

Analysis and Design of Algorithms
Subject Code : MCC301

Text Book: Design and Analysis of Algorithms by
 Thomas H. Cormen.

Pre-requisites:

- i) Programming
- ii) Data Structure
- iii) Discrete Mathematics

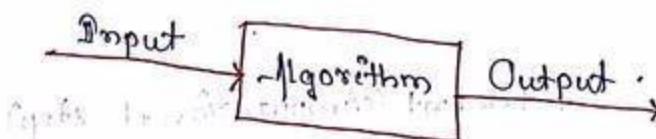
Introduction

Algorithm

It is a step-by-step process to accomplish a particular task.

Defn

An algorithm is any well-defined computational procedure that takes some input or a set of input and produces some output or a set of output. (Cormen).



In algorithm is a sequence of computation steps that transforms the input into output.

(Cormen).

Defn) In algorithm is a finite sequence of computational steps that accomplishes some task or that accomplishes a particular task.

*- In algorithm must contain finite number of statements and the statements (computational steps) must be well-defined.

Properties of an algorithm

- ① Input.
- ② Output.
- ③ Definiteness: All the computational steps must be well-defined (clear and unambiguous).
- ④ Finiteness: It must contain finite number of computational steps.
- ⑤ Effectiveness: The statement (computational step) must be carried out by pencil and paper i.e., the operation defined in

2

the computational step should be feasible operation.

Sorting Problem

Input: A sequence of numbers $\langle a_1, a_2, \dots, a_n \rangle$.

Output: A permutation of the input sequence $\langle a'_1, a'_2, \dots, a'_n \rangle$ such that $a'_1 \leq a'_2 \leq \dots \leq a'_n$.

Ex Input: $\langle 91, 82, 63, 9, 56 \rangle$.

Output: $\langle 9, 56, 63, 82, 91 \rangle$.

Instance of the input sequence.

Instance of the problem.

Correct Algorithm

An algorithm is said to be correct if it halts with correct output for all instances of a L problem.

* - A correct algorithm solves a computational problem.

Correctness Proof

* - Mathematically, the algorithms have to be proved as correct.

Design

- * Well-defined design techniques are used for each algorithm or to design algorithms.

Analysis

To design fast algorithms on the basis of efficiency.

Problem

- * Given two numbers m and n .
Find their greatest common divisor (GCD).

A:

Input: Two numbers m and n .

Output: $\text{GCD}(m, n)$.

Algorithm: Factoring method.

Algorithm 2: Euclid's method.

Algorithm 1:

1 Factorise m : Find prime m_1, m_2, \dots, m_k such that
 $m = m_1 * m_2 * \dots * m_k$.

2 Factorise n : Find prime n_1, n_2, \dots, n_l such that
 $n = n_1 * n_2 * \dots * n_l$.

15/1/2015

3: Find common factors.

Multiply them and find the GCD.

simple statement
loop
conditional
statement.

Algorithm 2:

1: Check if m divides n.

2: If m divides n, return m as GCD.

3: else.

4: while(m doesn't divide n)

5: Find remainder (r) = n mod m.

6: n = m

7: m = r

8: end of while.

9: return m as GCD.

Pseudocode

Procedure:

$$m = 36, n = 48$$

$$\begin{array}{r} 2 \mid 36 \\ 2 \mid 18 \\ 3 \mid 9 \\ 3 \mid 3 \\ 0 \end{array}$$

$$\begin{array}{r} 2 \mid 48 \\ 2 \mid 24 \\ 2 \mid 12 \\ 2 \mid 6 \\ 3 \mid 3 \\ 0 \end{array}$$

Algorithm 1:

$$m = 2 \times 2 \times 3 \times 3$$

$$n = 2 \times 2 \times 2 \times 2 \times 3$$

Common factors: 2, 2, 3.

$$\text{GCD} = 2 \times 2 \times 3 = 12$$

No. of operations
by Algorithm 1 = 14.

Algorithm 2:

1.

$$\begin{array}{r} 36 \mid 48 \\ 36 \end{array}$$

$$\frac{36}{72}$$

Iteration

$$\begin{array}{r} \underline{m} \\ 36 \\ \underline{n} \\ 18 \end{array}$$

Step-1.

$n \% m$

Iteration

$$\begin{array}{r} \underline{m} \\ 12 \\ \underline{n} \\ 36 \end{array}$$

Step-1.

$n \% m$

= 12

$n = 36, m = 12,$

LectureNotes.in

$36 \% 12$ (divides)

$GCD = 12.$

No. of operations by
Algorithm 2 = 3

* We compare algorithms on the basis of
the running time of an algorithm.

→ Number of primitive operations (conditions, mathematical operations) or step executed to solve the problem.

Instance:

$$m = 434, n = 966$$

Algorithm 1 :

$$m = 2 \times 7 \times 31$$

$$n = 2 \times 7 \times 139$$

Common factors : 2, 7.

$$GCD = 14 \quad \text{No. of operations} = 7$$

Algorithm 2 :

Iteration

$$\begin{array}{r} \underline{m} \\ 434 \\ \underline{n} \\ 966 \end{array}$$

Step-1.

$n \% m$

(not divides)

Step-1.

$$91 = 966 \% 434 = 98$$

98

98

484

98×5

$m = 98$

$$184 = 98 \times 4 + 42$$

2

$$42 \div 98$$

$$m = 484 \% 98$$

$$98 = 2 \times 42 + 14$$

$$14 \div 42$$

≈ 42

$$m = 98 \% 14$$

≈ 14

$$98 = 7 \times 14 + 0$$

$$\text{GCD} = 14$$

$$m = 42 \% 14$$

$= 0$ (divides)

$$\text{No. of operations} = 5$$

*— The running time and also depends on the inputs.

the time complexity instances for the

Analysis : Running time of an algorithm.

Running time : No. of primitive steps executed to complete the task.

Given an algorithm how to compute the running time.

Ex: Insertion sort

31, 9, 10, 27, 85
1st 2nd

with

9 31 10 27 85

3rd

9 10 31 27 85

9 10 31 27 85

compare

to sort

9 10 27 31 85

10	9	8	7	6	5	4
10	9	8	7	6	5	4

1) $i = 2 \dots m$ (or 1 is sorted)

Key = $A[2]$ element to be sorted.

If $A[2] < \text{key}$, then the algorithm assumes the array elements $[A[0..(j-1)]$ is already sorted.

2) $i = 1 \dots n-1$ (or 1 is sorted)

$A[1..j]$ is sorted.

Key = $A[2]$

i start for $A[1..j]$.

if $A[i] > \text{key}$

$A[i+1] = A[i]$

$i = i + 1$

Inversion Sort (I)

1) For $j = 2$ to $\text{length}(A)$,

n times (termination of loop for checking n).

2) $\text{key} = A[2]$

(n-1)

3) $i = j - 1$

(n-1)

4) While ($i > 0$ $\&$ $A[i] > \text{key}$), $A[i+1] = A[i]$, $i = i - 1$

$\sum_{j=2}^i (t_j - 1)$

$i = i - 1$

$\sum_{j=2}^i (t_j - 1)$

5) end of while.

6) $A[1..j] = \text{key}$.

(n-1)

7) end.

Let the 4th statement will execute t_j times - for a value of j .

Total time taken by 4th statement = $\sum_{j=2}^n t_j$.

Algorithm (Insertion Sort)

Insertion sort (A)

```

1. For j=2 to length(A)
2.   Key = A[j]
3.   i=j-1
4.   while(i>0 & A[i]>Key)
5.     A[i+1] = A[i]
6.     i=i-1
7. end of while
8. A[i+1]=key
  
```

CostTime

$$\begin{aligned}
 C_1 &= n-1 \\
 C_2 &= \sum_{j=2}^n t_j - 1 \\
 C_3 &= n-1 \\
 C_4 &= \sum_{j=2}^n t_j - 1 \\
 C_5 &= \sum_{j=2}^n (t_j - 1) \\
 C_6 &= \sum_{j=2}^n (t_j - 1) \\
 C_7 &= n-1
 \end{aligned}$$

Total time taken by insertion sort:

$$\begin{aligned}
 T(n) = & C_1 n + C_2(n-1) + C_3(n-1) + C_4 \sum_{j=2}^n t_j + C_5 \sum_{j=2}^n (t_j - 1) \\
 & + C_6 \sum_{j=2}^n (t_j - 1) + C_7(n-1)
 \end{aligned}$$

There are 3 diff. states for time complexity:

(i) Best case : sorted input

(ii) Worst case : input in reverse order

(iii) Average case : not completely sorted not reverse.

The state depends on the type of input.

Best Case Time Complexity.

Input is in sorted order.

$$T(n) = C_1 n + C_2(n-1) + C_3(n-1) + C_4(n-1) + C_5 \times 0 + C_6 \times 0 + C_7(n-1)$$

$$\therefore (C_1 + C_2 + C_3 + C_4 + C_7)n - (C_2 + C_3 + C_4 + C_7).$$

$$\text{Let } C_1 + C_2 + C_3 + C_4 + C_7 = a,$$

$$-C_2 - C_3 - C_4 - C_7 = b$$

$$\Rightarrow T(n) = an + b$$

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

Worst Case Time Complexity.

Input is in reverse order.

$$T(n) = C_1 n + C_2(n-1) + C_3(n-1) + C_4\left(\frac{n(n-1)}{2}\right)$$

$$+ C_5\left(\frac{n(n-1)}{2}\right)$$

$$+ C_6\left(\frac{n(n-1)}{2}\right) + C_8n - C_8.$$

$$\sum_{j=2}^n (t_j)$$

$$= 2 + 3 + 4 + \dots + n$$

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

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

$$= 1 + 2 + 3 + 4 + \dots + (n-1)$$

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

$$\Rightarrow T(n) = (C_1 + C_2 + C_3 + \frac{1}{2}C_4 + \frac{1}{2}C_5 + \frac{1}{2}C_6)n + (\frac{1}{2}C_4 + \frac{1}{2}C_5 + \frac{1}{2}C_6)n^2$$

$$-(C_2 + C_3 + C_4 + C_8).$$

Worst case

$$\text{Let } a = \frac{1}{2}C_4 + \frac{1}{2}C_5 + \frac{1}{2}C_6, b = C_1 + C_2 + C_3 + \frac{1}{2}C_4 + \frac{1}{2}C_5 + \frac{1}{2}C_6,$$

$$c = -(C_2 + C_3 + C_4 + C_8).$$

$$\Rightarrow T(n) = an^2 + bn + c$$

$$\Rightarrow T(n) = O(n^2)$$

$n \rightarrow$ linear

$n^2 \rightarrow$ quadratic

Growth of functions

Running time / execution time

$T(n) \rightarrow$ function of input size.

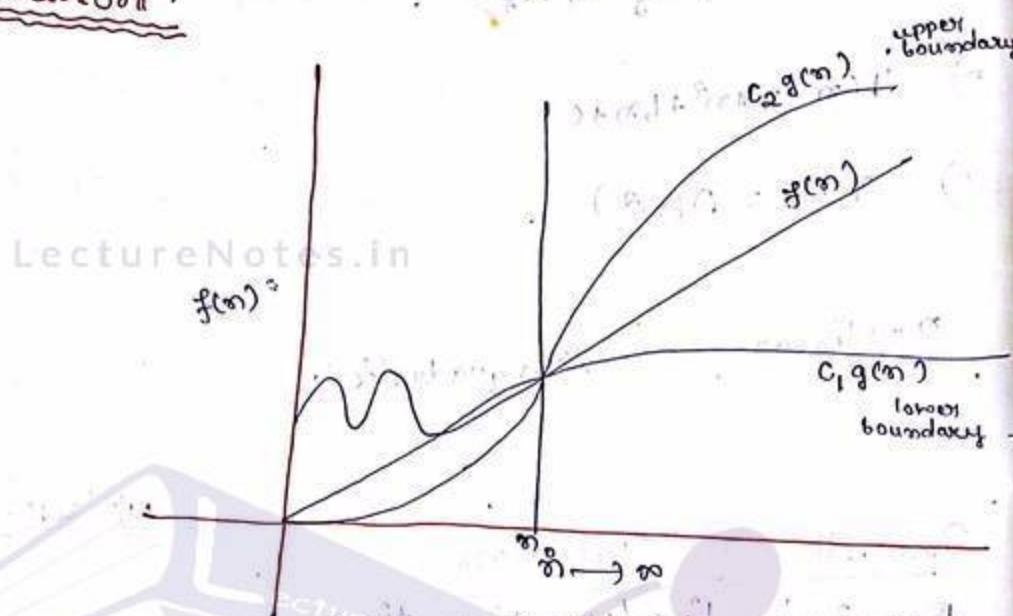
$n \rightarrow$ input size.

- * When $n \rightarrow \infty$ becomes very large, the behaviour of the running time is characterised by asymptotic notations.
- * Asymptotic behaviour of execution time means with large n how the function will grow.
- * Asymptotic notations are used to characterize the asymptotic growth of the functions.
- * There are 5 asymptotic notations:
 - (i) $\Theta(\text{Theta})$ notation.
 - (ii) $O(\text{Big-O})$ notation.
 - (iii) $\Omega(\text{Omega})$ notation.

iv) $O(\text{little oh})$ notation.

v) $\omega(\text{little omega})$ notation.

Θ -Notation



For a given function $g(n)$, $\Theta(g(n))$ gives a set of functions which we can write

$\Theta(g(n)) = \{f(n)\}$. If there exist positive constants c_1, c_2 and n_0 such that $0 < c_1 g(n) < f(n) < c_2 g(n)$ for all $n \geq n_0$.

*- Θ -notation says that for some positive constants c_1 and c_2 , $f(n)$ is sandwiched. but $c_1 g(n) < c_2 g(n)$ for $n \geq n_0$.
 Θ -notation gives the lower and upper boundary of the funⁿ $g(n)$ when $n \rightarrow \infty$

It gives asymptotic tight boundary if the higher order terms are negligible.

Prove that $\frac{1}{2}n^2 - 3n = O(n^2)$

$c_1 g(n) \leq f(n) - 0 \quad f(n) \leq c_2 g(n) \rightarrow$
 $c_1 g(n) \leq f(n) \rightarrow c_1 n^2 \leq \frac{n^2}{2} - 8n$
 $c_1 \leq \frac{1}{2} - \frac{8}{n}$.
 Put n .

Proof:
 $f(n) = \frac{1}{2}n^2 - 3n$
 $g(n) = n^2$, \therefore
 Find c_1, c_2, n_0 so that
 $0 \leq c_1 n^2 \leq \frac{n^2}{2} - 3n \leq c_2 n^2 \forall n \geq n_0$.

Divide the eqⁿ by n^2 , \therefore $0 \leq c_1 \leq \frac{1}{2} - \frac{3}{n}$ $\leq c_2$. \rightarrow (1)

$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$.
 consider this

$c_1 \leq \frac{1}{2} - \frac{3}{n}$
 $\frac{1}{2} - \frac{3}{n} \leq c_2$

For c_1 :
 $n=1, \frac{1}{2} - \frac{3}{1} = -\frac{5}{2}$
 $n=2, \frac{1}{2} - \frac{3}{2} = -1$
 $n=3, \frac{1}{2} - \frac{1}{2} = -\frac{1}{2}$
 $n=4, \frac{1}{2} - \frac{3}{4} = -\frac{1}{4}$
 $n=5, \frac{1}{2} - \frac{3}{5} = \frac{5-6}{10} = -\frac{1}{10}$
 $n=6, \frac{1}{2} - \frac{1}{2} = 0$
 $n=7, \frac{1}{2} - \frac{3}{7} = \frac{7-6}{14} = \frac{1}{14}$
 $n=8, \frac{1}{2} - \frac{3}{8} = \frac{8-6}{16} = \frac{1}{8}$

$c_1 = \frac{1}{14}$ (as c_1 is positive), $\therefore n_0 \approx 7$.

$c_2 = \frac{1}{2}$

$\therefore \frac{1}{2}n^2 - 3n \leq O(n^2)$.

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

$$g(n) = n^2$$

Prove $\exists c_1, c_2 \in \mathbb{R}$ $f(n) = O(g(n))$.

