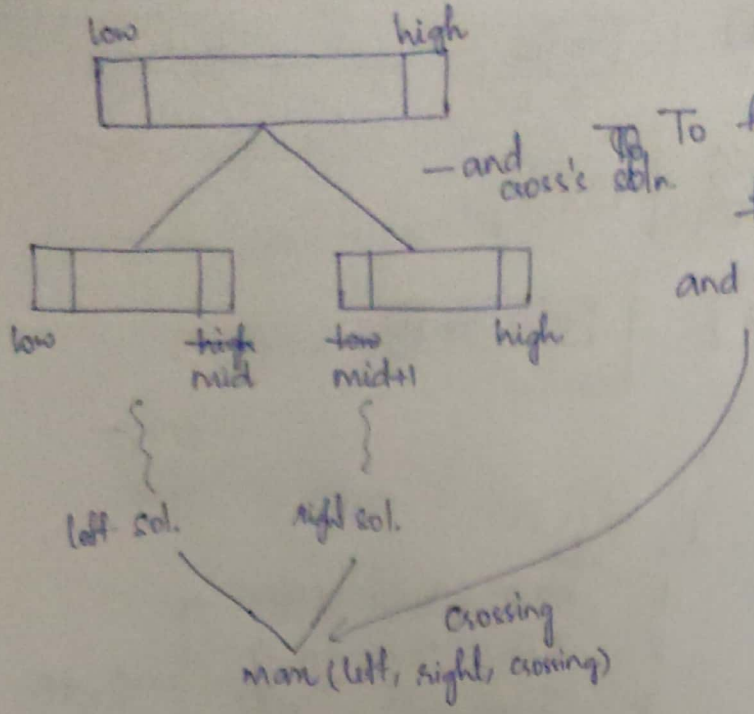
Answer may be on left or right subarray or both.

| -10 | 25 | 8 | 34 | -12 | 28 | -40 | 6 |
1 2 3 4 5 6 7 8

Answer crossing the mid element

↳ Called "Crossing Sum"

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$



To find crossing sum, — and cross's soln.

from mid to low (find sum)

and mid+1 to high (find sum)

Algorithm: T(n)

FIND MAXIMUM SUB-ARRAY (A, LOW, HIGH)

① if low = high (termination) — O(1)

return (low, high, A[low])

else

mid ← $\left(\frac{low + high}{2}\right)$ — O(1)

(left low, left high, left sum) ← find maximum subarray (A, low, mid) — T(n/2)

(right low, right high, right sum) ← find maximum subarray (A, mid+1, high) ↳ T(n/2)

(cross low, cross high, crossing sum) ← find max crossing subarray (A, low, mid, high) ↳ O(n)

max (left sum, right sum, crossing sum) ↳ O(1)

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

↳ By case 2 of Master Theorem

$\boxed{T(n) = \Theta(n \log n)}$

Problem Complexity = n.

06/02/19.

(Generally) Exact recurrence,

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + \Theta(n) \qquad n \neq 2^k$$

floor, ceil

$$T(n) \leq 2T\left(\left\lceil \frac{n}{2} \right\rceil\right) + \Theta(n)$$

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

→ Max. Subarray problem — PC $\frac{}{n}$ AC $\frac{}{\Omega(n)}$

Matrix Multiplication:

$A = (a_{ij})_{m \times n}$

$B = (b_{ij})_{p \times q}$

$C = AB$

$\Rightarrow n = p$ — for compatibility

Order of $C = n \times q$

For finding one entry, no. of multiplications = n (or) p.

Total no. of multiplications = $(m \times q) \times n$

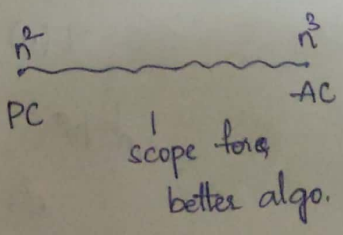
$= mnq$ (Θ) mpq

For a square matrix, no. of multiplications = n^3

→ Scalar multiplications — elemental multiplications done $= \Theta(n^3)$

Problem Complexity = $\Theta(n^2)$

→ Algorithm complexity = $\Omega(n^2)$



→ Dividing the problem,

$$A = (a_{ij})_{n \times n} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

$$B = (b_{ij})_{n \times n} = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$C = AB$

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} * \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$(n/2)$ each $\quad$ $(n/2)$ each

$C_{11} = A_{11}B_{11} + A_{12}B_{21}$; $C_{12} = A_{11}B_{12} + A_{12}B_{22}$

$C_{21} = A_{21}B_{11} + A_{22}B_{21}$; $C_{22} = A_{21}B_{12} + A_{22}B_{22}$

→ Divide till $n = 1$ — stop and multiply.

$A_{11}B_{11}$, $A_{12}B_{21}$ — subproblems of order $(n/2)$.

8 matrix multiplications — when $n \geq 2$; $n = 2^k$

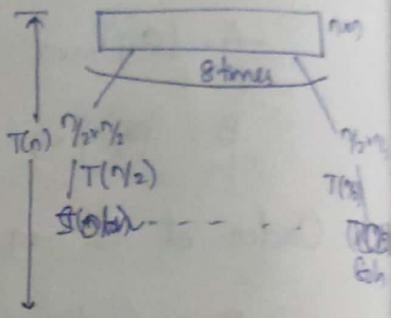
of order $n/2$.

→ For dividing the problems,

→ Divide the problem and find soln.

and "add".

No. of addition operations = n^2. (general)



For order $\frac{n}{2} \times \frac{n}{2}$, add. operations = $\frac{n^2}{4}$ $\left(\frac{n}{2} \times \frac{n}{2}\right)$

(4) Four additions perform in subproblems = $4 \times \frac{n^2}{4} = n^2$

The addition operation — combining.

Total work:

$$T(n) = 8T\left(\frac{n}{2}\right) + n^2 \quad , n \geq 2.$$

↳ addition.

$n^{\log_b a} = n^{\log_2 8} = n^3$ (1st case)

$n^2 = O(n^3)$ — Master Theorem

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

$T(n) = \Theta(n^{\log_2 8})$

$$\boxed{T(n) = \Theta(n^3)}$$

same as normal multiplication (no use of extra work)

$$T(n) = 8T(n/2) + n^2 \quad , n \geq 2$$

If $T(n) = 7T(n/2) + n^2$ — better than $T(n) = 8T(n/2) + n^2$

$\Downarrow$

7 subproblems solved. ∴

$\rightarrow T(n) = n^{\log_2 7}$

$\Rightarrow$ Reduced no. of subproblems from 8 to 7. and increased the no. of additions and subtractions to 18.

Now, $T(n) = 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2 \quad , n \geq 2$

↳ $n/2 \times n/2$ - additions.

Strassen's Matrix multiplication algo.:

→ Multiplications were reduced at the cost of addition.

$$T(n) = 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2 \quad , n \geq 2$$

$$T(n) = \Theta(n^{\log_2 7})$$

→ Another algo. was given with dividing subproblems each of size $n/4 * n/4$ and ensured the complexity during divide and conquer is not greater than n^2.

$$T(n) = aT(n/2) + \Theta(n^2)$$

$a = ?$ — for the algo. to be better than Strassen's

↳ max. value of $a = 48$

$f(n) = n^2$

$$n^{\log_4 a} < n^{\log_2 7}$$

$$\log_4 a < \log_2 7$$

$$\Rightarrow a < 4^{\log_2 7}$$

$$a < 2^{\log_2 49}$$

$$a < 49$$

08/02/19.

Strassen's matrix multiplication: Section 4.9

$$T(n) = 7T(n/2) + 18\left(\frac{n}{2}\right)^2 \qquad n = 2^k,\ k \geq 0$$

7 multiplications:

$P_1 = A_{11} S_1$ $\quad S_1 = B_{12} - B_{22}$

$P_2 = S_2 B_{22}$ $\quad S_2 = A_{11} + A_{12}$

$P_3 = S_3 B_{11}$ $\quad S_3 = A_{21} + A_{22}$

$P_4 = A_{22} S_4$ $\quad S_4 = B_{21} - B_{11}$

$P_5 = S_5 S_6$ $\quad S_5 = A_{11} + A_{22}$

$P_6 = S_7 S_8$ $\quad S_6 = B_{11} + B_{22}$

$P_7 = S_9 S_{10}$ $\quad S_7 = A_{12} - A_{22}$

$S_8 = B_{21} + B_{22}$

$S_9 = A_{11} - A_{21}$

$S_{10} = B_{11} - B_{12}$

$C_{11} = P_5 + P_4 - P_2 + P_6$

$C_{12} = P_1 + P_2$

$C_{21} = P_3 + P_4$

$C_{22} = P_5 + P_1 - P_3 - P_7$

Order of $P_i, S_i = n/2 \Rightarrow T(n/2)$

$$T(n) = 7T(n/2) + 18\left(\frac{n}{2}\right)^2, \quad n \geq 2$$

$10\left(\frac{n}{2}\right)^2 + 8\left(\frac{n}{2}\right)^2 = 18\left(\frac{n}{2}\right)^2$

$n = 1$ – Termination.

$T(n) = \Theta(n^{\log_2 7})$

Correctness of algo: Check if $C_{11} = A_{11}B_{11} + A_{12}B_{21}$ and others.

For matrix of order $n \times n$ where $n \neq 2^k$, $k \geq 1$

Then add rows and columns to make it 2^k with the values zero.

$C = A_{18\times 18} B_{18\times 18}$ $\Rightarrow$ add (32-18) zero rows and columns

$2^4 \leq 18 \leq 2^5$

16 $\quad$ 32

Running time is changed,

input 'l' is made to 'n', then $T(n) = n^{\log_2 7}$

of 'l', $T(n) = l^{\log_2 7}$ but $T(n)$ – changed.

We can say than,

$n \leq 2l$ (such value is consider)

$T(l) \leq (2l)^{\log_2 7} \Rightarrow l^{\log_2 7} < l^{\log_2 7}$

$\rightarrow \boxed{T(l) = l^{\log_2 7}}$

So, running time is same (doesnot affect).

Space occupied is more.

Eg: $C = A_{1800 \times 3} B_{3 \times 2}$ (think)

→ For a problem with (divide and conquer), the $T(n)$ can be calculated as

$T(n) =$ (no. of subproblems actually solved) * max. time to solve + (Divide / Conquer time)

→ Generally, no repetition of subproblems (so above eqn ✓)

→ For subproblems which repeat, we neednot solve again.

Suppose,

sub_problem	solved.
A	✓
B	✓
A	neednot solve again (checked from table and directly soln. taken)

→ Suppose, n^2 #subproblems

$O(n)$ – max. time to solve $\Rightarrow T(n) = n^2 * O(n)$

if repeated – # distinct subproblems $< n^2$

$\Rightarrow T(n) < n^2 * O(n)$

if # distinct subproblems – $\Theta(n)$

$\rightarrow T(n) = \Theta(n) * O(n) = O(n^2)$

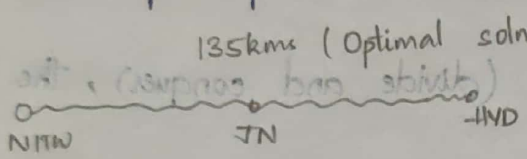
→ If repeated problems are not solved, T(n) may reduce.

The repeated subproblems are called "Overlapping subproblems".

→ For a problem, among all feasible solns, we need to find "optimal soln." (either minimization, maximization)

Eg: Shortest path problem, Travelling salesPerson (TSP) (min. cost Hamiltonian cycle), Minimum Spanning Tree, Graph Colouring, Max Flow, Matrix Chain multiplication problem, KnapSack Problem.

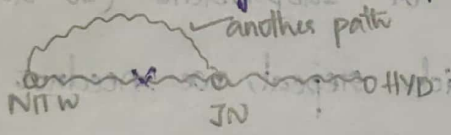
Shortest path problem — Dijkstra's algo. (Single source prob.)



Optimal soln. contains an element (JN)

→ The optimal soln. for path b/w (NITW to JN) also is also along this path.

Proof:



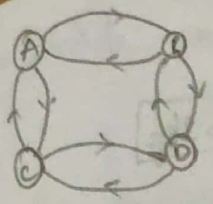
If that is not shortest, another makes shortest path

=> Our statement is wrong (NITW to HYD).

⇒ Optimal soln. to a problem contains optimal soln to its subproblems. and also,

⇒ Optimal soln. to subproblems combinely gives optimal soln. to the problem.