Proof:

$$0 \leq c_1 n^2 \leq 2n^2 + 3n \leq c_2 n^2 \quad \forall n \geq n_0$$

\Rightarrow

$$c_1 \leq 2 + \frac{3}{n}$$

$$\leq c_2$$

$$(c_1 \leq 2 + \frac{3}{n}, 2 + \frac{3}{n} \leq c_2)$$

$$n=1$$

$$2 + \frac{2}{1} = 4$$

$$n=2$$

$$2 + \frac{2}{2} = \frac{8+2}{4} = 10/4 = 5/2$$

$$n=3$$

$$2 + \frac{2}{3} = \frac{18+2}{9} = \frac{20}{9} = \frac{20}{9}$$

$$n=4$$

$$2 + \frac{2}{4} = \frac{32+2}{16} = \frac{34}{16} = \frac{17}{8}$$

$$n=1 = 2 + 2 = 4$$

$$3n \leq n^2 \quad \forall n \geq 3$$

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

$$\Rightarrow 3n + 2n^2 \leq 3n^2 \quad \forall n \geq 3$$

$$\Rightarrow 3n + 2n^2 \leq c_2 n^2 \quad \text{where } c_2 = 3$$

$$n^2 \leq 2n^2$$

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

$$\Rightarrow c_1 n^2 \leq 2n^2 + 3n \quad \text{where } c_1 = 1$$

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

$$c_1 = 1, c_2 = 3, n_0 = 3$$

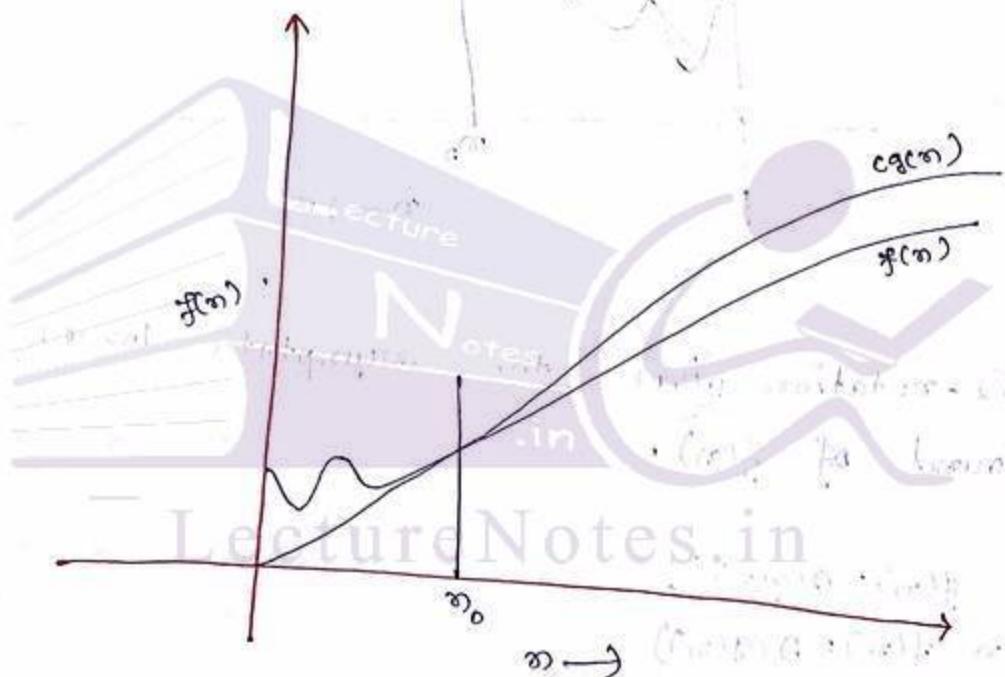
O(Big-O) Notation

For a given function $g(n)$, $O(g(n))$ gives a set of functions

$O(g(n)) = \{f(n) : \text{If there exist positive constants } C \text{ and } n_0 \text{ such that}$

$$O \leq f(n) \leq Cg(n).$$

for all $n > n_0\}$.



* O-notation gives the upper boundary of $f(n)$.

It gives the asymptotic upper boundary.

Ω (Omega) Notation

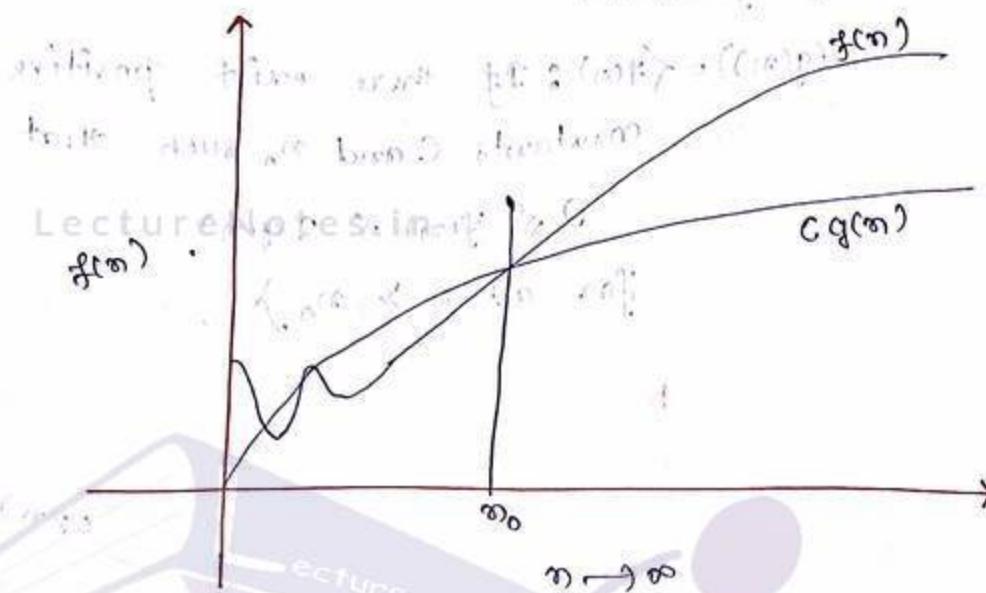
For a given $g(n)$, $\Omega(g(n))$ gives a set of functions

$\Omega(g(n)) = \{f(n) : \text{If there exist positive}$

constants c_1 and n_0 such that

$$0 \leq c g(n) \leq f(n)$$

for all $n > n_0$.



* Ω -notation gives bound of $f(n)$.

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

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

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

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

the $g(n)$ is asymptotic tight bound of $f(n)$.

$\Theta \rightarrow$ asymptotic upper bound.

$\Omega \rightarrow$ asymptotic lower bound. $n \% \in \Omega(n)$ not

(iv) O (little- O) Notation

* O (little- O) notation is used to denote an asymptotically tight upper bound that is not asymptotically tight.

$O(g(n))$ denotes a set of functions

$O(g(n)) = \{f(n) : \text{for any positive constant } c,$
 there exist a constant $n_0 > 0$
 such that $0 < f(n) < cg(n)$
 for all $n > n_0\}$.

Ex: $2n = O(n^2)$..

$$f(n) = 2n$$

$$g(n) = n^2$$

Ex: $2n^2 \neq O(n^2)$.

If $f(n) = O(g(n)) \Rightarrow f(n) = O(g(n))$,

but the reverse is not true.

To know $f(n) \in O(g(n))$,

$$\boxed{\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0}$$

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

$$\boxed{\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \text{constant}}$$

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

ω (little Ohmega) Notation

ω (little-ohmega) defines the lower bound which is not asymptotically tight.

$\omega(g(n))$ denotes a set of functions.

$\omega(g(n)) = \{f(n) : \text{for any tve. constant } c_g < 1, \text{ there exist a constant } n_0 > 0 \text{ such that}$

$$\exists c_g > 0 \quad \forall n > n_0 \quad c_g g(n) < f(n) \quad \text{for all } n > n_0 \}$$

$$\boxed{\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty}$$

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

* Note:

$\Leftrightarrow f(n) \in \omega(g(n))$

$\Leftrightarrow f(n) \in \omega(g(n))$

$\Rightarrow g(n) = O(f(n))$ (Little-oh)

$\Leftrightarrow f(n) \in \omega(g(n)) \text{ if and only if } g(n) = O(f(n))$

Transitivity:

- $\exists f(n) = O(g(n))$ and $g(n) = O(h(n))$ imply $f(n) = O(h(n))$.
- $\exists f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n))$ imply $f(n) = \Omega(h(n))$.
- $\exists f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$ imply $f(n) = \Theta(h(n))$.
- $\exists f(n) = o(g(n))$ and $g(n) = o(h(n))$ imply $f(n) = o(h(n))$.
- $\exists f(n) = \omega(g(n))$ and $g(n) = \omega(h(n))$ imply $f(n) = \omega(h(n))$.

Reflexivity:

- $\exists f(n) = O(f(n))$.
- $\exists f(n) = \Omega(f(n))$.
- $\exists f(n) = \Theta(f(n))$.

Symmetric:

- $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$.

Transpose Symmetry:

- $f(n) = O(g(n))$ if and only if $g(n) = \omega(f(n))$.
- $f(n) = o(g(n))$ if and only if $g(n) = \omega(f(n))$.

Asymptotic Function comparison

Real Numbers (a, b) (Interpretation of the notation)

$\Leftarrow f(n) = O(g(n)) \approx a < b$.

$\Leftarrow f(n) = O(g(n)) \approx a = b$.

$\Leftarrow f(n) = \Omega(g(n)) \approx a > b$.

$\Leftarrow (f(n) = \Theta(g(n)), \text{ it } \approx a < b, \text{ or } a > b)$.

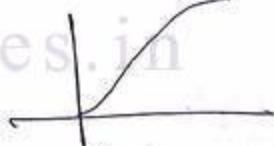
$\Leftarrow f(n) = \omega(g(n)) \approx a > b$.

Monotonically Increasing Function

Given m and n if $m > n$, it implies

$$f(m) > f(n),$$

then $f(\cdot)$ is called a monotonically increasing function.



Strictly monotonically increasing function

Given m and n if $m > n$, it implies

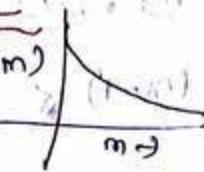
$$f(m) > f(n),$$

then $f(\cdot)$ is called a strictly monotonically increasing function.

Monotonically Decreasing Function

Given m and n , if $m > n$ implies $f(m) \leq f(n)$,

then $f(x)$ is called a monotonically decreasing function.



Strictly Monotonically Decreasing Function

Given m and n if $m > n$ implies $f(m) < f(n)$,

then $f(x)$ is called a strictly & monotonically decreasing function.

$\lceil x \rceil \rightarrow \text{ceiling}$.

The least integer that is greater than or equal to x .

$\lfloor x \rfloor \rightarrow \text{floor}$.

The greatest integer that is less than or equal to x .

$$\lceil 5 \rceil = 5$$

$$\lfloor 5 \rfloor = 5$$

$$\lceil 5.97 \rceil = 6$$

$$\lfloor 5.9 \rfloor = 5$$

~~$x-1, \lfloor x \rfloor, x, \lceil x \rceil, x+1$~~

~~$(x-1) \leq \lfloor x \rfloor \leq x < \lceil x \rceil \leq x+1$~~

~~greatest integer function is called floor function.~~

Polynomial

Constant $\neq x^d$, $d \geq 0$,

1 polynomial of degree (d)

$$P(x) = \sum_{i=0}^d a_i x^i \quad \text{where } a_d \neq 0$$

27/7/2015

Standard Notation and Common Functions

Floor and Ceiling

$$(x-1) < \lfloor x \rfloor \leq x < \lceil x \rceil < x+1$$

$$f(x) = \lceil x \rceil$$

$$f(x) = \lfloor x \rfloor$$

monotonically increasing functions.

Modular Arithmetic

Gives remainder of the division.

$$50/2 \sim 1 \quad 5 \cdot 1 \cdot 2 \equiv 1$$

Polynomial

$P(n) = \sum_{i=0}^{\infty} a_i n^i$, where $a_0, a_1, \dots, a_i \rightarrow \text{co-efficient}$
 $n^i \rightarrow \text{exponent}$
 $i \geq 0$.

Polynomial $P(n)$ is of degree (d) if n^d , $d \geq 0$
 and $a_d \neq 0$.

$P(n)$ is called asymptotically positive polynomial
 if and only if a_d is positive. ($a_d > 0$) .

$P(n)$ is called polynomially bounded, if it is
 asymptotically ^{positive and} bounded.

$$P(n) = O(n^k)$$

for some constant, k .

Ex: $P(x) = 1 + x + x^2$.

$$\Rightarrow P(x) = O(x^2)$$

It is a polynomially bounded function.

$$P(x) = 2x + 3x^3 + 4x^9$$

$$\Rightarrow P(x) = O(x^9)$$

* - All asymptotically positive polynomial are polynomially bounded. true for all unbounded

$n^\alpha \rightarrow$ monotonically increasing if $\alpha > 0$. $n \rightarrow$ variable as constant

$n^\alpha \rightarrow$ monotonically decreasing if $\alpha < 0$.

Exponentials

For all real

$$a > 0$$

and m, n

$$a^0 = 1$$

$$a^m \neq 0$$

LectureNotes.in

$$(a^n)^m = a^{nm}$$

$$a^n \cdot a^m = a^{n+m}$$

$$(a^m)^n = a^{mn}$$

LectureNotes.in

all these terms are exponentials

- * The rates of growth of polynomials and exponentials can be related by, for all real constants a and b , $a \geq 1$

$$\lim_{n \rightarrow \infty} \frac{n^b}{a^n} = 0$$

$n^b \rightarrow$ polynomial

$a^n \rightarrow$ exponential

$$\Rightarrow n^b = o(a^n)$$

need to

- * We design algorithms which are polynomially bounded not exponential

$e^n \rightarrow$ exponential form

Logarithmic

$$\lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n = e^x,$$

for all x .

Logarithmic

$$\log_b n = \log_2 n / \log_2 b$$

$$\ln n = \log_e n$$

$$\lg n = (\log_2 n)^k$$

$$\log_b \log_2 n = \log_2 (\log_2 n)$$

logarithm is strictly increasing function.

For $b > 1$, for $n > 0$, $\log_b n$ is strictly increasing function.

$$b^{\log_b a} = a$$

$$\lim_{n \rightarrow \infty} \log_2 n = \infty$$

A function $f(n)$ is called polylogarithmically bounded

$$f(n) = O(\lg^k n)$$

$$\lim_{n \rightarrow \infty} \frac{\lg^k n}{n^a} = 0$$

$$\Rightarrow \lg^k n \in o(n^a).$$

$$\lim_{n \rightarrow \infty} \frac{\lg^b n}{n^a} = 0$$

for any constant $a > 0$,

$$\forall a, \lg^b n = o(n^a)$$

A polynomial function grows faster than polylogarithmic function.

Factorials

$$n! = 1, \quad n=0$$

$$n(n-1)! \quad , \text{ if } n > 0$$

Finding's Approximation

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + o\left(\frac{1}{n}\right)\right),$$

n^n is a weak upperbound of $n!$.

$$n! = O(n^n)$$

$$n! \approx \sqrt{n} (2^n)$$

$$\lg(n!) \approx O(n \log n)$$

For $n > 1$,

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

where $\frac{1}{12n+1} < x_n < \frac{1}{12n}$

Functional Iteration

$f^{(i)}(n)$ denote the function $f(n)$ iteratively, applied i times to an initial value of n .

Let $f(n)$ be a function on real numbers. For non-negative value of i

$$\begin{cases} f^{(i)}(n) = n & \text{if } i=0 \\ f(f^{(i-1)}(n)) & \text{if } i>0 \end{cases}$$

Base condition $f^{(0)}(n)=n$

$$f(n) = 2n = 2f^0(n) \quad f(n) \rightarrow \text{applied } i \text{ times on } n$$

$$f^2(n) = 2^2 n \quad f^2(n) = f(f^0(n)) = f(n)$$

$$f^3(n) = 2^3 n \quad f^3(n) = f(f^2(n)) = f(f^0(n)+1) = f(n+1)$$

$$f^4(n) = 2^4 n \quad f^4(n) = f(f^3(n)) = f(f^0(n)+2) = f(n+2)$$

Iterated Logarithm Function. $f^3(n) = f(f^2(n)) = f(n+2) = n+8$

$$\log_2 n \quad f^i(n) = f(f^{i-1}(n))$$

$$\log_2 2 = 1 \quad = f(n+(i-1)) = (n+i-1)+1$$

$$\log_2 4 = 2 \quad = n+i^0 + 1$$

$$\log_2 16 = 4 \quad = n+i^1 + 1$$

$$\log_2 65536 = 16 \quad = n+i^2 + 1$$

*- Growth rate of iterative logarithmic function is very slow.

Maximum is 5.

$$A = \log_2 n, B = n^6$$

$$A = O(B) \text{ and } A = o(B)$$

$$\text{as } \lim_{n \rightarrow \infty} \frac{\log^6 n}{n^6} = 0$$

$A = 2^n$, $B = C^n$

$$\Rightarrow A = O(B)$$

$$A = O(B)$$

$$\frac{2^n}{C^n}$$

$$\lim_{n \rightarrow \infty} \frac{2^n}{C^n}$$

* - $A = 2^n$, $B = n$

$$\lim_{n \rightarrow \infty} \frac{2^n}{n} = \infty$$

$$\Rightarrow A = \Theta(B)$$

$\Rightarrow A = \Theta(B)$, $A = O(B)$, $A = \Omega(B)$.

$$\lim_{n \rightarrow \infty} \frac{2^n}{n} = \infty$$

$$\Rightarrow A = \Omega(B)$$

* - $A = 2^n$, $B = 2^{n/2}$

$$\lim_{n \rightarrow \infty} \frac{2^n}{2^{n/2}} = \infty$$

$$\Rightarrow A = \omega(B)$$

$$\lim_{n \rightarrow \infty} \frac{2^n}{2^{n/2}}$$

$$= \lim_{n \rightarrow \infty} \frac{2^n}{\frac{2^n}{2}} = \lim_{n \rightarrow \infty} 2 = (\text{Divide } 2^n)$$

$$= \lim_{n \rightarrow \infty} \frac{1}{\frac{1}{2}} = \infty$$

$$\Rightarrow A = \omega(B)$$

$$\Rightarrow A = \Omega(B)$$

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

* - $A = n^{\log c}$, $B = C^{\log n}$

* - $A = \log(n!)$, $B = \log n^n$

$$\lim_{n \rightarrow \infty} \frac{\log(n!)}{\log n^n} = 1$$

$$\Rightarrow A = O(B)$$

$$\Rightarrow A = \Theta(B)$$

$$\checkmark \lim_{n \rightarrow \infty} \frac{\log C}{C^{\log n}}, B = C^{\log n}$$

$$C > 0$$

(n can never be
0 or 1)

$n^{\log C} \rightarrow$ polylogarithmic
 $C^{\log n} \rightarrow$ exponential

$$\Rightarrow \lim_{n \rightarrow \infty} \frac{n^{\log C}}{C^{\log n}} \approx 0$$

$$\therefore f = O(C^{\log n}) = O(B) \\ f = O(B)$$

$$\boxed{\log^{*} n < \log n < n^2 < 2^n < n! < n^n}$$

$$\text{Ex: } \log(\log^{*} n), 2^{\log^{*} n}, (\sqrt{2})^{\log n}, n^2, n!, (\log n)!$$

$$\log(\log^{*} n) < 2^{\log^{*} n} < (\sqrt{2})^{\log n} < n^2 < (\log n)! < n!$$

58.

$$f(n) = \frac{1}{2}n^2 - 3n \quad \frac{f(n)}{g(n)} = \frac{\frac{1}{2}n^2 - 3n}{n^2} = \frac{\frac{1}{2}n^2 - 3n}{n^2} = \frac{\lim_{n \rightarrow \infty} (\frac{1}{2} - \frac{3}{n})}{1} = \frac{1}{2}$$

$$\therefore f(n) \sim O(g(n))$$

Recurrences

When an algorithm makes a recursive call to itself, its running time can be expressed by recurrences.

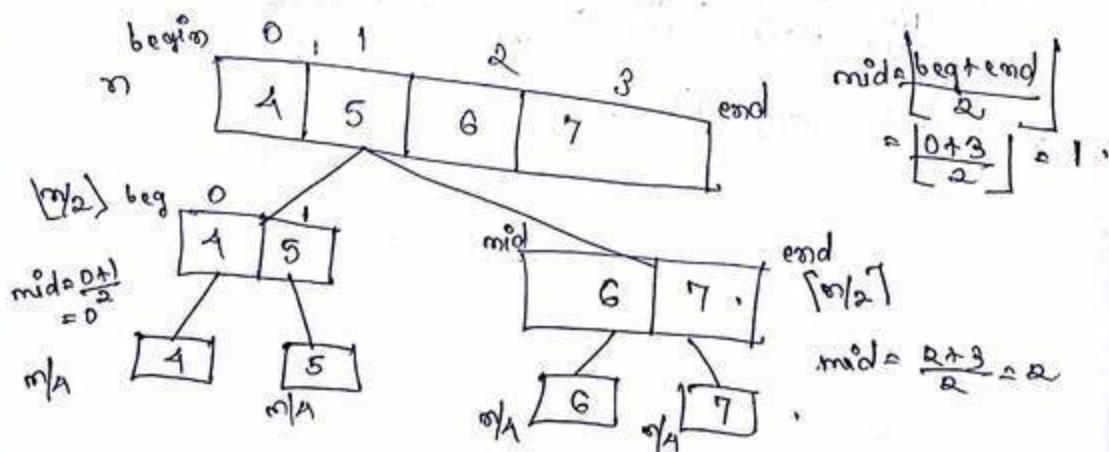
1) Recurrence is an equation or inequality that describes a function in terms of its value for smaller similar inputs.

Ex: The recurrence for merge sort can be given as

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

Divide and Conquer

1. Divide the problem into sub-problems
2. Conquer: solve the sub-problems
3. Combine: combine the result of subproblems to whole solution



- Algorithm of Merge-sort
- Mergesort (A , $\underline{\text{beg}}$, end)
- 1 if ($\underline{\text{beg}} < \text{end}$)
 - 2 $\text{mid} = \left\lfloor \frac{\underline{\text{beg}} + \text{end}}{2} \right\rfloor$
 - 3 mergesort (A , $\underline{\text{beg}}$, mid)
 - 4 mergesort (A , $(\text{mid} + 1)$, end)
 - 5 merge (A , $\underline{\text{beg}}$, mid , end)

$$\begin{aligned} T(n) &= T(\frac{n}{2}) + T(\frac{n}{2}) + f(n) \\ T(n) &= 2T(\frac{n}{2}) + f(n) \end{aligned}$$

Recurrence

Worst case :

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

↓
solⁿ of the recurrence

- * Three methods used to solve a recurrence.
- i) Substitution method
 - ii) Recursion tree method
 - iii) Master method

3/8/2015.

Master Method

The master method is used to solve the recurrence when the recurrence is in the form

$$T(n) = a \cdot T(\frac{n}{b}) + f(n),$$

Where $a \geq 1$, $b \geq 1$

$$T(\frac{n}{b}) = T(\lfloor \frac{n}{b} \rfloor) \text{ or } T(\lceil \frac{n}{b} \rceil)$$

* The problem of size n is divided into "a" parts each part is of size $\lceil n/b \rceil$ or $\lceil n/b \rceil$ and $f(n)$ is time.

↓ asymptotically

dividing and combining

Theorem 4.1

Let $a > 1, b > 1$ be constants. Let $f(n)$ be a function and let $T(n)$ be defined on the non-negative integers by the recurrence relation

$$T(n) = a T\left(\frac{n}{b}\right) + f(n),$$

then $T(n)$ can be bounded asymptotically as follows:

i) if $f(n) = O(n^{\log_b a - \epsilon})$

for some $\epsilon > 0$,

then $T(n) = O(n^{\log_b a})$.

ii) if $f(n) = O(n^{\log_b a})$,

then $T(n) = O(n^{\log_b a} \log n)$.

iii) if $f(n) = \Omega(n^{\log_b a + \epsilon})$

for some constant $\epsilon > 0$ and

if $a f\left(\frac{n}{b}\right) \leq c f(n)$, → Regularity condition

for some constant $c < 1$ and all sufficiently large n , then

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

Ex: Solve the given recurrence $T(n) = 9 T\left(\frac{n}{3}\right) + n$

Solution:

$$T(n) = 9 T\left(\frac{n}{3}\right) + n$$

Now, $a=9$, $b=3$, $f(n)=n$

Step-1: Now, $n^{\log_b a} = n^{\log_3 9} = n^{\log_3 3^2} = n^2$

Step-2:

$$\begin{aligned} f(n) &= n \\ &= O(n) \\ &= O(n^{2-1}) \\ &= O(n^{\log_b a - 1}) \\ &= O(n^{\log_b a - \epsilon}) \end{aligned}$$

where $\epsilon > 0$

$$\therefore f(n) = O(n^{\log_b a - \epsilon})$$

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

$$= O(n^2)$$

Ex: $T(2n/3) + 1$

$$a=1, b=\frac{3}{2}, f(n)=1$$

Step-1: $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

Step-2:

$$\begin{aligned} f(n) &= 1 \\ &\approx O(1) \\ &= O(n^{\log_b a}) \\ &= O(n^{\log_{3/2} 1}) \end{aligned}$$

$$\text{as } 1 = O(n)$$

$$= O(n)$$

$$= \Omega(n)$$

$\therefore \exists \delta, T(n) \geq \delta n^{\log_{3/2} 1} = \delta \Theta(n) = \Theta(n)$

$$T(n) = 3T\left(\frac{n}{4}\right) + n \log_6 n$$

$$a=3, b=4, f(n) = n \log_6 n$$

$$n \log_b a = n \log_4 3 = n^{0.793}$$

$$\begin{aligned} f(n) &= n \\ &\approx n^{0.793 + 0.207} \\ &= n^{0.793} \cdot n^{0.207} \\ &= n^{\log_b a + \epsilon} \quad \text{if } \epsilon = 0.207 \\ &= n^{\log_6 3 + \epsilon} \end{aligned}$$

The numbers will fit for the solution.

Check for regularity condition.

$$af\left(\frac{n}{b}\right) = 3f\left(\frac{n}{4}\right) = 3\left(\frac{n}{4}\right) \quad (\because f(n) = n)$$

$$\Rightarrow 3\left(\frac{n}{4}\right) \leq 0.9(n), \quad c = 0.9$$

$\therefore f(n)$ satisfies regularity condition.

$$\therefore T(n) = O(f(n))$$

$$= O(n)$$

$$Q: T(n) = 4T\left(\frac{n}{2}\right) + n$$

$$a=4, b=2, f(n)=n$$

$$n \log_b a = n \log_2 4 = n^{0.793} \quad n \log_2 2^2 = n^2$$

$$\begin{aligned} f(n) &= n^{2-1} \\ &= n^1 \end{aligned}$$

$$\therefore \left(n^{\log_b a - \epsilon} \right) = O\left(n^{\log_2 4 - \epsilon}\right)$$

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

$$= \Theta(n^2) \quad (\text{By. master-1})$$

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

$a=4, b=2, f(n)=n^2$

 $n^{\log_b a} = n^{\log_2 4} = n^2 \in \Theta(n^2) = \Theta(n^{\log_b a})$

$$\begin{aligned} T(n) &\approx n^2 \\ &= \log_b a \end{aligned}$$

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

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

$a=4, b=2, f(n)=n^3$

 $n^{\log_b a} = n^{\log_2 4} = n^2$
 $f(n)=n^3 = n^{2+1} = n^{\log_b a + 1} = \Theta(n^{\log_b a + 1})$

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

$$a f(n/2) = 4 f(n/2) = 4 \cdot \frac{n^3}{8} = \frac{n^3}{2} \approx 0.2(n^3)$$

$$\Rightarrow T(n) \approx \Theta(n^3)$$

Observations of Master's Rule

In rule-1, $f(n)$ is asymptotically less than $n^{\log_b a}$ by the amount or value n^{ϵ} for $\epsilon > 0$.

i.e., $f(n)$ is polynomially smaller than $n^{\log_b a}$.

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

⇒ $f(n)$ is equal to $n^{\log_b a}$. (Rule-2)

$$f(n) = \Omega(n^{\log_b a + \epsilon})$$

$f(n)$ is greater than

$n^{\log_b a}$ by amount n^{ϵ} .

i.e., $f(n)$ is polynomially greater than $n^{\log_b a}$.

Q.

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

$$n^{\log_b a} = n^{\log_2 2} = n^1 = n$$

$$f(n) = n \log n$$

$$= n^{\log_b a} \log n$$

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

(logarithmically greater)

5/8/2015. 9

$$T(n) = 2T(n/2) + n \log_2 n$$

$$a=2, b=2, f(n)=n \log_2 n$$

$$n \log_2 n = n^1 \log n$$

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

Recurrence
cond. $f(n) \leq c f(n)$

$$2f(n/2) \leq c f(n)$$

$$2f(n/2) \leq 0.2 n$$

$$2f(n/2) \leq 0.2 n \log_2 3$$

$$2f(n/2) \leq 0.2 n \log_2 3$$

$$2f(n/2) \leq 0.2 n \log_2 3$$

$f(n)$ is asymptotically larger than $n^{\log_2 2}$. But, not polynomially larger.

Box 4.4.1

$$\text{If } f(n) = \Theta(n^{\log_2 a} \log^k n),$$

then the master recurrence is $T(n) = \Theta(n^{\log_2 a})$.

$$T(n) = \Theta(n^{\log_2 a} \log^{k+1} n)$$

$$f(n) = n \log n$$

$$= n^{\log_2 2} \log n$$

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

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

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

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

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



The Recursion Tree Method

Recurrence Tree

Given recurrence for merge sort

$$T(n) = O(1), \text{ for } n=1$$

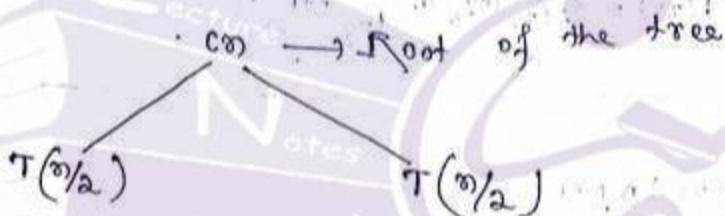
$$\text{base case: } T(n) = 2T\left(\frac{n}{2}\right) + O(n) \text{ if } n > 1$$

Construction of recursion tree

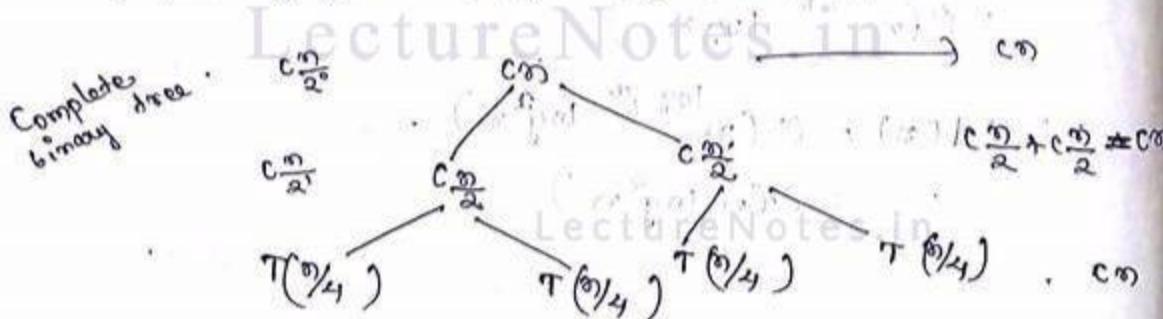
(a) $T(n)$

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

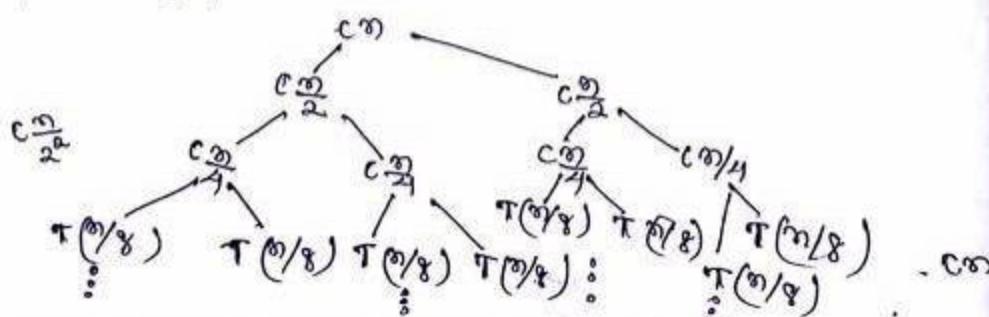
$$= T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + O(n)$$



$$T\left(\frac{n}{2}\right) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{4}\right) + O\left(\frac{n}{2}\right)$$



$$T\left(\frac{n}{4}\right) = T\left(\frac{n}{8}\right) + T\left(\frac{n}{8}\right) + O\left(\frac{n}{4}\right)$$



$\frac{c \cdot n}{2^k} T(1) \approx c$ c c c c c , \rightarrow on
 No. of c is equal to no. of n.

$$T(n) = \frac{c \cdot n}{2^k} + c$$

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

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

$$\Rightarrow k = \log n$$

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

$$= cn \log n + cn$$

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

(for height $(k+1)$).

$$T(n) = \begin{cases} T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

6/8/2015.

Arranging the following functions in ascending order:

$n \log n, (\log n)^3, 2^n, 4^{\log n}, 2^{n^2}, n^3, n^{100}$.