=> The problem is said to have "Optimal substructure prop" (or) "Principle of optimality".



Longest path from A→D is A→B→D
A→C→D

Longest path from A→B is not A→B (part of D)
but A→C→D→B.

⇒ The longest path doesnot contain "Optimal Substructure prop.".

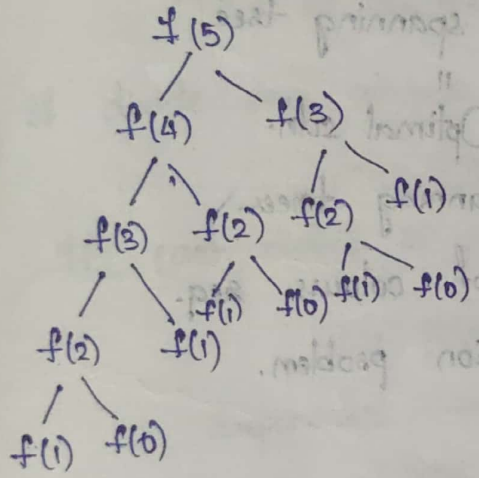
→ Problems with optimal substructure prop and overlapping subproblems

when dynamic prog. applied better soln. obtained.

11/02/19.

For fibonacci sequence,

$f(0)=0;\ f(1)=1;\ f(n)=f(n-1)+f(n-2),\ n\geq 2$



— overlapping subproblems.
(f(2), f(3) – calculated repeatedly)

Optimization problems: (Discrete)

↳ 1. Minimization problems

2. Maximization problems. → Knapshack (0-1) Knapshack problem.

Bag W weight = W

Eg:

Item	1	2	3	.....	n
Value (Rs)	V_1	V_2	V_3	.....	V_n
Weight (Kg)	W_1	W_2	W_3	.....	W_n

- An item should be taken fully or not touched.

Objective fn: Maximize $\sum_{i=1}^{n} v_i x_i$ $1 \leq i \leq n$ $x_i = \begin{cases} 0, & \text{if not chosen} \\ 1, & \text{if chosen.} \end{cases}$

S.T $\sum_{i=1}^{n} w_i x_i \leq W$

1. Item | Salt | Sugar | Gold | Silver

Price(Rs) | 30 | 60 | 1,00,000 | 60,000

Weight(kg) | 3 | 2 | 35 gms | 1 kg

W = 3kg

Optimal soln = {Gold, Silver}

Feasible soln = {Salt} {Sugar, Gold}

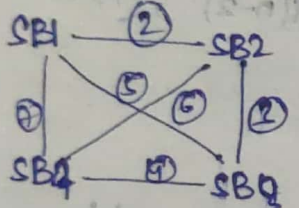
{Sugar} {:

{Sugar, Silver, Salt} (many)

Brute force: (Check all subsets)

and find optimal soln. among them.

2. Minimising the length of wire connected b/w substations (no loop)



– Minimum spanning tree

=

Optimal soln.

Feasible solns. = Spanning trees.

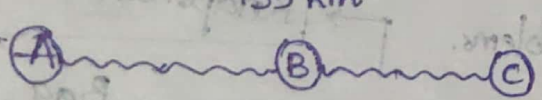
3. Graph colouring – min. no. of colours req.

└ Minimization problem.

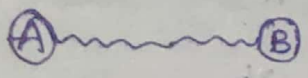
Shortest path problem:

If 'P₁' is shortest path from A to C through B, then 'P₁' – BC is the shortest path from A to B.

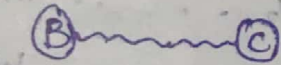
135 km



– shortest



→ shortest



→ shortest

→ Divide and conquer,

$T(n) =$ (no. of SP) (Max. time for each SP)

→ By ensuring each distinct SP is solved only one, DDG can be reduced.

By keeping a track of solved SP, the it can be ensured.

SubProb.	Sol.
(A)	(21)
⋮	⋮

For fibonacci,

sub problems = (n+1) – (distinct)

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

⇒ Similar to no. of nodes searched in AVL.

Total # SP = (exponential)

If divide and conquer,

$T(n) =$ (exp.) $* O(1) = O(exp)$

If each distint SP is solved exactly once,

$T(n) = (n+1) * O(1) = O(n)$

=> Exponential ⟶ Polynomial.

(DDG improved)

→ It is studied under dynamic programming.

→ Dynamic programming – optimal prob. with

(DP) i) optimal sub-structure problems

2) Overlapping prob.

when applied (DP), DDG is improved.

(Bottom-up method – solved from to smallest prob. (lowest SP) to up.)

(Top-down – from big problem to lowest SP).

Top-down > Bottom-up

Top-down – only req. solved

Bottom-up – unnecessary also solved.

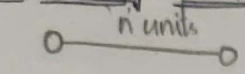
↳ DP (actual)

Top-down – (Memoization)

13/02/19

DDG through DP - (No. of distinct SP) (max(DDG of SP_i))

Rod Cutting Problem:



Divide the rod into 'i' pieces (integer length) so that sum of costs of pieces is maximum.

Length	Price
1	8
2	20
3	30
4	35

1, 2, 3 . . . n — lengths

P_1 P_2 P_3 P_n — price.

Input: n, prices for diff. length.

O/p: (Sum of all pieces) – should be maximum.

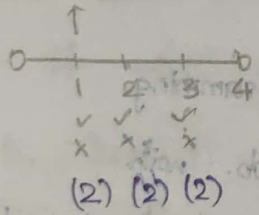
All possibilities — (1,1,1,1) = 32

(1,2,1), (1,1,2), (2,1,1) = 36

(3,1), (1,3) = 38 Brute force

(2,2) — 40

4 — 35



(2) (2) (2)

Total possibilities = 2^3

For n-length, total possibilites = 2^{n-1}.

O(n) – worst case (n-1 cuts)

(for Addition)

Max. time taken = O(n) for SP.

Total time = $2^{n-1} O(n)$

$= O(n.2^{n-1})$ ← brute force.

exponential

(not good)

Problem complexity = $O(n)$.

$O(n)$ – impossible for algo. unless there are any restrictions.

For, $n \geq 1$ (algo. applied)

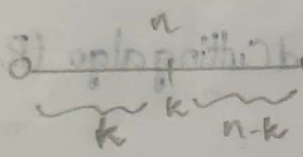
Algorithm:

Let g_n = max. price obtained by selling 'n' length rod.

$g_0 = 0$ ($n=0$, no price)

$g_1 = P_1$

$g_n = g_k + g_{n-k}$ (optimal substructure prop.)



If g_k - max price of 'k'

g_{n-k} - max. price of 'n-k'

$g_n = (g_k + g_{n-k})$ (max + max)

Sometimes,

Length	Price
1	8
2	20

(Here $g_n = g_1 + g_1 = 16$

But max. price = 20.

So, cut is not assured. (if cut, possibilities are formed)

~~But if cut~~ $\Rightarrow$ ~~k can be zero~~.

$g_n = g_k + g_{n-k}$, $n \geq 2$

$g_n = \max \{ g_1 + g_{n-1}, g_2 + g_{n-2}, \ldots, g_{n-1} + g_1, P_n \}$

P_n → (includes no cutting)

$g_0 = 0$ ∴ $g_1 = P_1$ but $g_i \neq P_i$ $(i \geq 2)$

n, p, Q — inputs

~~n = 0 :~~

Recursive algo:

Input: pi, Qi

```
RodCuttingalgo (P,i)
    if n=0
        return 0
    else
        profit ← -∞
        for k ← 1 to i
            Q ← RodCuttingalgo(P,k) + RodCuttingalgo (P, i-k)
            if Q > Profit
                Profit = Q;
if P[i] > Profit ⟸
    Profit = P[i]   return profit
```

$$T(n) = 1 + \sum_{k=1}^{n-1}[T(k) + T(n-k)]$$

$$\boxed{T(n) = 1 + 2\sum_{k=1}^{n-1} T(k)} \Rightarrow T(n) = O(2^n)$$

$T(n) \leq c.2^n$

$T(n)$ = exponential.

$$T(n) \leq 1 + 2\sum_{k=1}^{n-1} c.2^k$$

But each SP, takes $O(n)$.

But total = exponential.

=> Either each SP takes more time

or repetition results in exponential.

No. of distinct SP = $(n+1)$

But each SP = $O(n)$ => not exponential

=> Overlapping SP.

→ Ensured that each distinct SP only once,

$T(n) = (n+1)\ O(n) \Rightarrow (\#SP * \text{each SP})$

$= O(n^2)$ — gr better than exponential.

Now, top-down method (Like Fibonacci)

$g_n = g_1 + g_{n-1}$ (Memoization)

g_1 g_{n-2}

Bottom Up method, (Dynamic Programming)

g_1, g_2 — g_2 found

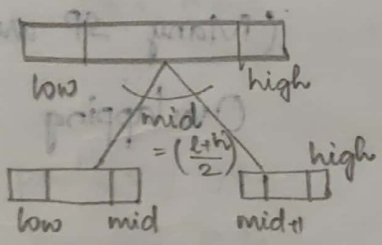
⋮

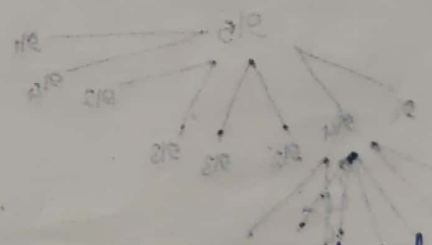
g_n found

18/02/19

1. Check if a number is repeated more than $\frac{n}{2}$ times. (in an array of 'n' numbers).

Finding majority element ($> \frac{n}{2}$ times).





Brute force: Check frequency of every element

$= O(n^2)$

→ Generating subproblems and recursion.

Soln. finding — check if element repeated $> \frac{n/2}{2}$ times.

For a problem, at an instant contains left subarray and right sub array.

	L	R	(four cases)
1)	✓	×	— Majority element (✓) present
2)	×	✓	(×) not present
3)	✓	✓	
4)	×	×	

Case 1: $> \frac{n/2}{2}$

If the original problem has a majority element, then it is from L-HS. (Left subarray).

→ If majority element, check the frequency in entire array.

Case 2: (Similar)

Case 3: Both contain majority elements:

if both same ⇒ that is answer.

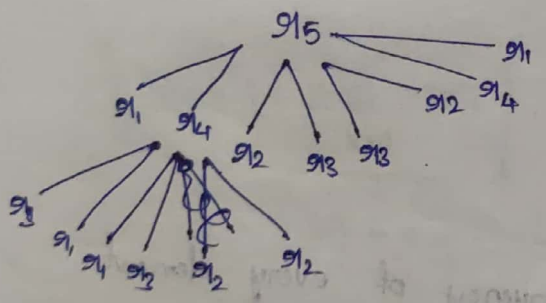
if not (check freq. of both)

Case 4: Both doesnot contain.

(No majority element possible).

Rod Cutting problem (cont.):

No. of distinct SP: $g_{10}, g_{11} \ldots g_{1n}$.



(Many SP are repeated)

Overlapping SPs.

Optimal Substructure + Overlapping SP — DP.

RT (exponential) — # ~~distinct~~ SP - exponential and max. time ≠ exponential (DS)

max. time for SP - exponential.

Case 1: ~~Not true~~

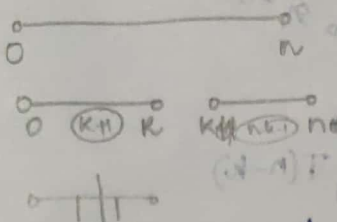
~~# distinct SP~~

Due to repetitions

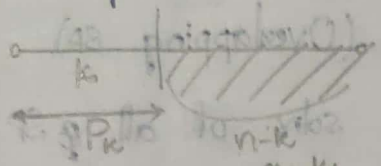
To calculate g_i, we need values from $g_0, g_1, \ldots, g_{i-1}$.

↳ Method 1.

Another method:



With respect to starting point, first cut is after 'l' units, $1 \le l \le n$.