$\log^3 n < n \log n < 4^{\log n} < n^3 < n^{100} < 2^{n^2}$.

$$a^{\log_b c} = c^{\log_b a}$$

$$4^{\log n} = n^{\log 4} = n^{\log 2^2} = n^{2 \log 2} = n^{2 \cdot 1} = n^2,$$

$$\underline{\text{Q.E.D.}} \quad n^3 + 2n^2 + 2n = \Theta(n^3)$$

\therefore

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

$$g(n) = n^3$$

$$\begin{aligned} &\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c \\ \Rightarrow f(n) &\sim \Theta(g(n)) \end{aligned}$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{n^3 + 2n^2 + 2n}{n^3}$$

$$= \lim_{n \rightarrow \infty} \left(1 + \frac{2}{n} + \frac{2}{n^2} \right)$$

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

$$\approx 1$$

LectureNotes.in

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

$\Rightarrow n^3 + 2n^2 + 2n \sim \Theta(n^3)$.

$$\text{Q.E.D. } P.T \log(n!) = O(n \log n) \quad 2$$

$$f(t) = \log(n!)$$

$$g(t) = t \log t$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = C$$

$$\Rightarrow \lim_{n \rightarrow \infty} \frac{\log(n!)}{t \log t}$$

LectureNotes.in

By Stirling's approximation

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n (1+O(Y_n))$$

$$\begin{aligned} \Rightarrow \log(n!) &\approx \log(\sqrt{2\pi n}) + \log\left(\frac{n}{e}\right)^n (1+O(Y_n)) \\ &= \log(\sqrt{2\pi n}) + \log(n/e)^n + \log(O(Y_n)) \\ &= \sqrt{2\pi} \log n + n \log(n/e) + \log(O(Y_n)) \\ &\quad (\because \text{neglecting one}) \\ &= \sqrt{2\pi} \log n + n \log n - n \log e + \log(O(Y_n)) \\ &= \sqrt{\pi/2} \log n + n \log n - n \log e + \log C \sim \log n \end{aligned}$$

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} &= \sqrt{\pi/2} \frac{\log n}{n \log n} + \frac{n \log n}{n \log n} - \frac{n \log e}{n \log n} + \frac{\log C}{n \log n} \\ &\sim \frac{\log n}{n \log n} + 1 - \frac{\log e}{\log n} + \frac{\log C}{n \log n} - \frac{1}{n} \end{aligned}$$

$$\approx 0 + 1 - 0 + 0 - 0$$

$$\approx 1$$

$$\Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1 \Rightarrow f(n) = O(g(n)) \Rightarrow \boxed{\log(n!) = O(n \log n)}$$

Q. Solve the recurrence

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

$$\approx T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn \quad \text{if } n > 1$$

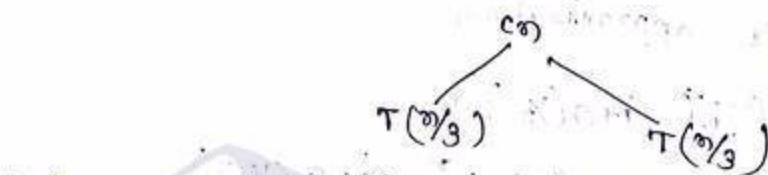
$$T(n) = 1 \quad \text{if } n = 1$$

(a)

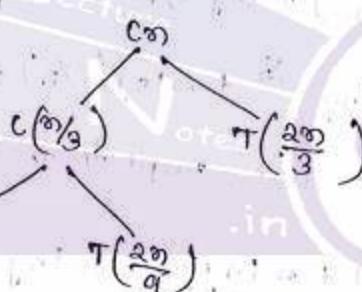
$$T(n)$$

LectureNotes.in

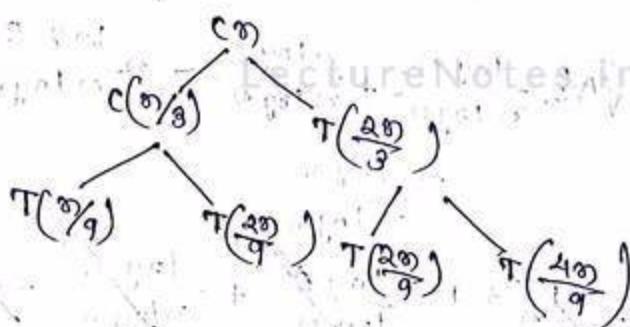
(b)



$$T\left(\frac{n}{3}\right) = T\left(\frac{n}{9}\right) + T\left(\frac{2n}{9}\right) + c\left(\frac{n}{3}\right)$$



$$T\left(\frac{2n}{3}\right) = T\left(\frac{2n}{9}\right) + T\left(\frac{4n}{9}\right) + c\left(\frac{2n}{3}\right)$$

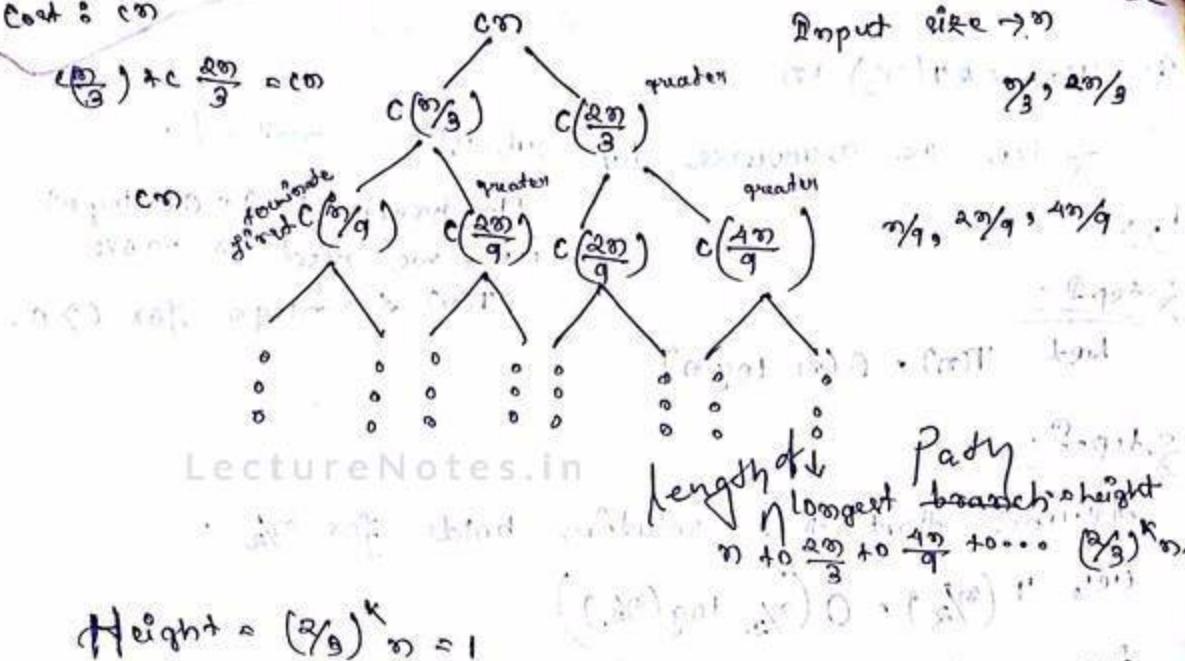


Computing T(n) is equivalent to the way we compute $\frac{1}{3}^n$

Cost : $C(n)$

Input size $\rightarrow n$

$$\left(\frac{C(n)}{3}\right) + C\left(\frac{2n}{3}\right) = C(n)$$



$$\text{Height} = \left(\frac{2}{3}\right)^k n = 1$$

$$\Rightarrow n = \left(\frac{2}{3}\right)^k = \left(\frac{2}{3}\right)^k$$

$$\log_{\frac{2}{3}} n = k$$

$$\Rightarrow k = \log_{\frac{3}{2}} n \approx \log_2 n$$

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

$$= cn \cdot \log n$$

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

Substitution Method

Step-1:

Make a guess:

Step-2:

Prove the guess by induction method.

$$(a \log a) \leq (a+1)$$

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

Solve the recurrence by substitution method.

Step 1:

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

Step 2:

Assume that the solution holds for $\frac{n}{2}$.

i.e., $T\left(\frac{n}{2}\right) = O\left(\frac{n}{2} \log\left(\frac{n}{2}\right)\right)$

i.e., $T\left(\frac{n}{2}\right) \leq c \frac{n}{2} \log\left(\frac{n}{2}\right)$ for $c > 0$.

Withth assumption, prove that,

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

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

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

$$\leq c(n \log\left(\frac{n}{2}\right)) + n$$

$$\leq c n \log\left(\frac{n}{2}\right) + n$$

$$\leq c n (\log n - \log 2) + n$$

$$\leq c n \log n - cn + n$$

$$\leq cn \log n$$

for $c > 1$

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

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

Prove the base condition

$$T(1) \approx 1$$

$$T(n) \leq cn \log n$$

$$\text{For } n=1,$$

$$T(1) \leq c1 \log 1 \quad (c=1)$$

$$\Rightarrow 1 \leq 0 \quad (\text{not satisfied})$$

$$\begin{aligned} T(2) &= 2 T\left(\frac{2}{2}\right) + 2 \\ &\approx 2 \cdot 1 + 2 = 4 \end{aligned}$$

$$T(2) \leq cn \log n$$

$$\leq 2 \cdot 2 \log 2 \quad (\therefore c=2)$$

$$\leq 4 \log 2$$

$$\leq 4 \cdot 1 \quad \text{for } n=2, c \geq 2$$

$$T(2) = O(n \log n) \quad \text{for } n=2, c \geq 2.$$

Hence, it is proved.

Matrix-Matrix Multiplication.

Divide and Conquer Method.

Matrix-Matrix Multiplication.

Divide and Conquer Method.

18/2015

Merge Sort.

Input: A sequence of numbers

$$\langle a_1, a_2, \dots, a_n \rangle$$

Output: A permutation of the input sequence

$$\langle a'_1, a'_2, \dots, a'_n \rangle \text{ such that } a'_1 < a'_2 < \dots < a'_n$$

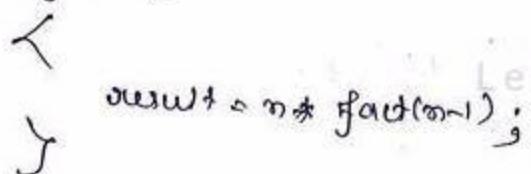
$$a'_1 \leq a'_2 \leq \dots \leq a'_n$$

Eg: Input sequence: $\{5, 4, 3, 2, 1\}$

Output: $\{1, 2, 3, 4, 5\}$.

- *- It follows divide and conquer approach.
- *- Any algorithm which is recursive in nature follows divide and conquer approach.

Eg: fact(n)



Divide and Conquer.

3 steps:

(i) Divide: the problem into sub-problems of same kind.

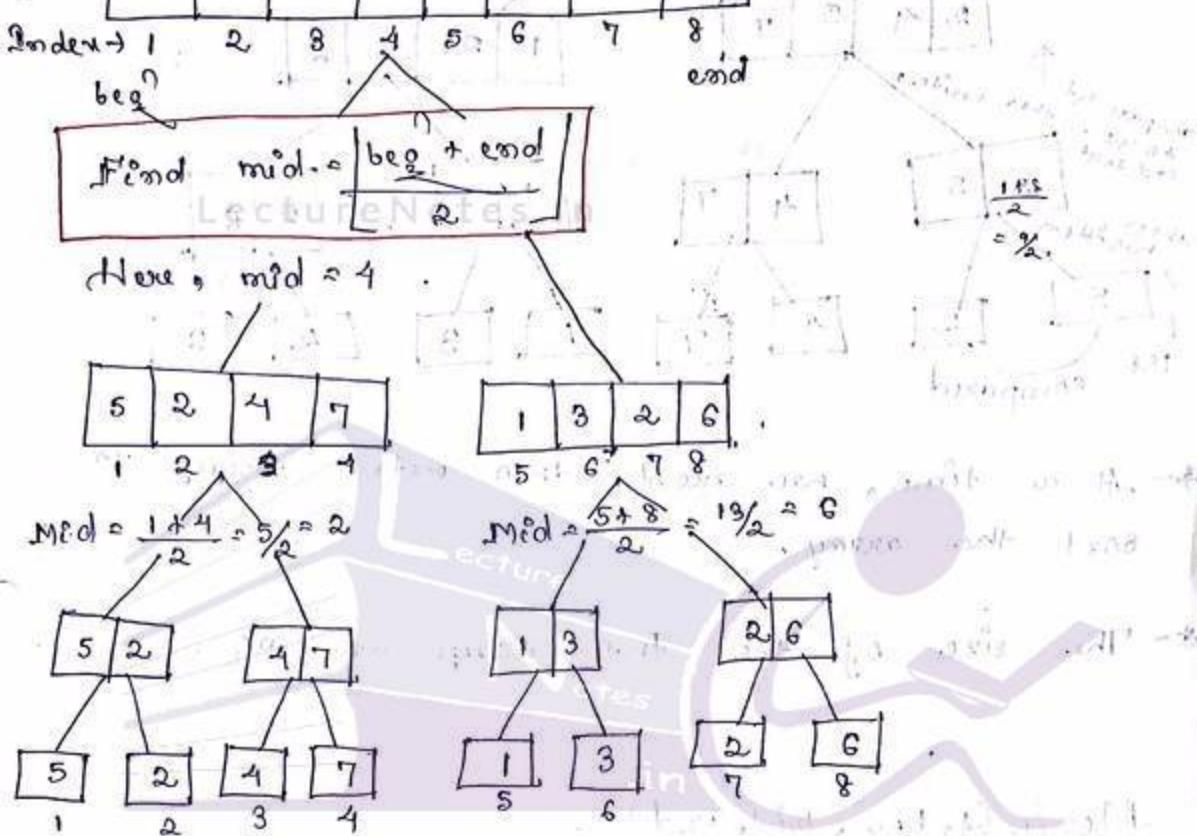
(ii) Conquer: solve the sub-problems/solve the sub-problems.

iii) Combine: the solution of the sub-problem to find the whole solution.

Ex:

5	2	4	7	1	3	2	6
Index → 1	2	3	4	5	6	7	8

 → Input array.



Algorithm of Merge Sort MergeSort($A, \underline{\text{beg}}, \text{end}$)

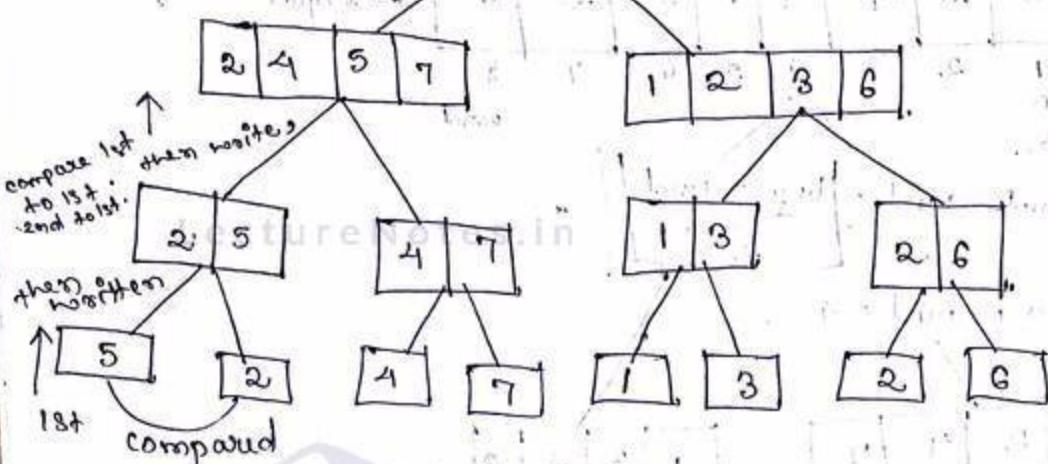
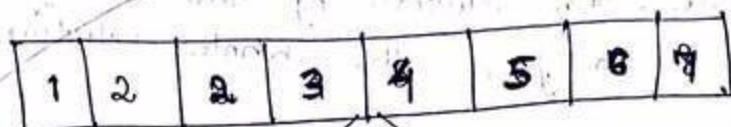
1. if $\underline{\text{beg}} < \text{end}$

$$\text{mid} = \frac{\underline{\text{beg}} + \text{end}}{2}$$

mergeSort($A, \underline{\text{beg}}, \text{mid}$)

mergeSort($A, \text{mid} + 1, \text{end}$)

merge($A, \underline{\text{beg}}, \text{mid}, \text{end}$)



* At a time, we need two extra arrays to sort the array.

* The size of the two arrays are $n/2$.

Merge (A , beg_1 , mid , end)

$\Leftarrow n_1 \leftarrow \text{mid} - \text{beg}_1 + 1$

$\Leftarrow n_2 \leftarrow \text{end} - (\text{mid} + 1) + 1 = \text{end} - \text{mid} - n_1 = \text{end} - \text{mid}$

\Leftarrow Create two arrays $L_1 = \{n_1+1\}$
and $L_2 = \{n_2+1\}$.

\Leftarrow for $i \leftarrow 1 + 0$ to n ,

\Leftarrow do $L_1[i] \leftarrow A[\text{beg}_1+i-1]$.

(Line 4 and 5 of the algorithm copied the $1 + 0$ elements of A to L_1 , and it assumed

that those elements are already sorted). 28

12/8/2015.

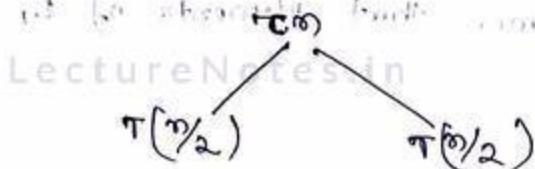
- Q: for $i=1$ to n_2
- do $L_2[i] \leftarrow 1 \text{ mod } L_1[i]$
- // The algorithm assumes that elements of L_1 and L_2 are sorted.
- Q: $L_1[1, n_1] \leftarrow \infty$
- Q: $L_2[n_2+1] \leftarrow \infty$
- 10: $i \leftarrow 1$
- 11: for $x = \text{beg}$ to end
- 12: $j \leftarrow 1$
- 13: for $x = \text{beg}$ to end
- 14: if $L_1[i] \leq L_2[j]$
- 15: $M[x] \leftarrow L_1[i]$
- 16: $i \leftarrow i+1$
- 17: else
- 18: $M[x] \leftarrow L_2[j]$
- 19: $j \leftarrow j+1$
- 20: end of for
- 21: end merge.

Analysis of Merge Sort

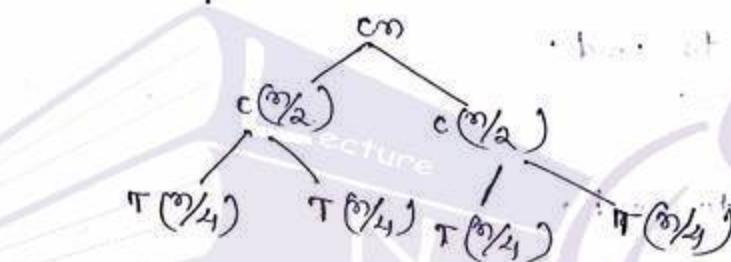
Recurrence

$$T(n) = \begin{cases} O(1), & n=1 \\ 2T(\frac{n}{2}) + O(n), & \text{otherwise} \end{cases}$$

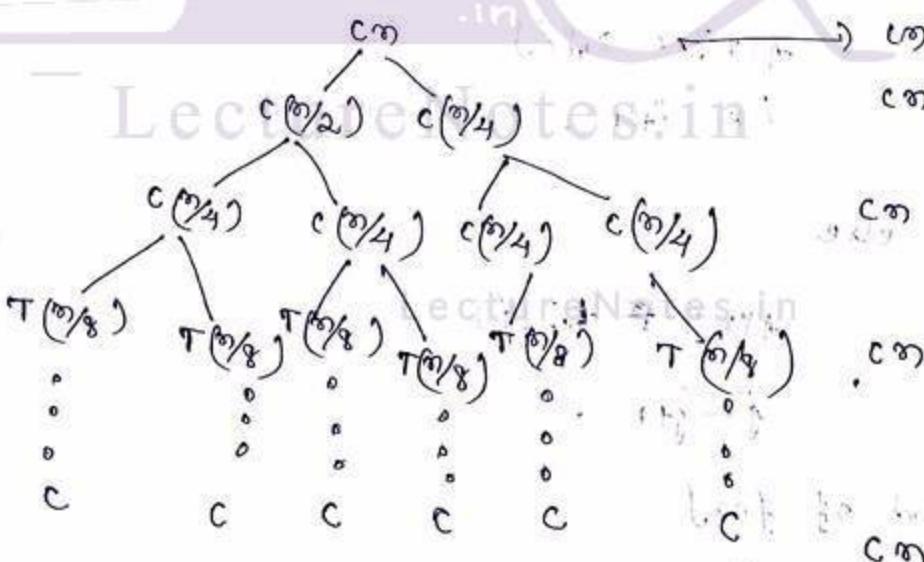
$$= 2T(\frac{n}{2}) + cn$$



$$T(\frac{n}{2}) = 2T(\frac{n}{4}) + c(\frac{n}{2})$$



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



Total cost.

$$\begin{aligned} T(n) &= cn + cn + \dots + cn \\ &= cn \log n \\ &= O(n \log n) \end{aligned}$$

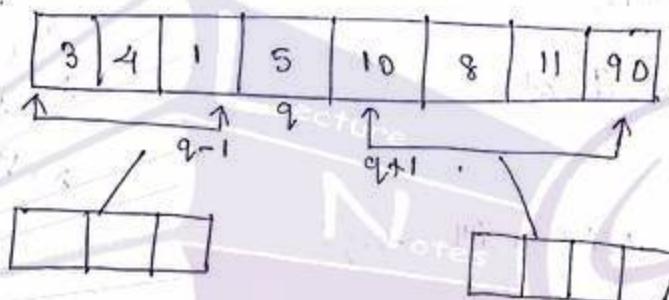
Height of complete binary tree is $\log n$

Quick Sort

- Follows divide and conquer approach.

Divide: Partitions the input array $A[P \dots R]$ into two sub-arrays $A[P \dots (q-1)]$ and $A[q+1 \dots R]$ such that all the elements of $A[P \dots (q-1)]$ are less than or equal to $A[q]$ and all the elements of $A[q+1 \dots R]$ are greater than or equal to $A[q]$.

A:



Conquer: Sort the sub-arrays $A[P \dots (q-1)]$ and $A[q+1 \dots R]$ by calling quick sort recursively.

Combine: Since the sub-arrays are sorted in place no need to combine and $A[P \dots R]$ is now sorted.

Algorithm

Quick sort (A, P, R)
if $P < R$

$P \rightarrow$ beginning of index
 $R \rightarrow$ end of index of array

then $q \leftarrow \text{Partition}(A, P, R)$

3: Quicksort ($A, P, (q-1)$)

4: Quicksort ($A, (q+1), \alpha$)

5: End :

Algorithm of Partition

Partition (A, P, α)

1: Pivot $\leftarrow A[P]$,

2: $i \leftarrow (P-1)$

3: for ($j = P$ to $(\alpha-1)$)

4: if $A[j] < \text{Pivot}$

$i \leftarrow i+1$

$i \leftarrow i+1$

swap $A[i]$ with $A[j]$

5: End of for.

6: Exchange $A[P]$ with $A[i]$

7: Return ($i+1$)

8: End .

10	8	90	11	5	7	9
$P=1$						$\alpha=7$

Find: q by calling partition()

Pivot element $\in A[0] \dots q$

$i = P-1 = 0$

17/8/2015

When $p = 0$, the array contains single elements.

Input:

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

Pivot

4

i 0

j 1

$A[i][j] = 2$ < pivot

i 1

swap

i 2

$A[i][j] = 8$ > pivot

i 3

$A[i][j] = 7$ > pivot

i 4

$A[i][j] = 1$ < pivot

i 5

swap $A[i][j], A[j][j]$

i 5

$A[i][j] = 3$ < pivot

i 3

swap

i 6

$A[i][j] = 6$ > pivot

i 7

$A[i][j] = 4$ > pivot

i 8

swap $i+1, j$, $A[i][j]$

i 8

$A[i][j] = 4$ > pivot

i 9

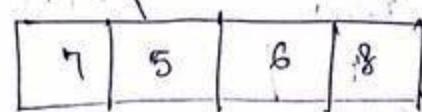
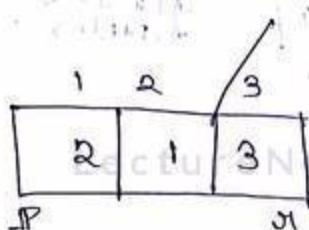
return $i+1, A[0][j] \rightarrow q$

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

return $i+1, A[0][j] \rightarrow q$.

After the 1st partition, q takes its perfect position.

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



$$\text{Pivot } A[3] = [3]$$

i 0

i 1 < pivot

i 1

swap

2	1	3
---	---	---

i 2 : < pivot

i 2

swap

2	1	3
---	---	---

$$\text{swap } A[1] = A[3], A[2] = A[3]$$

q ← A[3].

2	1	3
P	Pi	q

LectureNotes.in
(Imbalance partitioning)

1	2
P	qi

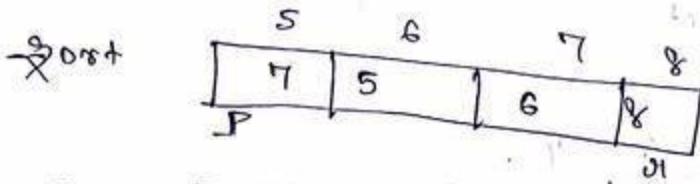
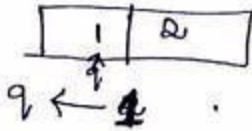
P < qi.

Pivot 1

i 0

i 1 & pivot

~~swap A[2:4] = A[1:3], A[0:1] = A[2:3]~~



Pivot

8

 P

5

i

4

j

5

 < pivot

i

5

 5
swap

7	5	6	8
---	---	---	---

j

6

 < pivot

i

6

 6
swap

7	5	6	8
---	---	---	---

j

7

 < pivot

i

7

 7
swap

7	5	6	8
---	---	---	---

j

8

 < pivot

swap A[2:4] = A[1:3], A[0:1] = A[2:3]

q ← i + 1 = 8

sort

5	6	7	8
7	5	6	8

P q

Pivot

5

 P

8

i

4

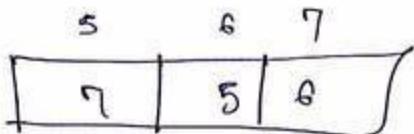
j

5

 < pivot

i 5

swap

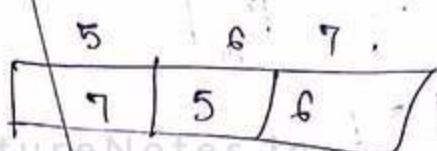


5 pivot

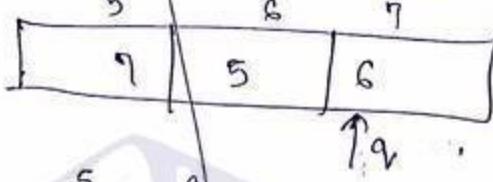
6

6

swap



swap $A[5+1] = A[6] = 6$, $A[5] = A[5] = 6$



↑
q

sort



Pivot

5

P = 5

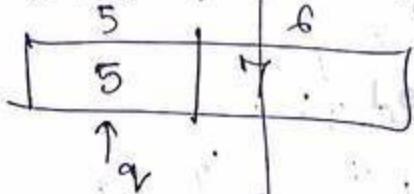
4

5

5 pivot

swap

$A[5+1] = A[6] = 7$, $A[5] = A[6] = 6$



5

7

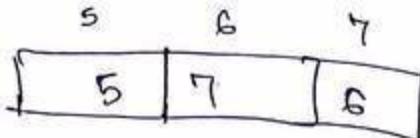
swap

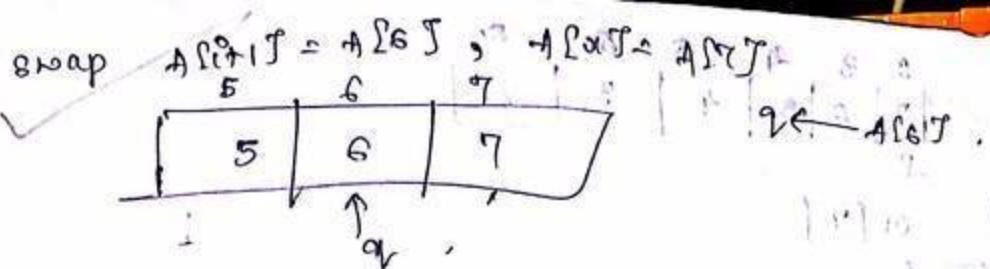
6

5

5 pivot

swap





$\boxed{5}$

$\boxed{7}$

LectureNotes.in

Q. 4 $\boxed{8 \ 7 \ 6 \ 5 \ 4 \ 3 \ 2}$

18/8/2015.

Given an input array A , S.T state of the array after each partitioning.

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

Pivot $\boxed{1}$

Pivot $\boxed{2}$

i $\boxed{0}$

i $\boxed{1}$ ~~pivot~~

i $\boxed{2}$ ~~pivot~~

i $\boxed{3}$ ~~pivot~~

i $\boxed{4}$ ~~pivot~~

i $\boxed{5}$ ~~pivot~~

i $\boxed{6}$ ~~pivot~~

swap $A[1]$ and $A[2]$

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

$\boxed{2}$

2	3	4	5	6	7
7	6	5	4	3	8

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

P 2 01 7
pivot 8

1 1

2 2 ~~pivot~~

3 2

2	3	4	5	6	7	
P	7	6	5	4	3	8

4 3 5 pivot

2	3	4	5	6	7	
P	7	6	5	4	3	8

6 4 5 ~~pivot~~

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

7 5 5 pivot

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

8 6 5 pivot

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

swap $\{1\}$ ~~$\{1\}$~~ $\{1\}$

2	3	4	5	6	7	
P	7	6	5	4	3	8

2	3	4	5	6	7	
P	7	6	5	4	3	8

P [a] a [6]

Pivot

[3]

[1]

[2] ⚡ pivot

[3] ⚡ pivot

[4] ⚡ pivot

[5] ⚡ pivot

swap 1[2] & 1[6].

2	3	1	5	6
3	6	.5	4	7

↑ q

3	4	5	6
6	5	4	7

P

Pivot

[6]

[7]

[2]

[3]

[4]

⚡ pivot

[5]

⚡ pivot

swap 1[2] with 1[6].

2	3	1	5	6
4	6	5	1	7

[3]

3	1	5	6
6	5	4	7

[4] ⚡ pivot

[4]

3	1	5	6
6	5	4	7

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

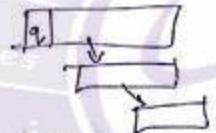
$$T(n) = T(n-1) + O(n)$$

Analysis of Quick Sort

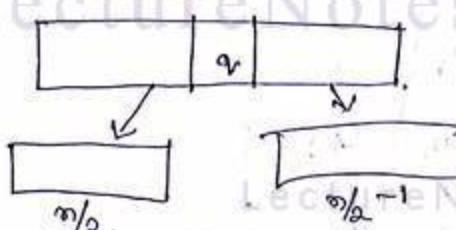
* The time taken by the quick sort depends on the partition point of the array.

* There are three cases of partition:

(i) Partition is not at all balanced i.e., most of the elements remain either after q or before q where q is the partitioning point. This case is called worst case partitioning.



(ii) The partition is completely balanced.



This is called best case partitioning.

(iii) The partitioning is not somehow balanced.

Let the partition is done in 9:1 ratio. This is called average case partitioning.

Time Analysis for worst case.

The recurrence for worst case partitioning is

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

Solving the recurrence by substitution method

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

Time Analysis for best case

The recurrence for best case partitioning is

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

$$\Rightarrow T(n) \leq 2T(n/2) + \Theta(n),$$

Solving the recurrence using master's method gives

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

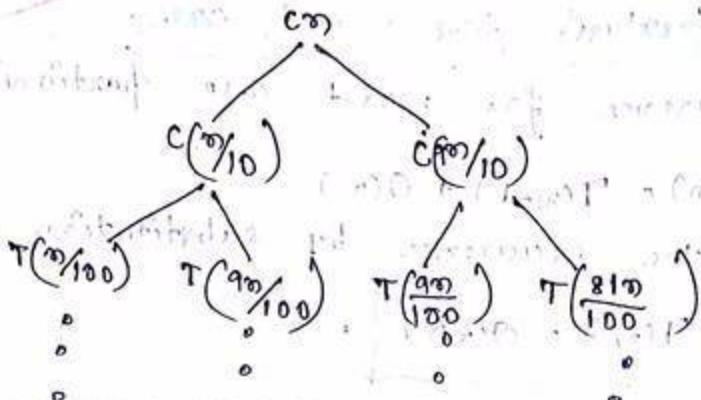
Time Analysis for average case

Let the array is partitioned by 9:1 ratio.

$$T(n)$$

$$T\left(\frac{n}{10}\right) \quad T\left(\frac{9n}{10}\right)$$





LectureNotes.in

$$C = O(1) \approx n$$

- Let after n number of steps the value of the leaf node in the longest path becomes 1.

$$\ell(n/10)^n = 1$$

$$\Rightarrow n(n/10)^n = 1$$

$$\Rightarrow n = (10/n)^n$$

$$\Rightarrow n \approx k \Rightarrow n = \log_{10} n \approx \log n$$

Total cost

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

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

Quicksort is the best sorting algorithm.

Randomized Quick Sort

* Worst case running time of quick sort = $\Theta(n^2)$ due to imbalanced partitioning.
 Best case and average case = $\Theta(n \log n)$.

* Imbalanced partitioning comes as the last element chosen as pivot element always.

Random, LectureNotes.in. It is a good idea to choose a random number between p and q.

P	1	2	3	4	5	6
a	6	5	4	3	2	1

$$P=1, a=5 \quad i=3$$

swap A[3] & A[1]

6	5	2	3	4
---	---	---	---	---

Algorithm

Randomized Quick Sort (A, P, R)

- 1. If $P < R$
- 2. $q \leftarrow \text{Randomized - partition } (A, P, R)$
- 3. Randomized - Quicksort ($A, P, (q-1)$)
- 4. Randomized - Quicksort ($A, (q+1), R$)
- 5. End

Algorithm of Randomized-Partition (A, P, π)

Randomized-partition (A, P, π)

$\Leftarrow i \leftarrow \text{Random}(P, n)$

$\Leftarrow \text{Swap}(A[\pi[i]], A[\pi[0]])$

$\Leftarrow q \leftarrow \text{Partition}(A, P, \pi)$

$\Rightarrow \Leftarrow \text{return } q$

Time complexity is $\Theta(n \log n)$.

Heap & cont

Heap :

→ heap is a data structure represented by an array and can be viewed as a nearly complete binary tree.

*-In the tree, each node corresponds to the value of the array. It is annotated with two elements :

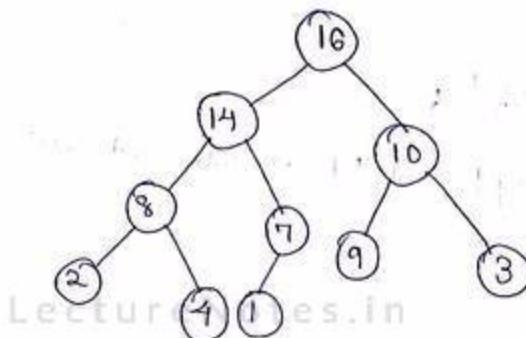
(i) array length given as length $\text{length}[A]$

(ii) heap size given as heap-size $\text{heap-size}[A]$.

*- $\text{A}[1]$ is the root of the tree.

- * For an element (i°), its parent resides at $\lfloor \frac{i}{2} \rfloor$
except the root.
- Parent (i°)
returns $\lfloor \frac{i}{2} \rfloor$;
- * For node (i°), its left child resides at $2i^{\circ}$
location.
left(i°)
returns $2i^{\circ}$;
- * For node (i°), its right child resides at $(2i^{\circ} + 1)$
location.
right(i°)
returns $(2i^{\circ} + 1)$;
- * There are two types of heaps:
i) Max heap
ii) Min heap
- i) Max Heap:
 $| \text{Parent}(i^{\circ}) | \geq | \text{Left}(i^{\circ}) | \geq | \text{Right}(i^{\circ}) |$
- ii) Min Heap:
 $| \text{Parent}(i^{\circ}) | \leq | \text{Left}(i^{\circ}) | \leq | \text{Right}(i^{\circ}) |$
- Leaf starts at $\lfloor \frac{n}{2} \rfloor + 1$.
- 1-1-1 idea for min heap construction

	1	2	3	4	5	6	7	8	9	10
A	16	14	10	8	7	9	3	2	4	1



In almost complete binary tree, given no. of nodes = $2^h - 1$.

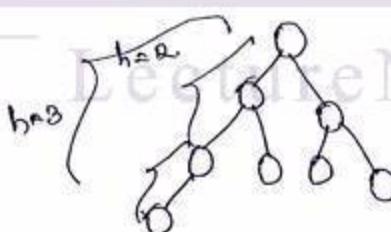
2018/2015

24/8/2015

(i) Min-heap ($A[\text{Parent}(i)] \leq A[i]$)

(ii) Max-heap ($A[\text{Parent}(i)] \geq A[i]$)

An almost complete binary tree of height (h) is complete upto height ($h-1$) and the tree is filled from left to right.