$g_n = P_k + g_{n-k}$ (if $l = k$)

$1 < l < n-1$

$$g_n = \max \begin{cases} P_0 + g_n \\ P_1 + g_{n-1} \\ P_2 + g_{n-2} \\ P_3 + g_{n-3} \\ \vdots \\ P_{n-1} + g_1 \\ P_n + g_0 \end{cases}$$

$P_0 = 0$

$g_0 = 0$

$g_1 = 1$

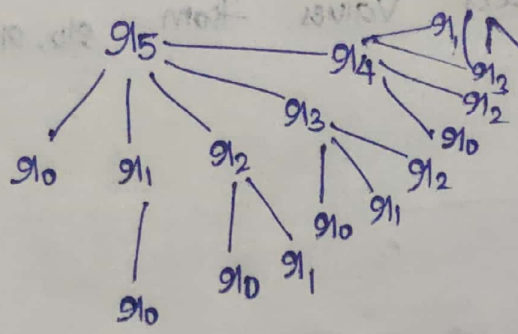
(contains only SP)

$$g_n = \max_{0 \le k \le n-1} (P_{n-k} + g_k)$$

```
profit ← -∞
for k ← 0 to i
    Q ← P[n-k] + RodCuttingAlgo(k);
```

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

```
RodCutting (P, l)
  if l = 0
      return 0;
  else
    for k ← 0 to l-1
      Q profit ← P[k] + RodCutting (P, l-k)
    if Q > profit
        profit = Q;
```



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

= (exponential)

→ Again repetitions are present. (Overlapping SP)

Maintain a table and maintain solns. of all SP.

(Write bottom up and Top down)

Memoization:

```
MemoizedCutRod(p,n)
    let r[0...n] be a new array
    for i=0 to n
        r[i] = -∞
    return MemoizedCutRodAux(p,n,r)

MemoizedCutRodAux(p,n,r)
    if r[n] ≥ 0
        return r[n]
    if n==0
        q=0
    else q=-∞
        for i=1 to n
            q = max(q, p[i]+ MemoizedCutRodAux(p,n-i,r))
    r[n] = q
    return q
```

06/03/19.

Binary Search:

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

Two pointers – (l, h)

Mid = $\frac{l+h}{2}$

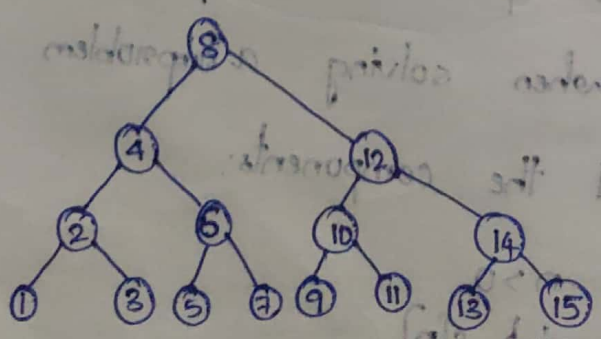
Compare with mid and proceed.

If h < l – element is not present in the list.

Algorithm:

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

→ mid element index



Height = log n

No. of comparisions = log n (maximum)

= 1 (minimum)

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

8/05/19

Rod Cutting Problem:

$$g_n = \max_{1 \le k \le n} \{P_k + g_{n-k}\}$$

$g_0 = 0$; $P_0 = 0$

$$T(n) = \sum_{k=1}^{n-1} T(k) = O(2^n) \text{ - exponential}$$

No. of subproblems solved is exponential.

distinct subproblems = n $(g_1, \ldots, g_n)$

→ Total time for all subproblems = exponential.

1. If no repetition, each subproblem takes O(exp) time
2. each subproblem = O(polynomial) but # SP = exponential.

But here,

Each SP = O(n) (case 1 - ruled out)

→ Case 2 ⇒ n distinct SP are repeatedly solved.

⇒ Overlapping SP.

⇒ 1. Optimal Substructure ⇒ DP can be applied.

2. Overlapping SP

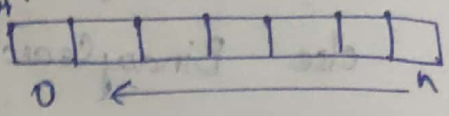
→ We have to ensure that distinct SP are solved exactly once.

Top-Down

```
Rod ( p, r, n)
  if r[n] >= 0
      return r[n]
  if n == 0
      q = 0
  else q = -∞
      for (i ← 1 to n)
          q = max (p[i], Rod ( p, r, n-1));
      r[n] = q;
      return q;
```

```
for i ← 1 to n
    r[n] = -∞
Rod ( p, r, n)
```

r array:



~~T(n) = T(n-i) *~~

Bottom Up

```
Rod
  r[0] ← 0
  for j ← 1 to n
      q = -∞
      for i ← 1 to j
          q = max{ q, p[i] + r[j-i]};
      r[j] ← q
  return r[n]
```

Time Complexity = $O(n^2)$

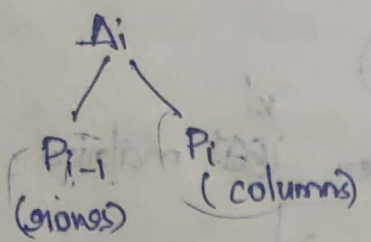
= n SP * O(n)

↳ max. for each SP.

Matrix Chain Multiplication:

$A_1, A_2 \ldots\ldots A_n$

$\forall 1 \le i \le n$



Product of $A_1, A_2 \ldots A_n$ by using minimum no. of matrix element multiplication.

$A_{10\times20}\ B_{20\times30}$ — No. of multiplications = $10\times20\times30$

$A_1 A_2 A_3 = A_1(A_2 A_3) = (A_1 A_2)A_3$

$A_1(A_2A_3)$ → $20\times30\times40$

Order $A_{1\,10\times20}\ (A_2A_3)_{20\times40}$

$= (10\times20\times40)$

Total = $10\times20\times40 + 20\times30\times40$

$= (24+8)\times10^3 = 32\times10^3$

$(A_1A_2)A_3$ — $(10\times20)\ 20\times30$

↓ $10\times20\times30$

$(A_1A_2)_{10\times30}\ A_{3\,20\times40}$

$10\times30\times40$

Total = $(6+12)\times10^3 = 18\times10^3$

$A_1A_2A_3A_4$ — (5 ways)

1. $(A_1A_2)(A_2A_3)$
2. $A_1(A_2A_3A_4) = A_1(A_2(A_3A_4)) = A_1(A_2A_3)A_4$
3. $(A_1A_2A_3)A_4 = A_1(A_2A_3)A_4 = (A_1A_2)A_3)A_4$

Proper parenthisation should be given.

No. of ways for parenthesis = n^{th} Catalan's number.

(A_1) no. of ways = 1

$(A_1)(A_2)$ no. of ways = 1

P(n) – no. of diff ways of parenthesising a chain of n matrices.

P(1) = 1 : P(2) = 1

$\underbrace{(A_1 A_2 \ldots A_k)}_{P(k)} \underbrace{(A_{k+1} \ldots A_n)}_{P(n-k)}$

First parenthesising is after k matrices :

$$P(n) = \sum_{k=1}^{n-1} P(k) * P(n-k), \quad n \geq 2$$

P(1) = 1

$P(n) = \Omega\left(\frac{4^n}{n^{3/2}}\right)$ – n^{th} catalan number.

For finding order = $\Omega\left(\frac{4^n}{n^{3/2}}\right) * O(n)$ ↳ max. time to find no. of multiplications

→ Check if it has "optimal substructure".

Take any subproblem

$A_i \ldots A_j \quad (i \leq j \leq n)$

start at some A_i and multiply chain till A_j.

Let's assume $A_i / \ldots / A_j$ is the optimum

$(A_i \ldots A_k)(A_{k+1} \ldots A_j)$

A_k is present in the optimum way.

↳ parenthesis is placed after A_k.

A_i to A_k – optimum way is A_i to A_k itself.

∵ $(A_i \ldots A_k)(A_{k+1} \ldots A_j)$ is the optimum soln for $A_i \ldots A_j$

⇒ (A_i to A_k) – optimum soln. from $(A_i \ldots A_k)$

(Like distance problem (NITW $\xrightarrow[JN]{}$ HYD))

NITW = A_i : HYD = A_j : JN = A_k.

$(A_1 \ldots\ldots A_7)$

$(A_1 \ldots A_4)(A_5 \ldots A_7)$ — optimum way

$\Rightarrow [(A_1 A_2)(A_3 A_4)][A_5 (A_6 A_7)]$ — "

None A_1 to A_4 optimum way is $[(A_1 A_2)(A_3 A_4)]$.

→ It has optimum substructure property.

11/03/19.

$A_1 - P_{i-1} \times P_i$

$A_1, A_2 \ldots\ldots A_n$

$\forall\ 1 \le i \le n$, $A_i \rightarrow$ order $P_{i-1} \times P_i$

$m[i,j]$ — no. of scalar multiplications to calculate $A_i \ldots\ldots A_j$

$(A_i \ldots\ldots A_k)(A_{k+1} \ldots\ldots A_j)$

$(i-1 \times k)$ $(k \times k+1)$ $(j-1 \times j)$

$m[i,k]$ $m[k+1,j]$

$A_{i-1 \times k}$ $B_{k \times j}$

$$m[i,j] = m[i,k] + m[k+1,j] + P_{i-1} P_k P_j$$

↓

But we want minimum case only.

$$m[i,j] = \begin{cases} \min\{m[i,k] + m[k+1,j] + P_{i-1} P_k P_j\} & i<j \\ 0, & i=j \end{cases}$$

Algorithm Recursive MCM (P, i, j)

```
{
    if (i == j)
        return 0;
    else if (i < j)
        q ← ∞
        for k ← i to j-1
            if (q > m[i,k] + m[k+1,j] + P_{i-1} P_k P_j)
                q ← m[i,k] + m[k+1,j] + P_{i-1} P_k P_j ;
        return q;
}
```

$$T(n) = \sum_{k=1}^{n-1} (T(k) + T(n-k)) + O(n)$$

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

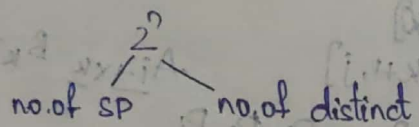
$T(n) > \Omega(2^n)$ — due to overlapping subproblems.

distinct subproblems $= O(n^2)$

- for $i=1$: $j \leftarrow 2$ to n

Max. time taken to solve $m[i,j]$ when $m[i,k]$ and $m[k+1,j]$ are available is $O(n)$.

∴ Max. time taken to solve the problem $= O(n^3)$.



$m[i,j]$ i j

	1	...	n-1	n
1	1,2	...	n-1, n	n-1
2	2,3	...	n-1, n	n-2
⋮				
n	n			1

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

$= \frac{n(n+1)}{2}$ — distinct SP.

Top-Down — Memoization

```
if i==j
    return 0
if m[i,j] ≠ ∞
    return m[i,j]
else
    q = ∞
    for k=i to j-1
        q = min(q, MCM(i,k,p) + MCM(k+1,j,p) + P_{i-1} P_j P_k)
    m[i,j] = q
    return m[i,j]
```

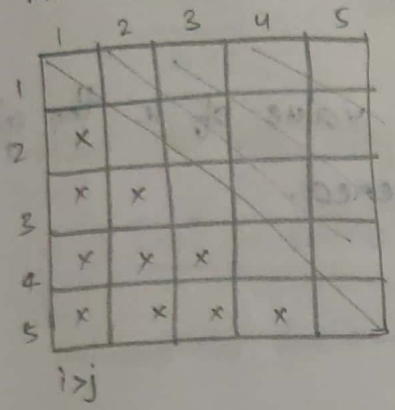
$O(n) \times n^2 = O(n^3)$

Bottom Up (Actual DP)

To calculate length 2, we need to calc. length 1,

for 3, we need 2.

Thus we need to calculate MCM length wise.



Length 1	Length 2	Length 3
1,1	1,2	1,3
2,2	2,3	2,4
3,3	⋮	⋮
⋮	n-1,n	n-2,n

l: length of matrix chains

i = 1 (min)

i = n-l+1 (max)

j = i+l-1 (min)

j = n (max)

BottomUP MCM (p)

1. n = length(p) - 1
2. for i = 1 to n
   M[i][j] = 0
3. for l = 2 to n
   for i = 1 to n-l+1
   j = i+l-1
   M[i][j] = ∞
   for k = i to j-1
   M[i][j] = min(M[i][j], M[i][k] + M[k+1][j] + $P_{i-1} P_k P_j$)

A_1 A_2 A_3 A_4 A_5

5×10 10×15 15×20 20×25 25×30

	1	2	3	4	5
1	0	750			
2		0	3000	8000	15500
3			0	7500	18750
4				0	15000
5					0

24 = 22 + 34

= 23 + 44

Now, we need to store the positions where the split is happening for the order in which matrices need to be multiplied.