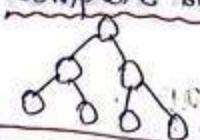
[$h=3$]

level 0

level 1

level 2

Complete binary tree



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

* Minimum number of nodes = 2^h

(filled upto $h-1$: nodes = $2^{h-1} \sim 1$
 $\approx 2^h - 1$)

for min almost complete binary tree = $2^h - 1 + 1$
 $= 2^h$

* Maximum number of nodes = $2^{h+1} - 1 \sim$
 $= 2^{h+1} - 2$



*- Heap ^{sort} uses three procedures (for sorting) as
input array :
(i) Max-heapify . ③
(ii) Build-Max-heap() . ②
(iii) Heap-Sort() . ①

(i) Max-heapify (A, i) : In which the heapify procedure
is the index on which will be applied.

1: $a \leftarrow \text{left of } i^{\circ}$. returns a°

2: $a_1 \leftarrow \text{right}(i^{\circ})$. returns (a_1) .

3: If $a \leq \text{heapsize}(A)$ & $a_1 \geq a$ then $a \leftarrow a_1$

4: Then largest = a

5: Else largest = i°

6: If $a \leq \text{heapsize}(A)$ & $a \geq \text{largest}$ then $a \leftarrow \text{largest}$

7: largest = a°

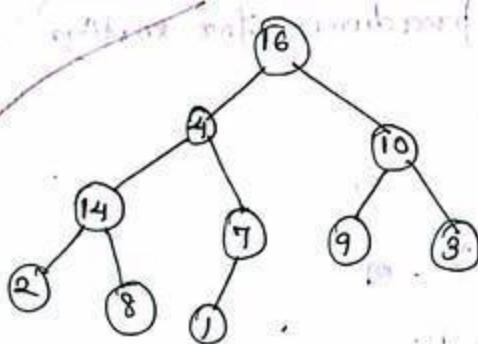
8: If largest $\neq i^{\circ}$

swap $(A[i], A[\text{largest}])$

9: max-heapify ($A, \text{largest}$)

10: max-heapify (A, i°)

11: End



16	4	10	14	7	9	3	2	8	1
1	2	3	4	5	6	7	8	9	10

16 > 4 > 14

Largest = 16 > 14

largest > A[16]

largest > ith

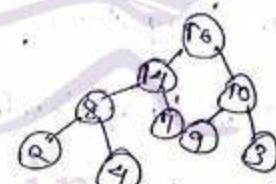
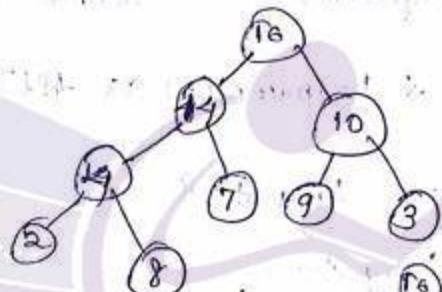
swap A[ith], A[Largest]

4 > 14 > 8

Largest = 4

A[16] > largest

swap A[16], A[largest]



i > heapsize

or i > heapsize

* When i is leaf node, the left and right child exceeds the array size.

Time Complexity of Max-heap

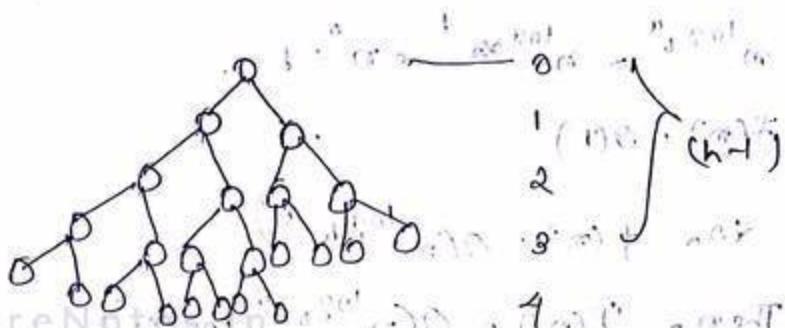
* Finding largest and exchange time is some constant C.

In worst case, the recursion will go up to $\log n$.

Worst case

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

height, $h = \log n$



Total no. of nodes upto height $(h-1)$ is:

$$2^{h-1} - 1 = 2^h \text{ (ignoring one)}$$

No. of nodes at level $2^l = 2^{h-1}$

To find nodes at level (h) ($\because h = h-1$)

$$\therefore 2^h = 2^h \cdot 2^1 = 2^{h+1} = 2^{h-2}$$

Total nodes of last level when tree is half full at last level

$$= 2^{h-2} \times 2 = 2^{h-1}$$

Total nodes of the tree

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

$$\Rightarrow 2^h + 2^{h-1} = n \quad (\because \text{total no. of nodes} = n)$$

$$\Rightarrow 2^{h-1}(2+1) = n$$

$$\therefore 2^{h-1} = \frac{n}{3}$$

$$\Rightarrow 2^h = 2n/3$$

$$T(n) = T(2n/3) + O(1)$$

Here, $a = 1$, $b = \frac{3}{2}$.

$$m \log_b a \approx m \log_2 1 \approx m \approx 1$$

$T(n) \approx O(1)$

So, $T(n) \approx O(m \log_b a)$

Then, $T(n) \approx O(m \log_b a \log m)$

$\Rightarrow T(n) \approx O(1 \cdot \log n)$

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

27/8/2015

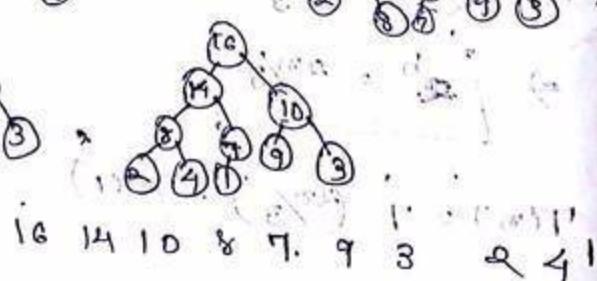
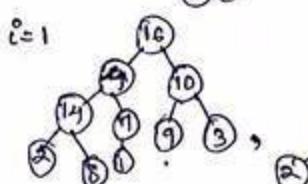
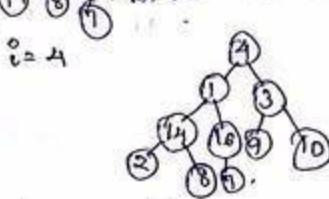
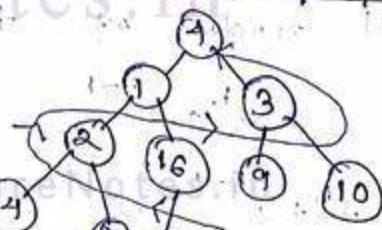
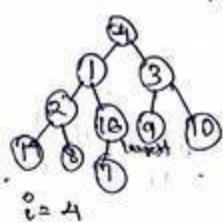
(ii) Build Max-heap (A)

Given an array (A). Construct a max-heap from the input array (A).
A. (A) constructs or converts to a max-heap (A').

Let A

4	1	3	2	16	9	10	14	8	7	1
---	---	---	---	----	---	----	----	---	---	---

$i = 10$ ($\frac{n}{2}$), $\rightarrow 16$.



$\Leftarrow \text{heapsize}(A) \leftarrow \text{length}(A)$

$\Leftarrow \text{for } i = \frac{n}{2} \text{ down to 1}$

$\Leftarrow \text{Max-heapify}(A, i)$

Properties of heap

An element heap has height $\lfloor \log_2 n \rfloor$ and atmost $\lceil \frac{n}{2^h+1} \rceil$ nodes at any height.

Time Complexity of build max-heap

$$T(n) = \sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{n}{2^{h+1}} \quad O(n \log n)$$

$$= \sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{n}{2^{h+1}} \times \frac{1}{2^h}$$

$$= \frac{n}{2} \left(\sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{1}{2^h} \right)$$

$$= n/2 \times 2 \quad \left(\because \sum_{h=0}^{\infty} \frac{1}{2^h} = 2 \right)$$

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

LectureNotes.in

31/8/2015-

iii) Heapsort(A)

$\Leftarrow \text{Build_max_heap}(A)$

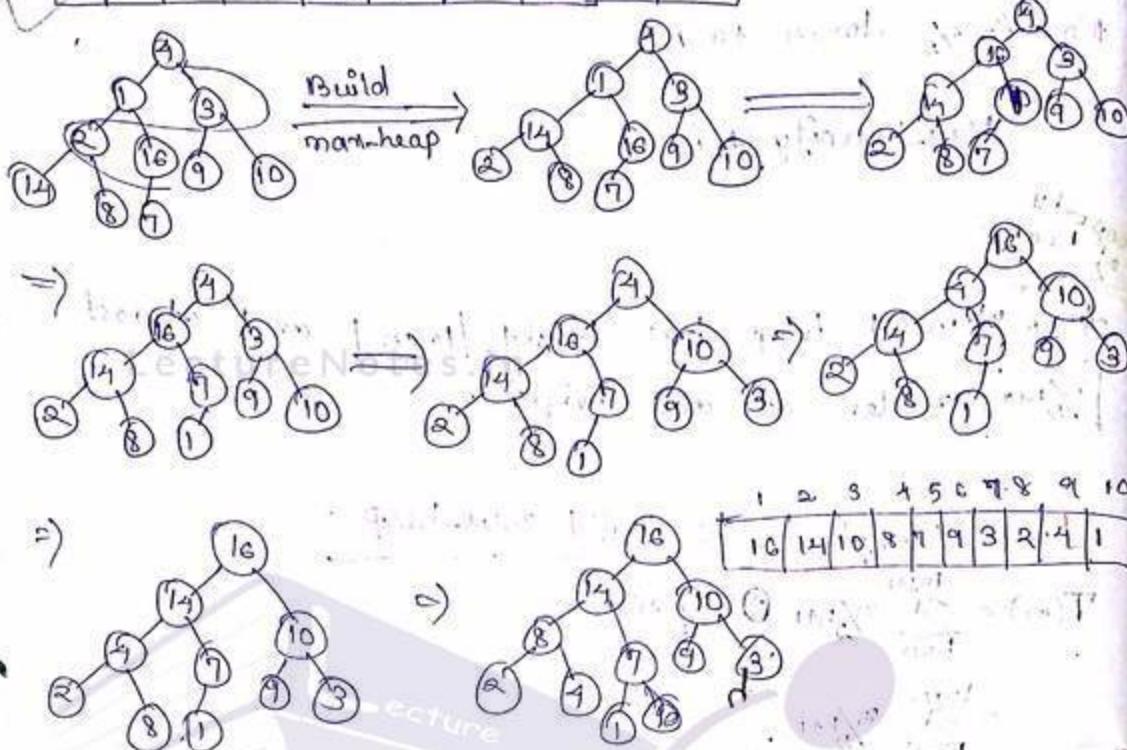
$\Leftarrow \text{for } (i = \text{length}[A] \text{ down to 2})$

\Leftarrow exchanges $A[1:i] \leftrightarrow A[i:n]$

$\Leftarrow \text{heapsize}[A] \leftarrow \text{heapsize}[A]-1$

$\Leftarrow \text{Max-heapify}(A, 1)$

1	2	3	4	5	6	7	8	9	10
4	1	3	2	16	9	10	14	8	7



for ($i = \text{length } A[1] + 1$ to 2)

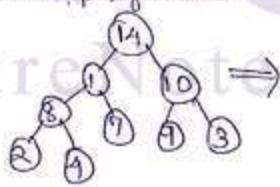
exchange $A[1]$, $A[i]$ \Rightarrow $\boxed{10}$

length $A[1] = 10$.

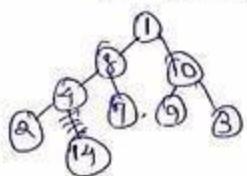
Max-heapify (A[1])



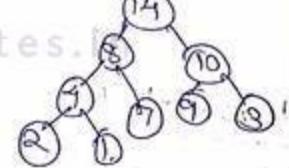
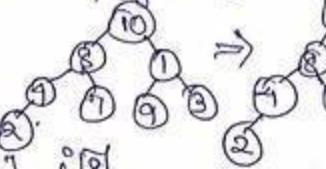
1	2	10	8	7	9	3
2	4	16				



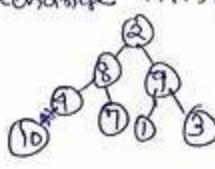
exchange $A[1]$, $A[1]$ \Rightarrow $\boxed{9}$



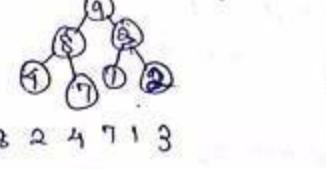
Max-heapify (A[1])



exchange $A[1]$, $A[1]$ \Rightarrow $\boxed{8}$

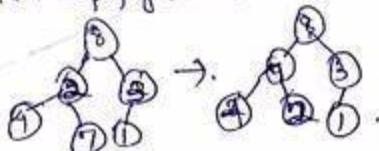


Max-heapify (A[1])



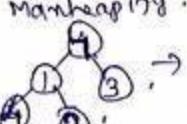
1 8 2 4 7 1 3

max-heapify (A[1:5])

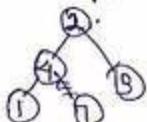


exchange

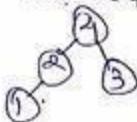
A[1:5], A[2:5]
max-heapify.



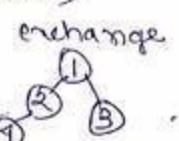
exchange A[1:5], A[2:5]



max-heapify (A[3:5])



exchange



max-heap



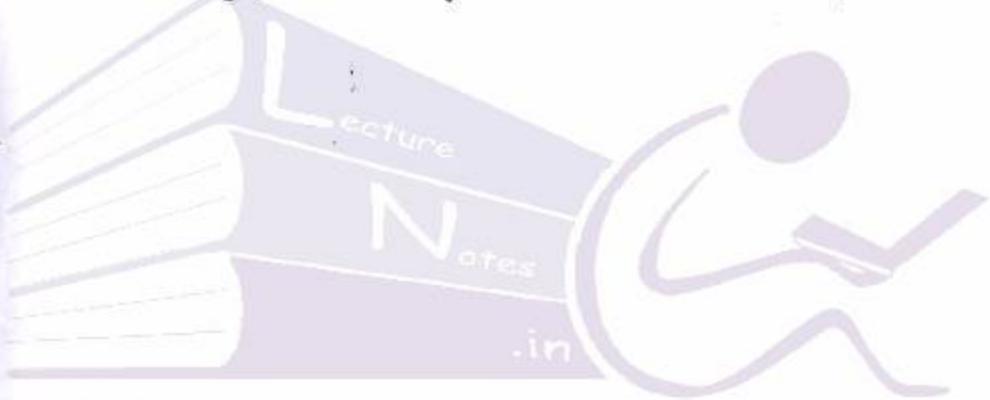
Time Complexity of heap-sort

line 2 : $O(n)$

line 3-5 : $(n-1)\log n = n\log n - \log n = O(n\log n)$,

Total time = $O(n) + O(n\log n) = O(n\log n)$

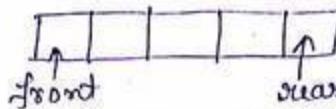
$O(n^2 \cdot \log n)$
 $= O(n^2)$.



LectureNotes.in

LectureNotes.in

Priority Queue → used in job scheduler.

Queue is a data structure, 

Insertion takes place at the rear end.

Deletion takes place from the front end.

* In priority queue, each element is associated with a priority ^{or key}, which decides the priority of that element.

* More prior element is deleted first.

3	2	1	
front		rear	

Job 1 → 3
Job 2 → 2,
Job 3 → 1

Job 4 → 4. (higher priority)

1	3	2	1
↑	↑	↑	↑

rear.

(as 4 should be at front, increment rear)

Job 5 → 6.

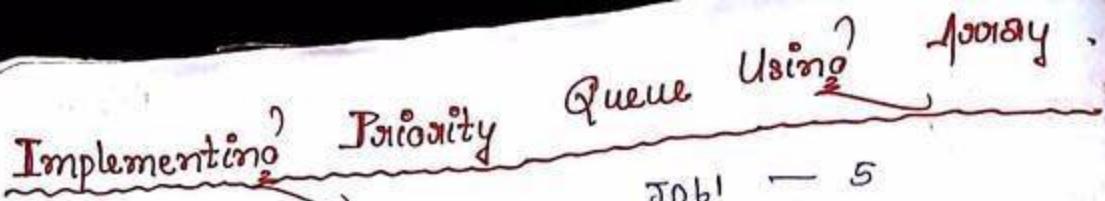
6	4	3	2	1
↑	↑	↑	↑	↑

Time taken to insert into the queue is

$O(n) \rightarrow$ for worst case as $(n-1)$ comparison and shifting are required.

Time taken for deletion : $O(1) \approx O(n)$.

Best case : $O(1) = O(n)$



Inversion takes $O(1)$ constant time

LectureNotes.in

Deletion

- * First search the element with highest priority. Then, take out that, after that rearrange.

Time taken is $O(n)$

- * If we are taking linear search, then the time taken for insertion and deletion is $O(n)$.

- * An efficient algorithm to implement priority queue are :

vi) Heap (Min / Max-heap).