∴ if $M[i,j] > q$

$M[i,j] = q$

$s[i,j] = k$ → for which value of k the split has happened.

Book → Source → Target

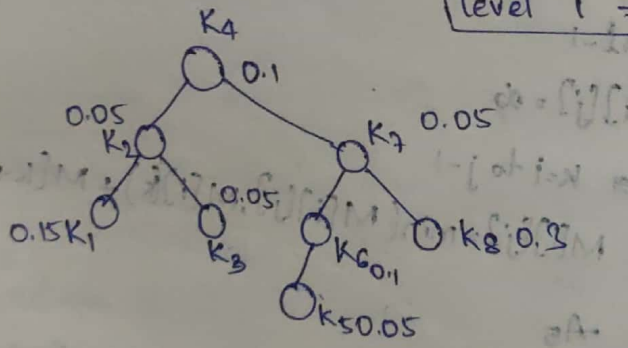
(Lang. in which it is) (to be translated into)

Probabilities of occurenenca – $P_1 \ldots P_n$

(Words) Nodes – $K_1 \ldots K_n$

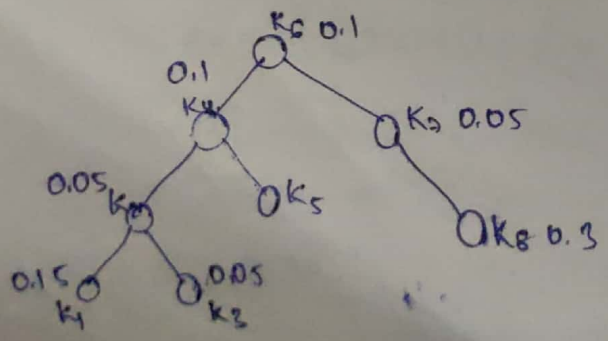
$$\sum_{i=0}^{n} P_i = 1$$

If keys are in BST, no.of comparisions for key in level $i = i+1$.



$\Rightarrow (0.1)(1) + (0.05)(2) + (0.15)(3) + (0.05)(3) + (0.05)(2) + (0.03)(3) + (0.1)(3) + (0.05)(4)$

$$\sum(\text{Search Cost}) = \sum_{i=1}^{n} [\text{level}(K_i) + 1] P_i$$



Optimal BST (OBST):

Brute force: Calculate the cost for all BSTs. The min. value is the required tree.

$\frac{^{2n}C_n}{n+1}$ – total no. of binary trees possible.

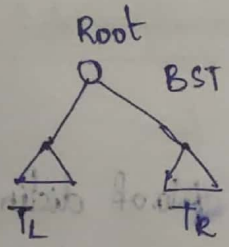
1. How do we say optimal substructure?

Def – same

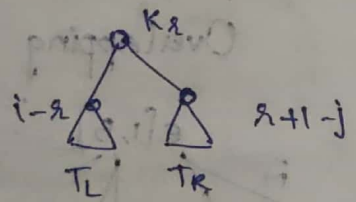
How to show – ① General subproblem

$k_1 < \ldots < k_j \quad (i \le j)$

② Assume there is an optimal solution of the problem considered.



③ Something is part of optimal soln.



④ The optimal solution of subproblem exists in the optimal solution of the problem from which it is derived.

⑤ Thus the problem has optimal substructure property.

$$e[i,j] = e[i, r-1] + e[r+1, j] + P_r + \sum_{i}^{r-1} P_{left} + \sum P_{right}$$

$$= \begin{cases} \min\limits_{i \le r \le j} \left(e[i,r-1] + e[r+1,j] + \sum\limits_{k=i}^{j} p \right) & \text{if } (i \le j) \\ 0 & \text{else.} \end{cases}$$

| 1 × Prob. of key at root

```
OBST(i, j, p){
  if i>j  return 0
  else
    q = ∞
    for int r = i to j
        q = min(q, e[i, r-1] + e[r+1, j])
    e[i,j] = q + Σ_{k=i}^{j} p_k }
```

$T(n) = \Sigma T(i) + T(n-i)$

$= 2 \Sigma T(i)$

$T(n) > 2 \Sigma(\Omega(2^i))$ if $T(n) = \Omega(2^n)$

$> 2 \times (2^n - 1)$

$\boxed{T(n) > C(2^n)}$

Explanation:

No. of SP

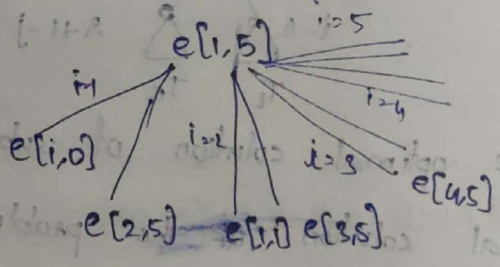
$O(n)$ – because we call 'n' times for any value

No. of distinct SP

${}^nC_2 + n = \Theta(n^2)$

⇓

Overlapping sdn.



Thus we can make

$e[i+1, i] = 0$!!

$O(n) \times n^2 = O(n^3)$

complexity of Eqn. itself

no. of SP

⇒ TopDown ≈ MCMTopDown

BottomUP same as MCMBottomUP (diagonally again)

→ A diff: fn can precalculate $\sum_{k=i}^{j} P_k$ for all values of i & j

26/03/19

X: Management (10)

Y: Cinema (6)

*a*n*t* => a, n, t occurs in this order (need not be adjacent)

Subsequence of X: man
nage
ant, nem ordering is important

Subsequence of Y: cnm
cna
nem.

Goal: Longest Common Sub-sequence of two sequences.

Longest Common Sub-sequence (LCS):

$X = (x_1, x_2, \ldots x_m)$

$Y = (y_1, y_2, \ldots y_n)$

Brute force: Check all subsequences of 'X' and 'Y'.

$X \Rightarrow 2^m$ (x_1 x_2 . . . x_m)
0 0 0
1 1 1 — two possibilities

$Y = 2^n$

Take a subsequence from X and check if it is present in 'Y' or not.

Time complexity = $2^m * n$ (or) $2^n * m$

for every ss check all letters of Y.

$$T(m,n) = O\left(2^{\min(n,m)} \max(n,m)\right)$$

General SubProblem:

Prefix: First Consecutive characters starting from first letter.

Student: S, St, Stu, stud. . . .

Let $X_i = (x_1, x_2 \ldots x_i)$ denote prefix of X whose length is 'i'.

$Y_j = (y_1, y_2 \ldots y_j)$ – prefix of Y of length = j.

→ For $X: (x_0, x_1 \ldots x_n)$ – 'n+1' prefixes exist. (ϕ – also considered)

$LCS(X_i, Y_j) \quad \forall \; 1 \le i \le m$
$1 \le j \le n$

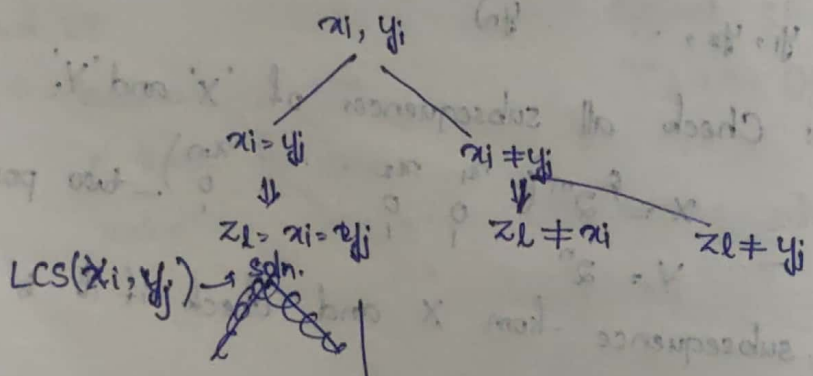
$\Longrightarrow LCS(X_m, Y_n) = LCS(X, Y)$ (Relating SP to original problem)

$LCS(X_i, Y_j)$ – solution has size 'l' $= (z_1, z_2 \ldots z_l)$

$X_i = (x_1, x_2 \ldots x_i)$
$Y_j = (y_1, y_2 \ldots y_j)$

If $x_i = y_j$, then $z_l = x_i = y_j$
if $x_i \neq y_j$

Eg: Slate Home — LCS = e
Slate Some — LCS = Se



$\therefore LCS(X_i, Y_j) = LCS(X_{i-1}, Y_{j-1}) + z_l$

$LCS(X_{i-1}, Y_{j-1})$ – optimal solution is $(z_1, z_2, \ldots z_{l-1})$

↓

If there is not correct, another optimal soln. exists then that becomes $LCS(X_{i-1}, Y_{j-1}) + z_l$. See

Suppose given, $LCS(X_i, Y_j) = 10$

→ $LCS(X_{i-1}, Y_{j-1}) = 9$ – but not true

if $\exists\, LCS(X_{i-1}, Y_{j-1}) = 13$

⇒ $LCS(X_i, Y_j) = 13 + 1 = 14 \neq 10$

⇒ (given wrong)

⇒ $LCS(X_{i-1}, Y_{j-1})$ = ~~given~~ (true)

⇒ Optimal Substructure exists.

⇒ When $z_l \neq x_i$ ⇒ $\{(x_1, x_2, \ldots x_{i-1})$ and $(y_1, y_2 \ldots y_j)\}$
$(z_1, \ldots z_l)$ $LCS(X_{i-1}, Y_j)$

Similarly $z_l \neq y_j$ ⇒ $\{(x_1, x_2 \ldots x_i)$ and $(y_1, y_2 \ldots y_{j-1})\}$
$(z_1, \ldots z_l)$ $LCS(X_i, Y_{j-1})$

The problem has optimal substructure property.

Recursive formula

$$LCS(X_i, Y_j) = LCS(X_{i-1}, Y_{j-1}) + 1, \text{ if } (x_i = y_i)$$

$$= \max(LCS(X_{i-1}, Y_j), \text{ if } (x_i \neq y_j)$$
$$LCS(X_i, Y_{j-1}))$$

$$= 0 \quad \text{if } (i=0) \text{ or } (j=0)$$

$0 \leq i \leq m$
$0 \leq j \leq n$

No.of SP = $(m+1)(n+1)$

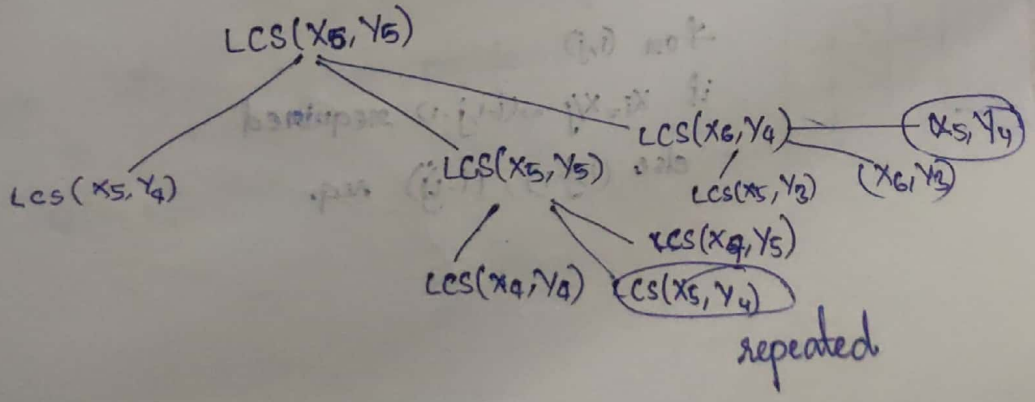
Time for each subproblem = 1 (only addition)

If distinct SP are solved exactly once,

$$T(n, m) = (m+1)(n+1)\,O(1)$$

$$\boxed{T(n,m) = O(mn)}$$

Take some 'i', 'j' – draw recursion tree.



LCS (i, j, X, Y)

if i=0 or j=0 return 0;

else

if (xi = yj) return 1 + LCS(i-1, j-1, X, Y);

else return (max (LCS(i, j-1, X, Y) , LCS(i-1, j, X, Y)))

a d b d a e b aabd

1 2 3 4 5 6 7 aaca

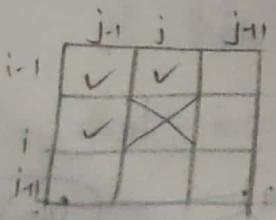
X: a d b d a e b

Y: d b a c b a b

27/03/19

i		0	1 d	2 b	3 a	4 c	5 b	6 a	7 b
	0	0	0	0	0	0	0	0	0
	1 a	0	0	0	1	1	1	1+0	1
	2 d	0	1+0	1	1	1	1	1	1
	3 b	0	1	2+1	2	2	1+1	2	1+1
	4 d	0	1+0	2	2	2	2	2	2
	5 a	0	1	2	1+2	3	3	1+2	3
	6 e	0	1	2	3	3	3	3	3
	7 b	0	1	1+1	3	3	1+3	4	4

X



for (i,j)

if $x_i = x_j$ ⇒ (i-1, j-1) required

else (i, j-1), (i-1, j) req.

Calculate row by row will solve the purpose.

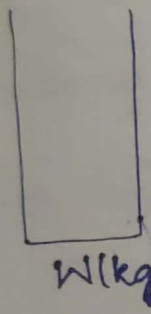
Values in i=0; j=0 = 0.

→ To find longest common palindrome of X:

find LCS (X, rev(X))

0-1 Knapsack problem:

Items	1	2	3	.....	n
Weight (kg)	w_1	w_2	w_3	.....	w_n
value (Rs)	v_1	v_2	v_3	.....	v_n



Maximize ~~weight~~ Value

$x_i = 1$ if i^{th} item is chosen

$= 0$ otherwise.

$$\text{Maximize} \sum_{i=1}^{n} v_i x_i$$

$$\text{S.T} \sum_{i=1}^{n} w_i . x_i \leq W \quad -(1)$$

Brute Force: Check all subsets for condition (1) and then check the maximum value. (2^n possibilities).

General SubProblem:

Knap (i, j) – allowed to pick first 'i' items and

|||

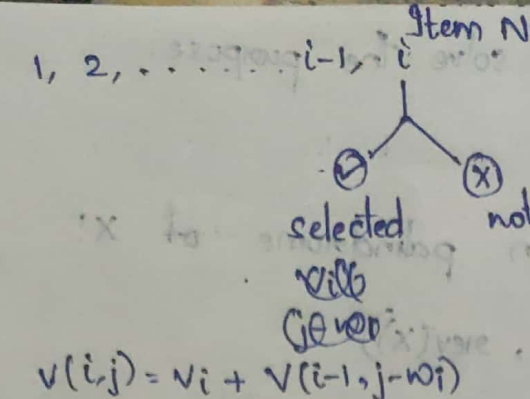
V(i, j) not allowed to touch 'i+1' to n.

and 'j' is remaining weight allowed in bag.

Knap (n, W) – original problem.

|||

V(n, W)



$V(i,j) = V(i-1,j)$

$V(i,j) = v_i + V(i-1, j-w_i)$

Optimal substructure property is present.

$$V(i,j) = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ \max\{v_i + V(i-1,j-w_i),\ V(i-1,j)\} & \text{if } w_i \le j \\ V(i-1,j) & \text{if } w_i > j \end{cases}$$

~~if ith item selected~~

~~if ith item not selected.~~

$V(i,j)$ $0 \le i \le n \Rightarrow n+1$

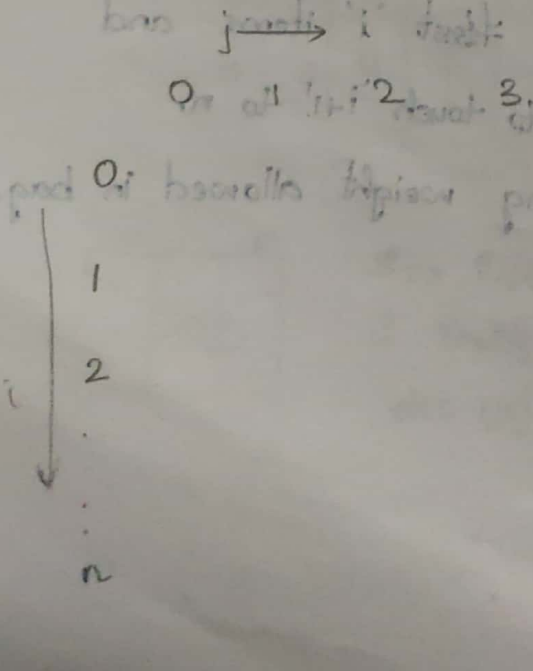
$0 \le j \le W \Rightarrow W+1$

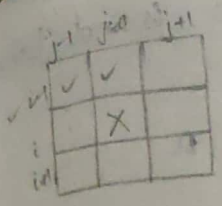
~~#~~ SP $= (n+1)(W+1)$ – distinct SP

$\Theta(nW)$ – # distinct SP

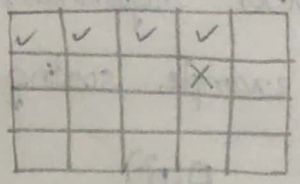
Time taken for each SP $= O(1)$

Overall time $= \boxed{T(n,W) = \Theta(nW)}$





(i,j) =
$(i-1, j)$ neg.
$(i-1, j-w_i)$ – all j



Calculate Row-Wise.

→ It is not polynomial-time algorithm.

'n' – 'n' different values

'W' – only a single value.

Input Size ∞

Prime No. algorithm = $\Theta(m) = 2^m$.

30/08/19

→ Bottom up method – some extra SP solved than TopDown method.

→ For brute force – 2^n SP and max. time in for each

$T(n) = (2^n * n)$ – exponential.

$\Theta(nW)$ – is not polynomial.

Algorithm:

```
IsPrime(n)
for i←2 to n-1
    if n%i=0
        print "Composite"
        return
    ~~else print "Prime"~~
print "Prime"
```

$\Rightarrow T(n) = \Theta(n)$ but not polynomial

Here input size = 1 (value = n)

$\Theta(n)$ – binary representation of 'n' is the input size

m = no.of bits in binary rep. of 'n' = $\lfloor \log_2 n \rfloor + 1$

Running time should be in terms of inputsize = m

$\Theta(n) = O(n \cdot 2^{m-1}) \Rightarrow$ exp

$T(n) = \Theta(n)$; $T(m) = \Theta(2^m) \Rightarrow$ exponential time algo.

→ In other algorithms, binary representation is not taken.

For example, sorting problem:

$n - 1, 2, \ldots$

$O(n^2)$

if 'm' is the largest inputsize of among the 'n' elements

$m = \lfloor \log_2(\text{largest}) \rfloor + 1$

If ① 'n' and 'm' are not dependent:

∵ No.of values and the largest input are independent

$T(n) = O((n*m)^2)$ n- input size

m- max. size of input.

Total i/p size = # inputs * max. size of each

$m = \lfloor \log_2(\text{largest}) \rfloor + 1$

'largest' is constant ⇒ 'm' – constant.

$\Rightarrow O(m*n) = O((c*n)^2) \Rightarrow O(c^2*n^2) = O(n^2)$.

If ② 'n' and 'm' are dependent.

0-1 KnapSack Problem:

$O(nW)$ o: n- i/p size

W- an element.

Searching problem: i/p are 'n' and 'key'

but $T(n) = \theta(n)$ and not dependent on key.

Q $O(nW)$:

Input parameters are: $n, \lfloor \log_2 W \rfloor + 1$

let $m = \log_2 W + 1$

$\Rightarrow W = 2^{m-1}$

Total Input size = ~~n * m~~

$= n * 2^{m-1}$

~~T(n) = O(n)~~

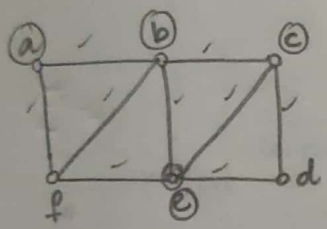
$T(n, m) = (n * 2^m)^{\omega}$ → exponential time

Vertex Cover Problem:

Goal: Let $c \subseteq V(G)$ &

s.t $\forall (u,v) \in E(G)$

$u \in C$ (or) $v \in C$ (or) both



Here $c = \{a, b, c, e\}$

'c' is called "vertex cover".

→ If every edge has atleast one of its end its vertices should be present in vertex cover.

Goal: Find 'c' with minimum size.

Brute Force: List all subsets of vertex set and check if it satisfies the condition and find minimum.

→ Noone has ever given optimal vertex cover solution in polynomial time for general graphs.

If solution for this is found out, many other algo. can be solved.

→ Chromatic Number can be solved for special cases. (Not solved for general graphs - except brute force).

Special Cases: Connected Acyclic graph

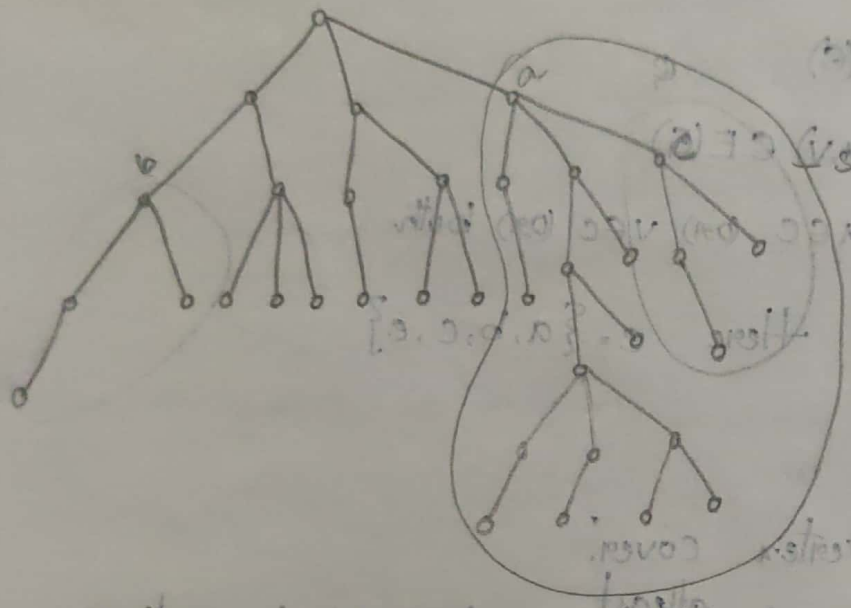
⇓

Tree.

For trees:

Input Size : A tree T:

Output : Optimal vertex cover of T.



General SubProblem:

SubTree rooted at a particular node.

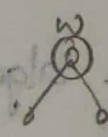
VC(v) – vertex cover of node 'v'.

If optimal soln. to a subproblem is known, soln. to the original problem is known.

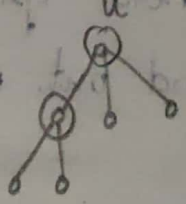
⇓

Optimal Substructure property.

VC(v) – vertex cover of vertex 'v'

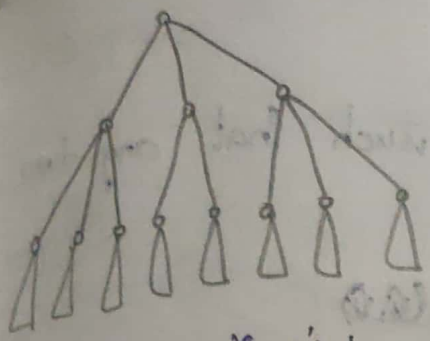


VC(w) = 1



VC(u) = 2

Original SubProblem = VC(root)



Case 1: ~~Root~~ 'v' ~~may~~ part of vertexcover(v).

$$1 + \sum_{u = children(v)} vc(u)$$

Case 2: 'v' not part of vertexcover(v).

$\sum_{u=children(v)} \times vc(u)$ — wrong

if 'u' not part of vc(u)

'u-v' edge not covered.

$\sum_{g=children} 1 + vc$ Let $g \in grandchildren(v)$

~~$\sum_{g} vc(g)$~~

$$|No.of\ children(v)| + \sum_{g \in grandChildren(g)} vc(g)$$

$$vc(v) = \begin{cases} \min \left\{ 1 + \sum_{u=chil(v)} vc(v),\ |\#children(v)| + \sum_{g \in grand(g)} vc(g) \right\} \\ 0, \text{ if 'v' is leafnode} \end{cases}$$

No.of distinct SP = 'n'

Max. time for any SP = O(n) (max - 'n-1' children.

30/4/19.

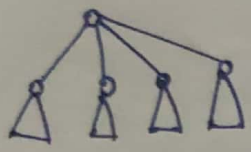
Independent Set:

Subset of vertex set such that any two vertices are not adjacent.