The inversion or deletion time

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

LectureNotes.in

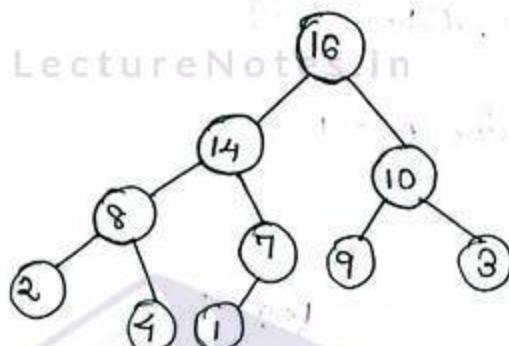
Implementation Priority Queue using Heap

* It uses three functions:

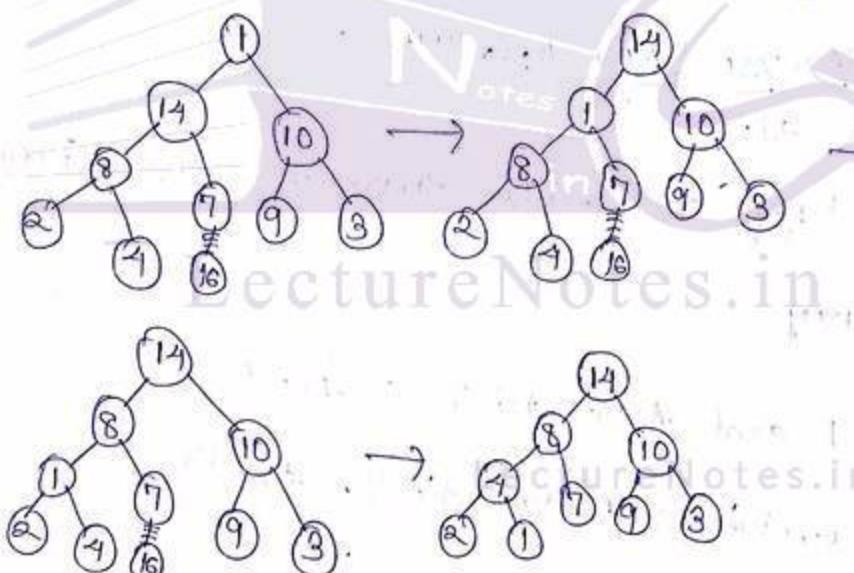
(i) heap-extract-max()

(ii) heap-increase-key()

(iii) max-heap-insert()



(iv) To find maximum element, apply heapsort.

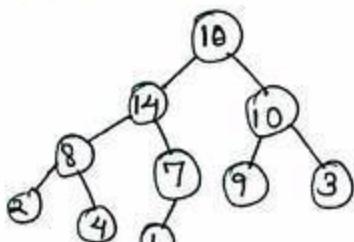


Time complexity for heap-extract-max()

is $O(\log n)$ as swapping only takes constant time; then rearranging will start from root to the leaf which is the height of the tree $\approx O(\log n)$.

- Heap-Extract-Max(A) \rightarrow A is a max-heap
- 1 if $\text{heapsize}(A) < 1$
 - 2 then Error "heap underflow"
 - 3 $\text{Max} \leftarrow A[1]$ last index of heap
 - 4 $\text{heapsize}(A) \leftarrow \text{heapsize}(A[1], \text{heapsize})$
 - 5 $\text{heapsize}(A) \leftarrow \text{heapsize}(A) - 1$
 - 6 $\text{Max-heapify}(A, 1)$
 - 7 Return max.

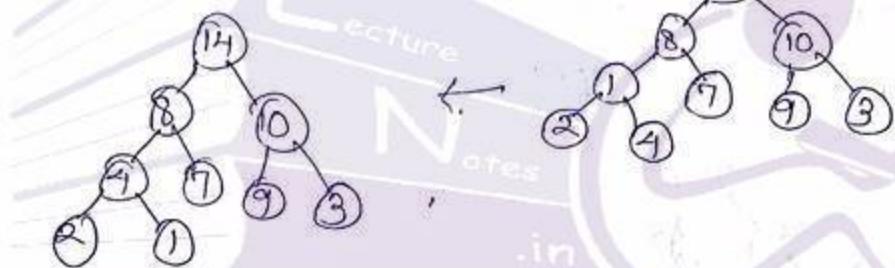
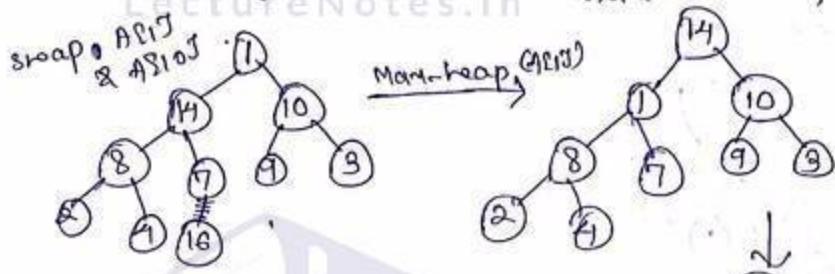
- Heap-increase-key(A, i, key)
- 1 if $\text{key} < A[i]$
 - 2 error "key is smaller than the current key"
 - 3 $A[i] \leftarrow \text{key}$
 - 4 while $i > 1$ and $A[\text{Parent}(i)] < A[i]$
 - 5 do exchange ($A[\text{Parent}(i)]$, $A[i]$)
 - 6 $i \leftarrow \text{Parent}(i)$



(Max-heap),

The value of the node represents the key value of a job.

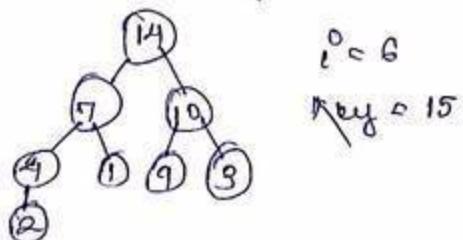
- 1st apply heap-extract-max. (extracts max value from the heap).



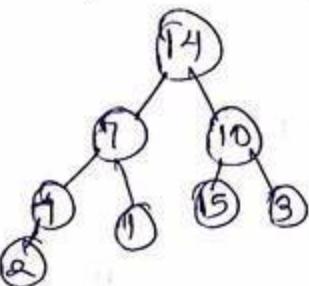
- 2nd apply heap-increase-key (it increases the key value of a node).

The input to procedure-2 is

- A, which is a max-heap.
- i , index of the node A whose key-value we want to increase.
- key, new key value which is of i .

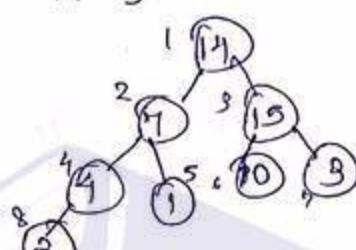


$\text{A}[0] \leftarrow \text{key}$
 $\text{A}[0] \leftarrow \text{key}(15)$



$i \leftarrow 1$
 $\text{parent}[i] \leftarrow \text{key}$

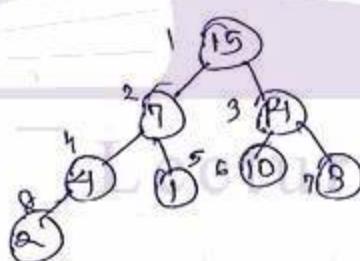
exchange



$i \leftarrow \text{parent}[i]$

$i \leftarrow 1$ $\text{parent} \leftarrow \text{key}$

sweep



$i \leftarrow \text{parent}[i]$

LectureNotes.in

3. Max-heap-insert(A, key):

increase heap size
assign \rightarrow to new node, with a priority,
Then, apply heap-increase-key.

$\Leftarrow \text{Neapsize}(A) \leftarrow \text{Neapsize}(A) + 1$

$\Leftarrow \text{↑}[\text{heapsize}] \leftarrow -\infty$

$\Leftarrow \text{Neap-Increase-Key}(A, \text{heapsize}, \text{key})$

7/9/2015

Time Complexity of Priority Queue

Insertion = $O(\log n)$

worst array or queue.
 $O(n)$

Deletion = $O(\log n)$

Increase Key = $O(\log n)$

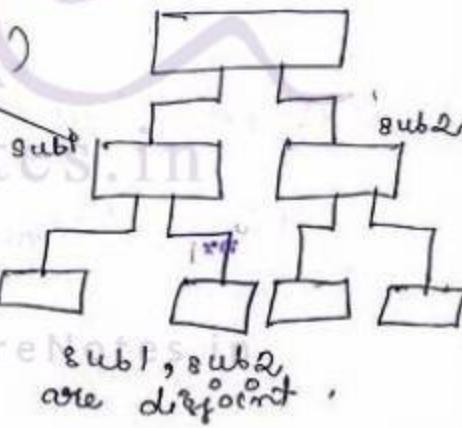
worst
Heap $\rightarrow O(1)$

Dynamaminc Programming

*- Dynamaminc programming approach solves the problems by combining the solution of sub-problems.

*- In divide and conquer approach, the sub-problems are independent/ disjoint and they solved recursively.

*- In dynamic programming approach, the sub-problems are over-lapping sub-problems, i.e., sub-problems share sub-sub problems.



- * - The dynamic programming is mainly used for optimization problems.
- Optimization problems have many solutions.
- * - Each solution has some value.
- * - Objective is to choose a solution among with maximum or minimum value.
- * - Four major steps of Dynamic programming approach.
 - i) characterize the structure of an optimal solution.
 - ii) recursively define the value of an optimal solution. (recurrence).
 - iii) Compute the value of an optimal solution (typically by bottom-up approach). (using table computation is done).
 - iv) construct an optimal solution from computed information. (back-track method as we are using bottom-up approach).

Matrix - Chain Multiplication

$$[C]_{p \times p} \quad [A]_{r \times m} \quad [B]_{m \times p}$$

↓

$$\text{Let } A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & & a_{2m} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & & a_{mm} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \vdots & \vdots & & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mp} \end{bmatrix}$$

$$C = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1p} \\ \vdots & \vdots & & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mp} \end{bmatrix}$$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + \dots + a_{1m}b_{m1}$$

$$c_{mp} = a_{m1}b_{1p} + a_{m2}b_{2p} + \dots + a_{mm}b_{mp}$$

Form

Matrix-Multiply (A, B)

Define C.

for (i=1 to m) , $(m \times p \times m)$.

for $j=1 to p$. $(p \text{ no. of } m \text{ multiplications}$
~~or~~ $(P \times m)$) .

for $i=1 to j$. $c_{ij} = 0$,

5. for $k=1$ to m . (multiplication)
6. $C[i][j] = C[i][j] + A[i][k] * B[k][j]$
 & addition.

7. end of for

8. end of for ,

LectureNotes.in

9. end of for .

10. end . $n^3 \text{ loop}$.

Time complexity for matrix multiplication.

Time complexity of matrix multiplication ,

$= O(n^3 p m)$, $n, p, m \rightarrow \text{scalars}$

$= O(n^3)$ (if all are square matrix
 $\text{of size } n \times n$) ,

For chain matrix multiplication

$\approx O(n^3(m))$

$= O(n^4)$.

Matrix-Chain Multiplication

Given a chain of n matrices $\langle A_1, A_2, \dots, A_n \rangle$.

Where for $i=1, 2, \dots, n$ matrix (A_i) has

dimension $P_{i-1} \times P_i$, fully parenthesize the

product A_1, A_2, \dots, A_n in a way that

minimizes the number of scalar multiplication.

$$(AB)_{pxn} = (A)_{p \times q} (B)_{q \times n}$$

$$T(n) = O(pq\alpha)$$

* Given a sequence or chain of matrices

$$A_1, A_2, \dots, A_n$$

Objective is to compute their product

$$A_1, A_2, \dots, A_n$$

To compute the chain of products we need to parenthesize the matrices.

Fully Parenthesized

A product of matrices is fully parenthesized if (i) it is a single matrix,

(ii) the product of two fully parenthesized matrix products are surrounded by parentheses.

$\text{Lcd}((A_1 A_2) (A_3 A_4)) \rightarrow$ fully parenthesized
 or, $(A_1 (A_2 A_3))$
 or, $((A_1 A_2) A_3)$

The fully parenthesized matrix product gives the order in which the matrices should be multiplied.

Ex: $A_1 A_2 A_3$.

Fully parenthesized product: $(A_1 (A_2 A_3))$
 or, $((A_1 A_2) A_3)$

* Since the matrix multiplication is associative, so, all parenthesis gives the same result.

Ex: $(A_1)_{p \times q} (A_2)_{q \times r} (A_3)_{r \times s}$.

$$A_1 A_2 A_3 = (A_1 A_2 A_3)_{p \times s} \\ = ((A_1 A_2) A_3)_{p \times s} = (A_1 (A_2 A_3))_{p \times s}$$

Given a sequence $\langle A_1 A_2 A_3 A_4 \rangle$ of matrices.
 The product $A_1 A_2 A_3 A_4$ can be parenthesized in how many ways?

$$\begin{aligned} & \overbrace{\quad}^{\text{Ans:}} ((A_1 A_2) A_3) A_4, ((A_1 (A_2 A_3)) A_4), \\ & ((A_1 (A_2 A_3) A_4)), (A_1 (A_2 (A_3 A_4))), \\ & ((A_1 A_2) (A_3 A_4)) \end{aligned}$$

Need of Parenthesizing?

$$(A_1)_{10 \times 100} \quad (A_2)_{100 \times 5} \quad (A_3)_{5 \times 50} \quad ((A_1 A_2) A_3) \\ (A_1 (A_2 A_3)).$$

Result: $(A_1 A_2 A_3)_{10 \times 50}$.

$$((A_1 A_2) A_3)$$

How many scalar multiplications we need?

To perform the matrix multiplication routine, will be called to perform the multiplication.

Ans: To compute $A_1 A_2$, in

number of scalar multiplications needs $\approx 10 \times 100 \times 5 = 5000$.

To compute $((A_1 A_2) A_3)$

$$= 10 \times 5 \times 50 \quad ((A_1 A_2)_{10 \times 5}) \\ \approx 2500,$$

Total no. of scalar multiplications

$$\approx 5000 + 2500$$

$$\approx 7500.$$

To compute $(A_1(A_2 A_3))$

Total computations needed to compute $(A_2 A_3)$

$$= 100 \times 5 \times 50 \approx 25,000$$

$$(A_1(A_2 A_3)) = 10 \times 100 \times 50 = 50,000$$

Total no. of scalar multiplications = 75,000

Now, the 1st one is 10 times faster than
2nd one.

9/9/2015

*- The parenthesization of the matrices which gives lowest number of scalar multiplication is called optimal parenthesization.

Objective of Matrix Chain Multiplication Problem -

To find out the optimal parenthesization.

Matrix Chain Multiplication Problem -

Given a chain of n matrices

$$\langle A_1, A_2, \dots, A_n \rangle$$

where for $i=1, 2, \dots, n$ matrix (A_i) has dimension $P_{i-1} \times P_i$

Fully parenthesize the product $A_1 A_2 \dots A_m$ in a way that minimizes the number of scalar multiplication.

$$P_0 \times P_1$$

$$P_1 \times P_2$$

$$\vdots$$

$$P_{m-1} \times P_m$$

\rightarrow Resulting matrix $((A_1 A_2) A_3) \dots P_0 \times P_m$.

*- The matrix chain multiplication can be solved by dynamic programming approach.

(*) Characterize the structure of an optimal solⁿ.

$$\begin{aligned}
 & A_i A_{i+1} \dots A_j \xrightarrow{\text{splitting}} A_i \dots A_j \\
 & ((A_i \dots A_{i+1}) (A_{i+2} \dots A_j)) \\
 & \text{splitting occurs at } A_i \otimes A_{i+1} \\
 & ((A_i \dots A_k) (A_{k+1} \dots A_j)) \\
 & = ((A_{i \dots k}) (A_{k+1 \dots j})) \\
 & \qquad \qquad \qquad \xrightarrow{P_{i \dots k} \times P_{k+1 \dots j}} O(P_{i \dots k} \times P_{k+1 \dots j})
 \end{aligned}$$

10/9/2015.

Note:
 Exhaustive searching of optimal parenthesization from all possible parenthesization doesn't yield an efficient algorithm.
 So, a dynamic programming approach

will be used to find out the optimal parenthesization

Let $A_{i_1 \dots i_j}$ where $i \leq j$ is used to denote the matrix results from evaluating
 $A_{i_1}, A_{i_1+1}, \dots, A_{i_j}$.

LectureNotes.in

* When $i < j$, then any parenthesization of the product A_i, A_{i+1}, \dots, A_j must split the product between A_k and A_{k+1} for some integer (k) in the range $i \leq k \leq j$.

* The cost of computing $A_{i_1 \dots i_j}$ = cost of computing $A_{i_1 \dots i_k}$
+ cost of computing $A_{k+1 \dots i_j}$
+ cost of computing their product.

(ii) Characteristics of Optimal Solution .

* Let an optimal parenthesization of $A_{i_1 \dots i_j}$ splits the product between A_k and A_{k+1} .

Then, the parenthesization of the prefix subchain $A_i \dots A_k$ within this optimal parenthesization of A_i, A_{i+1}, \dots, A_j must be

an optimal parenthesization.

* If the parenthesization of $A_1 \dots A_R$ is not optimal, then there were a less costly way to parenthesize A_1, A_{21}, \dots, A_R . Substituting that parenthesis in the optimal parenthesization of $A_1 \dots A_R$ will definitely lower the cost of the parenthesization. which is a contradiction.