$I \subseteq V(G)$ if $u, v \in I$

$\Rightarrow (u,v) \notin E(G)$

Goal: Find maximum cardinality / size of I - independent set with maximum cardinality is "maximum independent set". (MIS)



If root is included:

$$MIS(v) = 1 + \sum_{g \in grandchil(v)} MIS(g)$$

If not included:

$$MIS(v) = \sum_{w \in child(v)} MIS(w)$$

Here no. of times a subproblem solved = twice.

$2n^2$

Relation b/w problems solved and distinct SP is (n^2)

⇓

both asymptotically same.

$O(n^2)$

$$MIS(v) = \max \left\{ 1 + \sum_{g \in grandchil(v)} MIS(g) , \sum_{w \in child(w)} MIS(w) \right\}$$

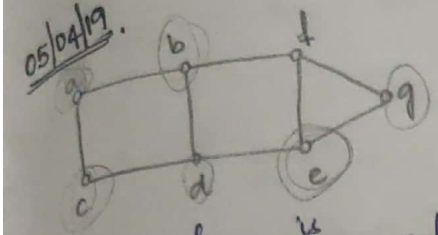
$$= \max \{ MIS^+(v) , MIS^-(v) \}$$

$MIS^+(v) = 1$; $MIS^-(v) = 0$

→ Suppose, if max. weight is asked:

$$\max\left\{ \text{Weight}(v) + \sum_{g \in \text{grandchil}(v)} MIS(g),\ \sum_{u = \text{child}(v)} MIS(u) \right\}$$

05/04/19.



A students ~~are~~ is selected / non-selected.

S.T for any student in non-selected, there exists atleast one friend ∧ student in selected group.

Edge ⇒ friendship

Such sets are $\{a, e\}, \{b, d, g\}$

Minimum of the sets to be found.

Input: $G(V, E)$

$D \subseteq V(G)$ D - Subset of vertices

S.T $\forall\ u \in V(G) \setminus D$

$\exists\ v \in D$ ST $(u, v) \in E(G)$

⇒ 'D' dominates itself and the entire graph.

A vertex 'v' dominates itself and ~~or~~ the adjacent vertices.

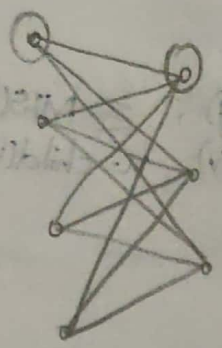
Such set of vertices is called "dominating set"

Goal: Find minimum dominating set (min. cardinality)

Domination Number = Size of min. dominating set.

→ Denoted by $\gamma(G) = \min\{|D| = D \text{ is a DS of } G\}$

Domination Number = 1 when degree of G = n-1.



$\gamma(K_{m,n}) = 2$

↳ Bipartite graph - $m \geq 2$; $n \geq 2$

$\gamma(K_n) = 1$

$\gamma(C_4) = 2$

Brute force:

List all subsets and check for dominating.

(2^n subsets present)

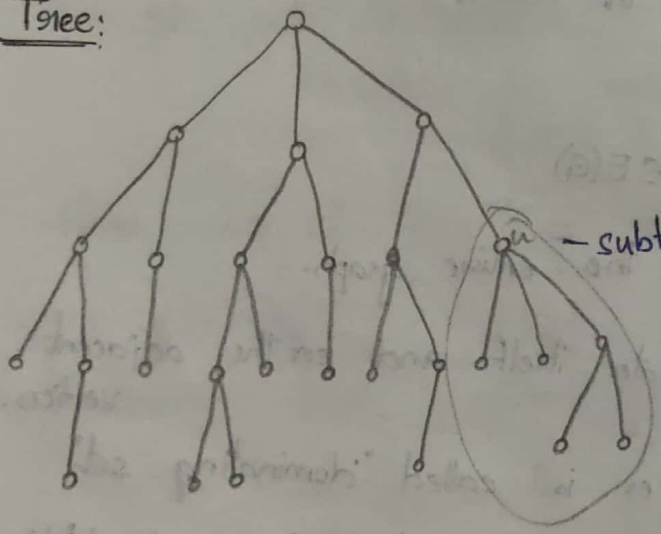
If Take a subset (k) and check if adjacent to all others ($n-k$)

$O(2^n * n^2)$

→ No efficient algorithm to solve for general graphs.

→ Algorithm is given if input is a tree.

Tree:



– subtree rooted at 'u'.

DP(v) – Domination Number of subtree rooted at v.

DP(root) – general problem solution.

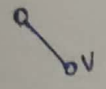
→ If vertex is part, children may also be part of it.

→ If vertex is not part, children cannot be part (all)

To check whether a vertex is dominated at 'v'.

$DS(v)$ – domination no. of subtree rooted at v.

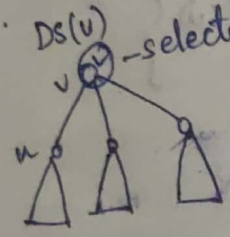
$DS'(v)$ – domination no. of subtree rooted at 'v' where 'v' is already dominated.



$DS(v) = 1$

$DS'(v) = 0$

$DS(leaf) = 1 \;;\; DS'(leaf) = 0$



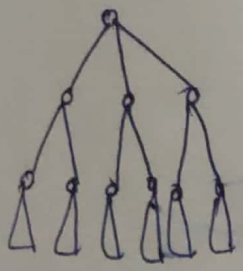
$$DS(v) = 1 + \sum_{u \in child(v)} DS'(u)$$

when 'v' is dominated (selected)

When root 'v' is not selected,

Atleast one of its children should be part.

(? Which one to be included? – many possibilities

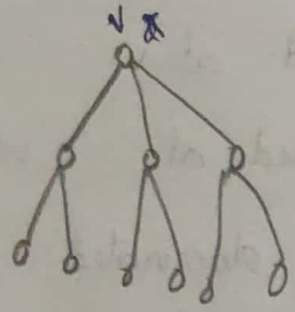


$$\min_{u \in child(v)} \left\{ 1 + \sum_{w \in child(u)} DS'(w) + \sum_{\substack{x \in child(v) \\ \text{and} \\ x \neq u}} DS(x) \right\}$$

$$DS(v) = \min \left\{ 1 + \sum_{u \in child(v)} DS'(v) \;,\; 1 + \sum_{w \in child(u)} DS'(w) \right\}$$

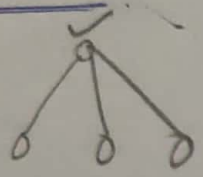
$$DS(v) = \min \left\{ 1 + \sum_{u \in child(v)} DS'(v) \;,\; \min_{u \in child(v)} \left\{ 1 + \sum_{w \in child(u)} DS'(w) + \sum_{\substack{x \in child(v) \\ \text{and} \\ x \neq u}} DS(x) \right\} \right\}$$

$DS'(v)$ — v - not part of answer.



$$DS'(v) = \sum_{u \in child(v)} DS(u)$$

YES Case :



$$DS'(v) = 1 + \sum_{u \in child(v)} DS'(u)$$ (same as yes case of $DS(v)$)

$$DS'(v) = \min \left\{ \sum_{u \in child(v)} DS(u) \;,\; 1 + \sum_{u \in child(v)} DS'(u) \right\}$$

- Actual answer = $DS(root)$

No. of distinct subproblems = $2n-1$ ($DS'(root)$ - not calc.)

$= O(n)$

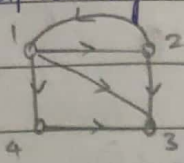
Max. time for any SP =

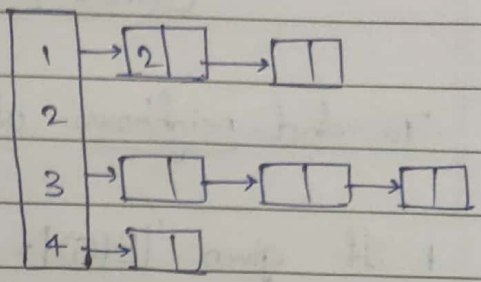
Design and Analysis of Algorithms

10/04/19

Representation of graphs: (Read from Cormen)

1. Adjacency matrix
2. Adjacency list





$A = (a_{ij})_{|V(G)| \times |V(G)|}$

$a_{ij} = \begin{cases} 1, & \text{if } (i,j) \in E(G) \\ 0, & \text{otherwise.} \end{cases}$

'u' and 'v'

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

(for undirected)

Adjacency (checking): $\Theta(1)$ / $O(1)$ (mat.)

Neighbours (finding): $\Theta(n)$ (mat.)

Space: $|V(G)| + 2|E(G)|$ (undirected)

$|V(G)| + |E(G)|$ (directed)

Space = $O(n^2)$ - for directed and undirected.

→ If weighted graph, adjacency matrix is not boolean.

$W = (w_{ij})_{n \times n}$

$w_{ij} = \begin{cases} 0, & \text{if } i=j \\ \text{weight of } (i,j), & \text{if } (i,j) \in E(G) \\ 0, & \text{otherwise.} \end{cases}$

→ In adjacency list, the no. of linked list nodes =

$deg(1) + deg(2) + \dots$

(degree of all vertices)

Degree of all vertices = 2 (Edges)

Space: $|V(G)| + 2|E(G)|$

↓

for every node.

Space = $\Theta(|V(G)| + |E(G)|)$

To check 'u' and 'v' are adjacent or not: $O(d(V(G))$

$O(|V(G)|)$ (or) $O(\Delta(G))$ (or) $O(\max\{deg(u), deg(v)\})$

$O(\Delta(G))$ — max. degree of G

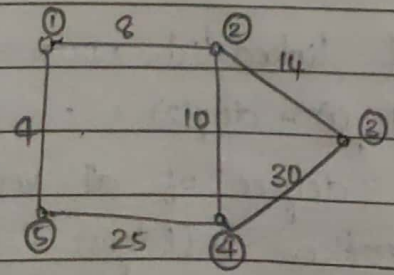
To find neighbours of a vertex 'v': $O(deg(v))$

1. If given $|E(G)| = O(|V(G)|)$ — representing the graph in adjacency list is better.
   * Space is saved.
   Adj. Mat: $O(n^2)$ Adj. list: $O(n+2e) = O(n+n) = O(n)$
   → If edges are less as compared to vertices, it is called 'sparse graph'.
2. If $|E(G)| = O(|V(G)|^2)$ — representation can be done in adj.mat.
   → Such graphs are "dense graphs".

Single Source Shortest Path (Dijkstra's):

Input:
Source vertex(s), $G(V, E, w)$
where $w: E(G) \longrightarrow R^+$; $s \in V(G)$

Out Find: Distance from s to 'u' where $u \in V(G)$



Here, BFS cannot be used.
Since BFS gives $30+25 = 55$
but optimal is $14+8+4 = 26$

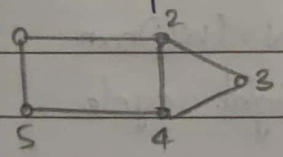
Dijkstra's algorithm: Running time depends on the data structure used to implement priority queue.
If Binary Min. Heap is used, T(n) $\boxed{R.T = O(|E(G)| \log(|V(E)|)}$

If input: Simple, undirected, unweighted graph
Then, instead of dijkstra's do 'BFS'.
BFS: Running Time = $O(|V(G)| + |E(G)|)$

All pairs shortest path problem:
(Find the shortest path between all pairs of vertices)

Input: $G(V,E,W)$ $\boxed{W: E(G) \to R}$ – all '-ve', '+ve', '0'.
Goal: Find $dist(u,v)$ $\forall$ $(u,v) \in E(G)$

If i/p: Simple undirected unweighted graph. ~~and $W: E(G) \to R^+$~~
(Simple – no self loops, no parallel edges).



$|E(G)| = O(|V(G)|)^2$

One method: Run BFS with $\forall v$ diff. sources

Running Time = $O(|V(G)| + |E(G)|) * |V(G)|$
$= O(|V(G)|^2 + |V(G)||E(G)|)$
$= O(|V(G)|^2 + |V(G)|^3)$
$= O(|V(G)|^3)$

If i/p: Simple undirected weighted graph and $w: E(G) \to R^+$

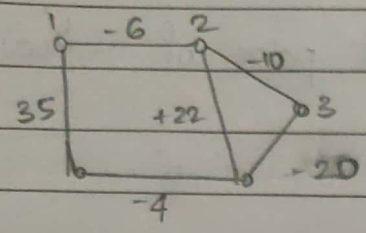
Can be solved using Dijkstra's ($\forall$ diff. sources)
$\forall$ 'v' times run.

$V = |V(G)|$; $E = |E(G)|$

Running Time = $O(|E(G)| \log|V(G)|) * |V(G)|$
$= \boxed{O(V^3 \log V)}$

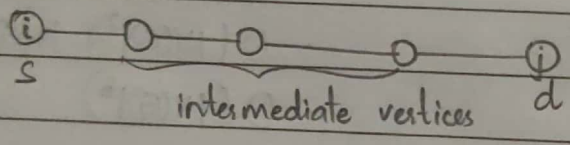
Negative Weight Cycle: A cycle with sum of its edges is negative is called 'negative weight cycle'.

→ Negative weight cycle ⟹ Negative edges
and not vice versa.



Here negative cycle exists
weight = -8
Dist. from (3,3) = -8 (1 cycle)
2nd time = -16; 3rd time = -24
Dist from (1,3) = -16 (1st cycle) (not proper)
2nd cycle = -6-10-8 = -24 (so.. on)

Input: $G(V,E,W)$ $W: E(G) \to R$; $W = (w_{ij})_{n \times n}$
Graph with no negative weight cycles.
1, 2, 3, . . . n, $|V(G)| = n$



Intermediate vertex set: Set of vertices that are allowed to be intermediate vertices.

① Intermediate vertex set is empty: → only edges are considered.

$$d_{ij}^{0} = w_{ij}$$

(distance)

D^0 - matrix containing distance b/w all pairs of vertices with no intermediate vertices.
If no edge — then, infinity.

D^1 - Size of intermediate vertex = 1
{1}

PAGE No.
DATE / /

$\{1,2\}$ - D^2 ; $\{1,2,3\}$ - D^3 ; $\{1,2,3,4\}$ - D^4 ... $\{1 \ldots n\}$ - D^n

12/04/19

Algorithm with Running time = $O(v^3)$ for all pairs shortest path.

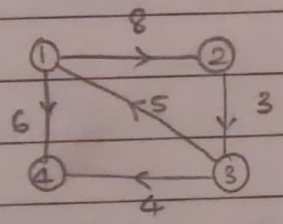
Floyd - Warshall's Algorithm.

Vertices: 1, 2, 3 . . . . , n

Sub-problem: (i) (j) $1 \le i \le j \le n$

If $\{\}$ - intermediate vertex set $\Rightarrow$ Weighted matrix = D^0

d_{ij}^k - shortest path from 'i' to 'j' with intermediate vertex set consisting of first 'k' vertices. $\{1, 2 \ldots k\}$



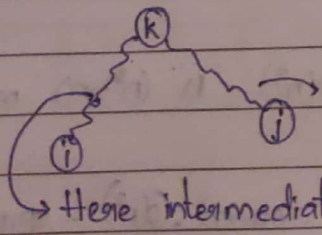
$D^0 = (d_{ij}^0)_{n \times n} = W$; $D^1 = (d_{ij})_{n \times n}$

$D^n = (d_{ij}^n)_{n \times n}$

Sub Problems: (i) (j) Intermediate set $\{1, 2 \ldots k\}$

Two cases: (1) k^{th} vertex is included in the path (i) ... (k) ... (j)
(2) not included (i) ... (j)

Case 1:



Here intermediate vertices = $\{1, \ldots k-1\}$

$$d_{ij}^k = d_{ik}^{k-1} + d_{kj}^{k-1}$$

Case 2: Not include $\Rightarrow$ $d_{ij}^k = d_{ij}^{k-1}$

$$d_{ij}^{k} = \begin{cases} \min \{ d_{ik}^{k-1} + d_{kj}^{k-1} , d_{ij}^{k-1} \} & \text{if } k>0 \\ w(i,i) & \text{if } k=0 \end{cases}$$

For calculating D^k, D^0 to D^{k-1} are needed.

No. of distinct subproblems = d_{ij}^{k} $1 \le i,j,k \le n$

$= n^3$

For each 'k', n^2 entries are required.

Max. time for each subproblem = $O(1)$.

FloydWarshall (i, j, k, G, W)

Algorithm (G, W)

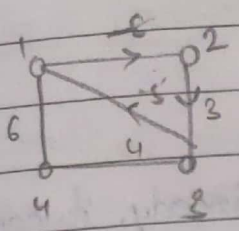
1. $D^0 \leftarrow W$
2. for $k \leftarrow 1$ to n
3. for $i \leftarrow 1$ to n
4. for $j \leftarrow 1$ to n
5. $d_{ij}^{k} = \min (d_{ij}^{k-1} + d_{kj}^{k-1} , d_{ij}^{k-1})$
6. return D^n

Running Time = $O(n^3)$

Space complexity = $O(n^3)$ 'n' $(n \times n)$ matrices.

If matrices are reused, $O(n^2)$ - space complexity.

To check the presence of negative cycles, run this algorithm and if dii - -ve, negative cycle.



$$d^3_{22} = \min\{ d^2_{23} + d^2_{23},\ d^2_{22} \}$$
$$= \min\{ 3 + 3,\ -10 \}$$

$$d^2_{22} = \min\{ d^1_{22} + d^1_{22},\ d^1_{22} \}$$
$$= \min\{ -8 - 8,\ -8 \}$$

$$d^1_{22} = \min\{ d^0_{21} + d^0_{12},\ d^0_{22} \}$$
$$= \min\{ 8, -8, 0 \}$$

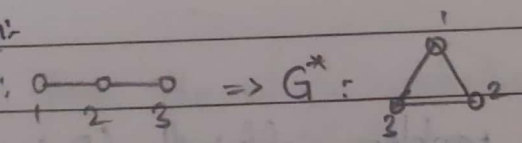
Transitive Closure:

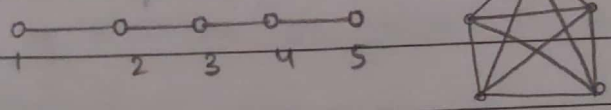
Let $G^*(V,E)$ be $V(G^*) = V(G)$

$E(G^*) = \{(u,v) : u \rightsquigarrow v \text{ in } G\}$

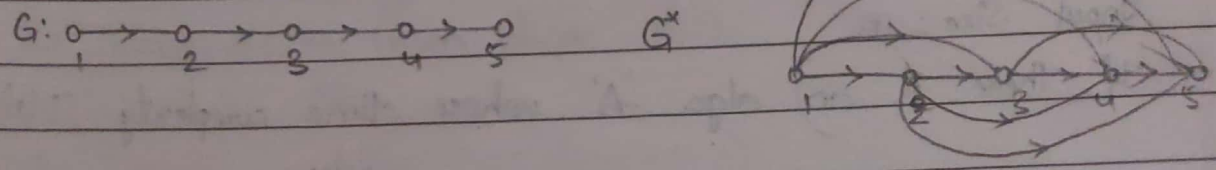
(if any path present, becomes edge in G^* in G)

Ex:





For undirected connected graph is given, transitive closure is complete graph.



Let t^k_{ij} (boolean) which gives whether 'i' and 'j' are connected or not (path exists) with 'k' intermediate vertices.

$t_{ij}^{k} = (t_{ik}^{k-1} \wedge t_{kj}^{k-1}) \vee t_{ij}^{k-1}$ (Faster)

$\because$ Only boolean

Running Time : $O(n^3)$

NOTE: If a problem is solved using Greedy, it can be solved using DP as well. (converse - not true)

Greedy	DP
→ Greedy has optimal substructure.	→ DP has optimal substructure.
→ We decide choice, based on that problem is solved.	→ After solving all SP, we make a choice.

→ In DP, all possible soln's approachable ways are checked but in greedy, we fix a path based on some logic.

5/04/19

Is it possible to solve a problem efficiently (Prob. complexity)?

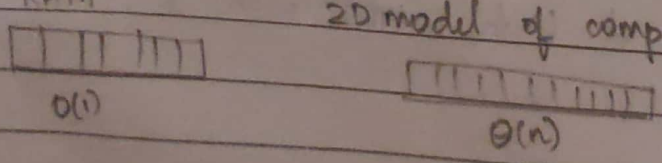
Complexity Classes:

Polynomial Time Solvable / Class P:

Input Size : n

If there is an algo. 'A' whose time complexity $T(n) = \Theta(n^c)$ where c - constant.

then, the problem is polynomial time solvable and efficient to solve or not difficult to solve.



PAGE No.
DATE / /

Optimization Version:

Clique Problem:

Input: A simple undirected graph G

Output: $\omega(G)$
Size of max. clique in G.

⇓

Clique Decision Problem (CDP): (Transform prob.)

Input: A simple undirected graph G and an integer K.
Output: Does the graph have a clique of size greater than k?

[Decision prob: Prob. for which answer is YES/NO.
Clique: Complete subgraph of a graph]

Time to check whether 'YES' answer is correct or not
$O(k^2)$
'k' - can take max. 'n' → $O(n^2)$

Input: G
Does G have a clique of size
Q1: ≥ n → Yes (max clique = n)
No
Q2: ≥ n-1 → Yes (max. clique = n-1)
No
Q3: ≥ n-2 → Yes (⇒ max. clique = n-2)
No

→ An optimisation problem can be solved by solving a series of decision problems.

Class NP (Non-deterministic Polynomial):

Only dealing with 'YES' cases.

When problem 'X' is decision problem and if answer is 'Yes' and 'certificate' is 'Y', then to verify 'Y' and 'X', time taken should be polynomial.

⭐ A decision problem 'X' is in NP if there exists a polynomial time algorithm A(x,y) such that for every input $x \rightarrow X$ instance

X(x) = 1 iff there exists some 'y' such that A(x,y) = 1, then

'y' is called a "witness" or "certificate", the algo. 'A' is called a "verifier" (or) a "non-deterministic algorithm".

$P \subseteq NP$

P class

NP class

- KnapSack
- Vertex Covers
- Independent Set of size
- Chromatic Number

But, if a problem is 'NP', then we don't know whether it is in 'P' or not.

HW: Write problems in NP which are not in P.

22/04/19. Let 'A' be a algorithm to solve 'X' problem and 'n' is input size and running time is cn^a (polynomial), then 'X' is in 'P'.

NP - Non Deterministic Polynomial.

'X' problem (decision)

'x' - instance of problem.

X(x) : Yes / No.

If $X(x)$ = yes, an algorithm $A(x, y)$ = yes
given

If the algorithm takes polynomial time in input size, then 'X' is in 'NP'.

$A(x, y) = O(x \times y)$

A - polynomial in 'x' and 'y'.

If 'y' is exponential, then 'X' is not in 'NP'.

* If there exists a polynomial time verification algorithm for 'X', then 'X' is in 'NP' with a certificate 'y' (if answer is 'YES')

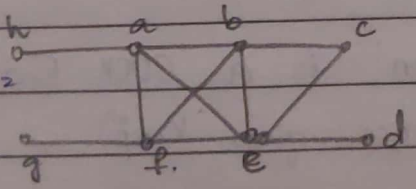
* For problems 'X' is in 'P', verification algorithm is also polynomial.

$\therefore P \subseteq NP$.

For problems in NP with no polynomial time algorithm, checking running time for verification problem algorithm in polynomial time. If yes then problem is in 'NP'.

CDP:

Input: G=



and k=3

Output is 'Yes' with {a,c,e} as certificate (Y)

To verify this, the algorithm should be polynomial time.

NOTE:

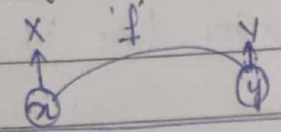
→ If decision prob version algorithm 'not polynomial'
⇒ optimisation - 'not polynomial'.

→ If optimisation — 'polynomial' ⇒ decision - 'problem'.

DATE / /

Polynomial Reduction:

If 'x' is instance of 'X', 'y' instance of 'Y', relating the problems with 'f' (polynomial time).



$X(x) = 1$ iff $Y(f(x)) = 1$

⇒ 'x' becomes $f(x)$ after applying function
'$f(x)$' is input to 'Y'.
'x' is i/p to 'X' ⇒ '$f(x)$' is i/p to 'Y'

⇒ Bijection function.

If there exists a polynomial time algorithm to solve 'Y', then 'x' can also be found in polynomial time (f).

⇒ If 'Y' can be solved, then 'X' can also be solved.
⇒ 'Y' problem handling is difficult than handling 'X'.

NOTE: 'Y' is atleast as difficult as 'X'.

⇒ This reduction is known as "Karp Reduction".

'COOK Reduction'— Running algorithm for 'Y' any number of times to solve 'X'.

NOTE: Every 'Karp Reduction' is a 'COOK-Reduction' but not vice-versa. (Solving 'cook' only once, gives 'Karp').

If 'X' is a set of 'NP' problems and 'Y' is a problem, then handling 'Y' is as hard as handling 'NP' problems (X). It is called "NP-hard" (Y)

⇒ If solution to 'Y' is given, 'X' (set of NP) solution can be given.

⇒ A problem (Y) is said to be NP-hard, if all problems in set NP are polynomially reduced to the problem (Y)

→ The first problem to be proved as 'NP-hard' is 'SATISFIABILITY PROBLEM'. (By Stefen Cookin 1971) Turing Award

Let x_1, x_2, x_3, $\alpha = (x_1 \vee \bar{x}_2) \wedge (x_3 \rightarrow x_1)$

for $x_1 = T$ α = True

$x_2 = T/F$

$x_3 = T/F$

For any entry for x_1, x_2, x_3, if α = True ⇒ Totology

* For atleast one entry of x_1, x_2, x_3, if α = True ⇒ Satisfiable.

Contingency - (neither totology nor contradiction)

Satisfiability problem:

Input: Proposition variables and formula.

α: contains $\wedge, \vee, \sim$ (any operation can be reducable)

Output: Satisfiability yes/No.

→ Satisfiability problem is in 'NP' as if o/p is 'YES', verification can be done in polynomial time.

→ If a problem is in 'NP' and 'NP-hard', then it is in 'NP-complete'.

→ Instead of transforming all problems from NP to problem(Z), by transforming SATISFIABILITY problem to 'z', then 'z' verifying properties of NP-hard, 'z' can be said to be 'NP-hard'. (Any problem proved to be NP-hard can be used instead of Satisfiability problem). -(Transitive ~~Problems~~ Property)

→ Next year (1972) 21 other problems are also shown to be NP-hard. (Richard)

→ 1000s of problems are proven to be NP-complete.

5/04/19.

SAT:

Input: Propositional Variables: $x_1, x_2 \ldots x_n$; $\alpha = \ldots$

Q: Is 'α' satisfiable?

Eg:- x_1, x_2, x_3 : $\alpha = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2) \wedge (x_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$

This problem is Product Of Sums.

It is in 'Conjunctive Normal Form' (CNF)

$$\alpha = \bigwedge_{i=1}^{q} C_i \qquad C_i = \bigvee_{s=1}^{?} l_s$$

C_i → Clause

l_s - literal (either a propositional variable or its compliment)

then α is in CNF.

Eg:- $x_1 \wedge (x_2 \vee x_3)$ ✓ CNF ; $x_1 \vee (\bar{x}_2 \wedge x_3)$ ✗

$(x_1 \vee \bar{x}_2) \wedge (x_1 \vee x_3)$ ✓ CNF

→ Any propositional logic can be transformed into CNF.

CNF - conjunction of clauses.

* CNF is true when all its clauses are true
* Clause is true when one of its literal is true

→ If a propositional logic (α) is in CNF form the satisfiability ~~propositional~~ problem, this problem is called as **CNF-SAT**.
(Restricted form of SAT)

→ CNF-SAT is also **NP-Complete**.

→ If initially CNF-SAT is proven to be NP Complete (i.e., NP-hard), then the general version SAT is also NP-complete (since as hard as CNF-SAT)

PAGE NO.
DATE / /

CDP (Clique Decision Problem):

Input: A simple undirected graph G and an integer 'k'.

Q: Is $\omega(G) \geq k$?

Theorem: CDP is NP-complete

Proof: CDP $\in$ NP ✓? (Verification can be done in polynomial time) $O(e^2)$

and CDP $\in$ NP-hard ?

To prove this, chose a problem already proven to be NP-hard (SATISFIABILITY (or) CNF-SAT) and transform that problem into CDP. (reduce)

CNF-SAT $\propto$ CDP

i/p: PV's, B formula i/p: Graph, integer

$\alpha = \bigwedge_{i=1}^{q} C_i$

Reducing CNF-SAT to CDP

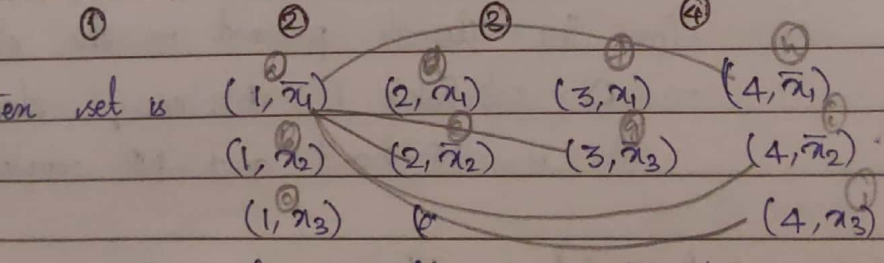
☞ Number all clauses

Each Vertex Set is ordered pair containing (Clause No., Literal in class)

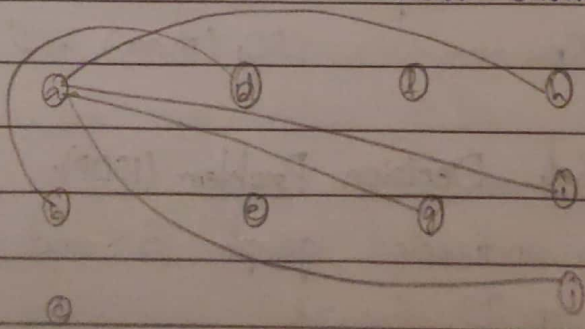
Eg: $\alpha = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$

(1) (2) (3) (4)

Now vertex set is $(1, \bar{x}_1)$ $(2, x_1)$ $(3, x_1)$ $(4, \bar{x}_1)$ $(1, x_2)$ $(2, \bar{x}_2)$ $(3, x_3)$ $(4, \bar{x}_2)$ $(1, x_3)$ $(4, x_3)$



There is an edge if i) Diff clause Number and Literal should not be compliment of each other.



→ It is possible to perform this in polynomial time.

DATE

Claim: α is satisfiable iff G has a clique of some size $(l) \geq ?$.

$P \leftrightarrow Q \equiv (P \rightarrow Q) \wedge (Q \rightarrow P)$ (size - not known)

Claim:

$P \rightarrow Q$: If 'α' is satisfiable, then 'G' has a clique of size $\geq ?$.

⇓

all clauses are true

For all clauses to be true, atleast one literal in every clause is true.

Size = No. of clauses $= g$

If x_3 in ① is True then $(1, x_3)$ part of clique

x_1 in ② $= T$ $(2, x_1)$ "

$\bar{x}_1$ in ③ $= T$ $(3, \bar{x}_1)$ "

$\bar{x}_2$ in ④ $= T$ $(4, \bar{x}_2)$ "

Because they are adjacent (∵ x_1 and $\bar{x}_1$ not True at a time)

$Q \rightarrow P$: If 'G' has a clique of size $\geq g$, then 'α' is satisfiable.

⇒ 'g' vertices form clique.

For all PV's truth value has to be given.

Given a clique of size 'g' →

for the literals present in the clique, make them 'true'

⇒ α - satisfiable (∵ for every clause, literal is true)

It is NP-hard and NP-complete.

26/08/19. Independent Set problem (ISP):

i/p: G ; o/p: Size of max. IS of G

To check if every size is possible or not as IS

$^nC_n + {}^nC_{n-1} + \cdots + {}^nC_1$ (max.) $= 2^n$

Independent Set Decision Problem (ISDP):

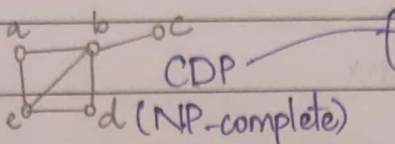
i/p: A simple, undirected graph G and an integer 'k'.

Q: G has an IS size $\geq k$?

Theorem: ISDP is NP-complete

ISDP is in NP ✓ (Check all possible sets of ^nC_k and if yes Checking can be done in polynomial time then ✓ done)

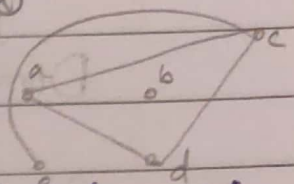
ISDP ∈ NP-hard?



i/p: G and k

Q: ω(G) ≥ k

i/p: H = $\bar{G}$



* If a set is a clique in G, then it is independent set in $\bar{G}$.

∴ i/p to ISDP is $\bar{G}$ and 'k' (since from the above property)

Claim: G has a clique of size ≥k iff $\bar{G}$ has an IS of size ≥k

Input to CDP is 'x'; Input to ISDP is 'f(x)'.

(P → Q) ∧ (Q → P)

The problem ISDP is in NP and NP-hard ⇒ NP-complete.

→ Vertex Cover Problem is not in NP. (∵ if given k check all ≤k)

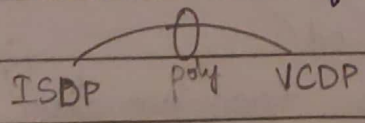
If k = n ⇒ $^nC_n + ^nC_{n-1} + \dots + ^nC_0 = 2^n$

Vertex Cover Decision Problem (or) Node Cover Decision problem:

i/p: Simple, undirected graph and integer 'k'

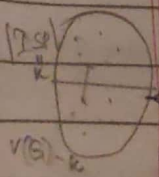
Q: Does G have a VC of size ≤k?

Theorem:



i/p to ISDP: G and k

i/p to VCDP: G and n-k



[edges exist b/w upper to lower (or) b/w lowers)
↳ ⇒ |Vertex Cover| can be |(V(G) - k)|]

Claim: G has a IS of size ≥k and G has a VC of size ≤n-k

R → Q: No edges b/w vertices of IS

So edges for IS are covered in vertices in VC

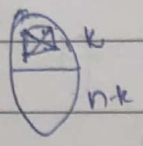
(Similarly Q → P)

for CDP to VCDP

i/p: G and k i/p: $\bar{G}$ and n-k

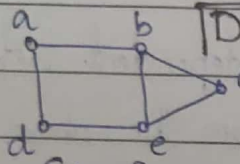
Claim: G has a clique of size $\geq k$ iff $\bar{G}$ has $|VC| \leq n-k$

$P \rightarrow Q \wedge Q \rightarrow P$



Dominating Set: (Minimization problem)

Find Domination Number (minimum size of DS)



{b, e}

DSDP:

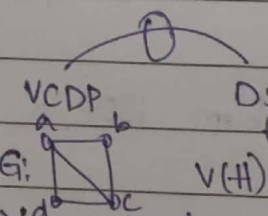
i/p: simple undirected graph G and integer 'l'.

Q: DS of size $\leq l$?

Theorem: DSDP is NP-complete

Proof: DSDP $\in$ NP-hard?

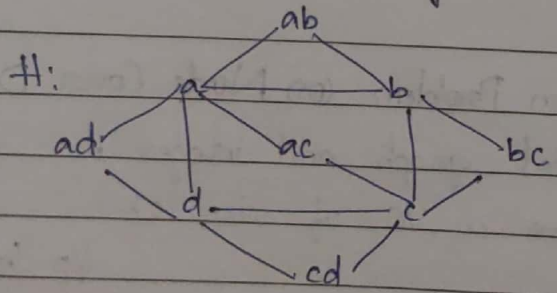
DSDP $\in$ NP ✓



$V(H) = V(G) \cup E(G)$

No. of vertices in H = $|V(G)|$ and $|E(G)|$

Edge set of H is {All edges in 'G' + edge vertices adjacent to its end vertices}



Edges in H = 3 * V(G)

$\Rightarrow$ Polynomial time transformation is possible.

Claim: G has a VC of size $\leq k$ iff. H has a DC of size $\leq k$

* Every Vertex Cover is a Dominating Set

P→Q: If a set is vertex cover of G, then it is dominating set of H (∵ new vertices added also dominated)

If we say that new edge vertices are not dominated in H, this means that that edge not covered in G.

Q→P: If dominating set given, then it is vertex cover of G. But if {a, c, bc} (DS of H) not VC of G. In such cases 'bc' is replaced with 'b' or 'c' and made VC of G (True).