* The optimal cost of $A_1 \dots A_R$
= optimal cost of parenthesization of $A_1 \dots A_k$
+ optimal cost of parenthesization of $A_{k+1} \dots A_R$
+ computational cost of their product.

* Now, we can construct the optimal solⁿ of the problem from optimal solⁿs to sub-problems.

The recursive solution

Let $m[i, j]$ be the minimum number of scalar multiplication needed to compute $A_i \dots A_j$.

Then, $m[i, n]$ gives the cheapest cost or optimal cost of finding $A_1 \dots A_n$.

$m_{i,j} = 0$ (single matrix so, cost = 0)
(no multiplication).

for $i = j$,

* If $i = j$, the problem is trivial i.e.,
the chain consists of just one matrix.

∴ no scalar multiplication needed

compute the product.

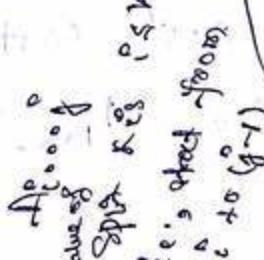
* Assuming the optimal parenthesization of
the chain $A_1 A_2 \dots A_K$ only for $i < j$ splits between the
 A_i and A_{i+1} .

Then, $m_{i,j} = m_{i,k} + m_{k+1,j} + p_{i,k} \times p_k \times p_j$.

$$\begin{array}{c} \downarrow \\ A_{i+1} \dots A_K \\ \downarrow \\ A_{i+1} \times p_k \quad p_k \times p_k \\ \text{(} A_{i+1} \times p_k \text{)} \cdot \text{(} A_{i+1} \dots j \text{)} \quad p_k \times p_j \end{array}$$

The recurrence to compute optimal parenthesization
for matrix chain multiplication is

$$m_{i,j} = \begin{cases} 0 & \text{if } i = j \\ m_{i,k} + m_{k+1,j} + p_{i,k} \times p_k \times p_j & \text{if } i < j \end{cases}$$



Depending on K ,
the no. of optimal
solns depend on

Objective of matrix chain multiplication is to find optimal solⁿ with parenthesization. 14/9/2015.

Find Optimal Solⁿ:

Let n=6.

The chain of the matrices is

$A_1, A_2, A_3, A_4, A_5, A_6$,

Our objective is to find the parenthesization of the matrices so, that the number of scalar multiplication will be minimized

	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆
$A_1 : 80 \times 35$ P ₀ : 80 $A_2 : 35 \times 15$ P ₁ : 35 $A_3 : 15 \times 5$ P ₂ : 15 $A_4 : 5 \times 10$ P ₃ : 5 $A_5 : 10 \times 20$ P ₄ : 10 $A_6 : 20 \times 25$ P ₅ : 20 P ₆ : 25	30	35	15	5	10	20	25

The optimal value of the chain

$$A_1 \cdot A_2 \cdots A_6 = A_{1 \dots 6}$$

which gives the minimum scalar multiplication to compute the product

$$A_{1 \dots 6} \text{ is } m_{1,6}.$$

If there are n number of matrices

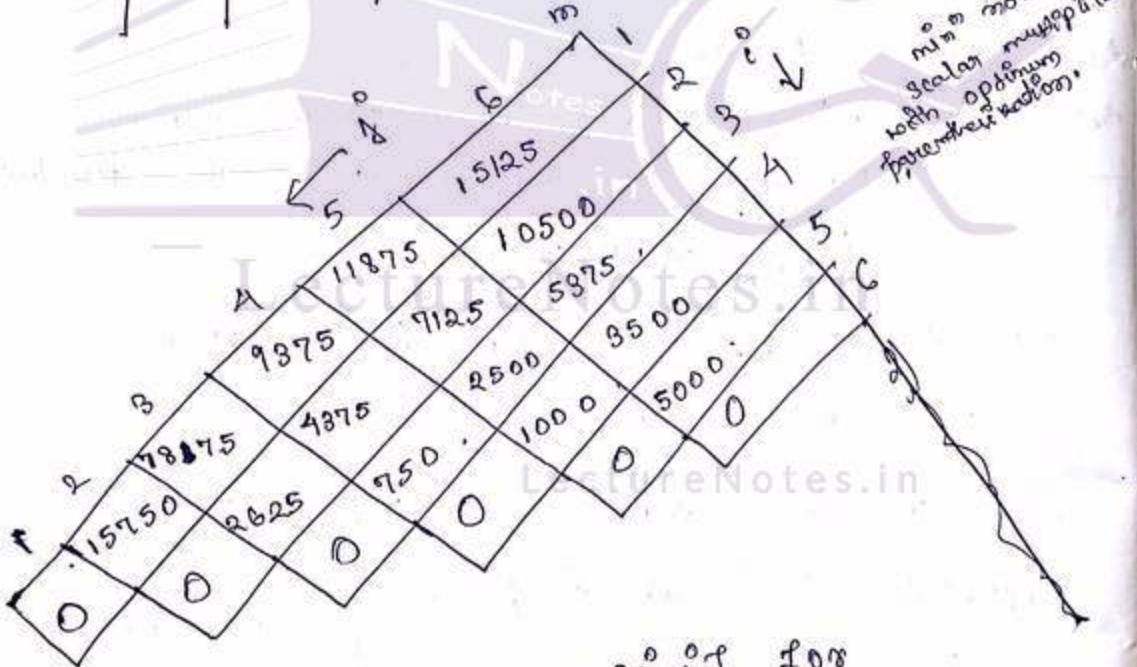
$$m_{1,n}.$$

$m_{5,2}$ is invalid as $5 > 2$.

$m_{5,5}$ = 0, as $5 = 1$, (as no scalar multiplication).

* A dynamic programming table of size $(n \times n)$ is computed to find out $m_{1,n}$.

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	5	10	15	20	25	30
2	0	10	20	30	40	50	60
3	0	15	30	45	60	75	90
4	0	20	40	60	80	100	120
5	0	25	50	75	100	125	150
6	0	30	60	90	120	150	180



for Compute the value $m_{i,j}^{\text{opt}}$ for

$$i = 1 \text{ to } 6$$

$$j = 1 \text{ to } 6$$

with bottom-up approach.

min no. of
scalar multiplications
with optimum
parenthesization.

$m_{1,1}^{1,1}, m_{2,2}^{2,2}, m_{3,3}^{3,3}, m_{4,4}^{4,4}, m_{5,5}^{5,5}, m_{6,6}^{6,6} = 0$

* - Compute $m_{1,2}^{1,2}$ means $\pi_1 \times \pi_2$

$i=1, j=2$
Using recurrence
 $K=1$

The parenthesis can be split at point $K=1$

$$m_{1,2}^{1,2} = m_{1,1}^{1,1} + m_{2,2}^{2,2} + P_0 \cdot P_1 \cdot P_2$$

$$\begin{aligned} &= 0 + 0 + (30)(35)(15) \\ &= 15750 \end{aligned}$$

$$\begin{array}{r} 35 \\ \times 15 \\ \hline 175 \\ 35 \\ \hline 525 \\ \times 10 \\ \hline 525 \\ 308 \\ \hline 1575 \end{array}$$

$m_{2,3}^{2,3}$

$i=2, j=3$

$K=2$

$$m_{2,3}^{2,3} = m_{2,2}^{2,2} + m_{3,3}^{3,3} + P_1 \cdot P_2 \cdot P_3$$

$$\begin{aligned} &= 0 + 0 + (35)(15)(5) \\ &= 2625 \end{aligned}$$

$$\begin{array}{r} 35 \\ \times 15 \\ \hline 175 \\ 35 \\ \hline 525 \\ \times 5 \\ \hline 2625 \end{array}$$

$m_{3,4}^{3,4}$

$i=3, j=4$

$K=3$

$$\begin{aligned} m_{3,4}^{3,4} &= m_{3,3}^{3,3} + m_{4,4}^{4,4} + P_2 \cdot P_3 \cdot P_4 \\ &\approx 750 \end{aligned}$$

$m_{4,5}^{4,5}$

$i=4, j=5$

$K=4$

$$\begin{aligned} m_{4,5}^{4,5} &= m_{4,4}^{4,4} + m_{5,5}^{5,5} + P_3 \cdot P_4 \cdot P_5 \\ &= 5 \times 10 \times 20 = 1000 \end{aligned}$$

$m[5,6]$

$$i=5, j=6$$

$$R = \boxed{5},$$

$$m[5,6] = m[5,5] + m[6,6] + p_4 p_5 p_6$$

$$= 10 \times 20 \times 25$$

$$= 5000$$

LectureNotes.in

15/9/2015

$m[1,3]$

$$i=1, j=3$$

$$R = 1, 2$$

$$m[1,3] = \min_{R=1} (m[1,3], m[1,3])$$

$$\begin{array}{r} 35 \\ 105 \\ \hline 5250 \end{array}$$

$m[1,3]$ when $R=1$

$$= m[1,1] + m[2,3] + p_0 p_1 p_3$$

$$\begin{array}{r} 2625 \\ 3250 \\ \hline 7875 \end{array}$$

$$= 0 + 2625 + (30)(35)(5)$$

$$= 18375$$

$m[1,3]$ when $R=2$

$$= m[1,2] + m[3,3] + p_0 p_2 p_3$$

$$\begin{array}{r} 75 \\ 225 \\ \hline 15750 \\ 2250 \\ \hline 18000 \end{array}$$

$$= 15750 + 30 \times 15 \times 5$$

$$= 18000$$

of $m[1,3]$ when $\boxed{R=1}$ gives optimal

no. of scalar multiplication, so we will consider this one

$m_{12,4J}$

$\ell=2 \quad j=4$

$R=2, 3$

$$m_{12,4J} = \min_{R=2} (m_{R,4J}, m_{R,4J})$$

$m_{12,4J}$ when $R=2$

$$= m_{12,2J} + m_{12,4J} + P_1 P_2 P_4$$

$$= 0 + 750 + 35 \times 15 \times 10$$

$$= 6000$$

$$\begin{array}{r} 35 \\ \times 15 \\ \hline 175 \\ 175 \\ \hline 5250 \\ 750 \\ \hline 6000 \end{array}$$

$m_{12,4J}$ when $\boxed{R=3}$

$$m_{12,4J} = m_{12,3J} + m_{12,4J} + P_1 P_3 P_4$$

$$= 2625 + 35 \times 5 \times 10$$

$$= 4375$$

$$\begin{array}{r} 2625 \\ \times 5 \\ \hline 1375 \end{array}$$

$m_{12,5J}$

$\ell=3, \quad j=5$

$R=3, 4$

$$m_{12,5J} = \min_{R=3} (m_{R,5J}, m_{R,5J})$$

$R=4$

$m_{12,5J}$ when $\boxed{R=3}$

$$= m_{12,3J} + m_{12,5J} + P_2 P_3 P_5$$

$$= 0 + 1000 + 15 \times 5 \times 20$$

$$= 2500$$

$$\begin{array}{r} 1000 \\ \times 15 \\ \hline 15000 \\ 15000 \\ \hline 25000 \end{array}$$

$m_{12,5J}$ when $R=4$

$$= m_{12,4J} + m_{12,5J} + P_2 P_4 P_5$$

$$= 750 + 15 \times 10 \times 20$$

$$= 3750$$

$$\begin{array}{r} 750 \\ \times 10 \\ \hline 7500 \\ 30000 \\ \hline 37500 \end{array}$$

$$m\Omega_{4,6}^J, \begin{matrix} \\ \circ \\ \circ \end{matrix}$$

$$R = 4, 5$$

$$R = 4, 5$$

$$m\Omega_{4,6}^J = \min_{R=4,5} (m\Omega_{4,6}^J, m\Omega_{4,6}^J)$$

$$m\Omega_{4,6}^J \text{ when } R=4$$

$$= m\Omega_{4,1}^J + m\Omega_{5,6}^J + P_3 P_4 P_6$$

$$\begin{array}{r} 252 \\ 1250 \\ \hline 5000 \\ \hline 6250 \end{array}$$

$$= 5000 + (5)(10)(25)$$

$$= 6250$$

$$m\Omega_{4,6}^J \text{ when } \boxed{R=5}$$

$$= m\Omega_{4,5}^J + m\Omega_{6,6}^J + P_3 P_5 P_6$$

$$\begin{array}{r} 251 \\ 250 \\ \hline 500 \\ \hline 500 \end{array}$$

$$= 1000 + (5)(20)(25)$$

$$\begin{array}{r} 1252 \\ 6250 \\ \hline 1000 \end{array}$$

$$= 3500$$

$$m\Omega_{1,4}^J$$

$$R = 1, 2, 3$$

$$m\Omega_{1,4}^J = \min_{R=1} (m\Omega_{1,4}^J, m\Omega_{1,4}^J, m\Omega_{1,4}^J)$$

$$m\Omega_{1,4}^J = \min_{R=2} (m\Omega_{1,4}^J, m\Omega_{1,4}^J, m\Omega_{1,4}^J)$$

$$m\Omega_{1,4}^J \text{ when } R=1$$

$$m\Omega_{1,1}^J + m\Omega_{2,3}^J + m\Omega_{3,4}^J + R_0 P_1 P_4$$

$$\begin{array}{r} 951 \\ 10500 \\ \hline 4975 \\ \hline 14875 \end{array}$$

$$= m\Omega_{1,1}^J + m\Omega_{2,3}^J + P_0 P_1 P_4$$

$$= 0 + 4975 + (90)(35)(10)$$

$$= 14875$$

$m\varphi_{1,4}J$ when $R=3$

$$= m\varphi_{1,3}J + m\varphi_{4,4}J + P_0 P_3 P_4$$

$$= 7875 + (80)(5)(10)$$

$$= 9375$$

$$\begin{array}{r} 3 \\ \hline 1500 \\ 7875 \\ \hline 9375 \end{array}$$

when $R=2$

$$m\varphi_{1,4}J = m\varphi_{1,2}J + m\varphi_{3,4}J + P_0 P_2 P_4$$

$$= 15750 + 750 + (30)(15)(10)$$

$$= 21,000$$

$$\begin{array}{r} 151 \\ \hline 15750 \\ 75 \\ \hline 15250 \\ 15750 \\ \hline 21000 \end{array}$$

$m\varphi_{2,5}J$

$R=2$ $R=5$

$R=2,3,4$

$$m\varphi_{2,5}J = \min_{R=2} (m\varphi_{2,5}J, m\varphi_{2,5}J, m\varphi_{2,5}J)$$

$m\varphi_{2,5}J$ when $R=2$

$$= m\varphi_{2,2}J + m\varphi_{3,5}J + P_1 P_2 P_5$$

$$= 2500 + (35)(15)(20)$$

$$= 13,000$$

$$\begin{array}{r} 35 \\ \hline 125 \\ 35 \\ \hline 525 \\ 14500 \\ 2500 \\ \hline 13000 \end{array}$$

$m\varphi_{2,5}J$ when $R=3$

$$= m\varphi_{2,3}J + m\varphi_{4,5}J + P_1 P_3 P_5$$

$$= 2625 + 1000 + (35)(5)(20)$$

$$= 7125$$

$$\begin{array}{r} 35 \\ \hline 1751 \\ 21 \\ \hline 3500 \\ 1000 \\ 2625 \\ \hline 7125 \end{array}$$

$m\varphi_{2,5}J$ when $R=4$

$$= m\varphi_{2,4}J + m\varphi_{5,5}J + P_1 P_4 P_5$$

$$= 4375 + (35)(10)(20)$$

$$= 11375$$

$$\begin{array}{r} 35 \\ \hline 7000 \\ 1375 \\ \hline 11375 \end{array}$$

$m\{3,6\}$

$P_3 \rightarrow P_6$

$R = 3, 4, 5$

$$m\{3,6\} = \min_{R=3} (m\{3,6\}, m\{3,6\} + m\{3,6\})$$

$$m\{3,6\} = R = 5$$

$m\{3,6\}$ when $R = 3$

$$= m\{3,3\} + m\{4,6\} + P_2 P_3 P_6$$

$$= 3500 + (15)(5)(25)$$

$$\approx 5375$$

$$\begin{array}{r} 15 \\ \times 5 \\ \hline 75 \\ 75 \\ \hline 5375 \end{array}$$

$m\{3,6\}$ when $R = 4$

$$= m\{3,4\} + m\{5,6\} + P_2 P_4 P_6$$

$$= 750 + 5000 + (15)(10)(25)$$

$$= 9500$$

$$\begin{array}{r} 125 \\ \times 15 \\ \hline 625 \\ 3750 \\ \hline 9500 \end{array}$$

$m\{3,6\}$ when $R = 5$

$$= m\{3,5\} + m\{6,6\} + P_2 P_5 P_6$$

$$= 2500 + (15)(20)(25)$$

$$= 10000$$

$$\begin{array}{r} 125 \\ \times 15 \\ \hline 625 \\ 3750 \\ \hline 9500 \end{array}$$

$m\{1,5\}$

$P_1 \rightarrow P_5$

$R = 1, 2, 3, 4$

$$m\{1,5\} = \min_{R=1} (m\{1,5\}, m\{1,5\}, m\{1,5\} + m\{3,5\})$$

$$m\{1,5\} = R = 4$$

$$\begin{array}{r} 15 \\ \times 2 \\ \hline 300 \\ 1500 \\ \hline 2500 \\ 2500 \\ \hline 10000 \end{array}$$

$m\varphi_{1,5}^J$ when $R=1$

$$= m\varphi_{1,1}^J + m\varphi_{2,5}^J + P_0 \cdot P_1 \cdot P_5$$

$$= 7125 + (30)(35)(20)$$

$$= 28125$$

$\frac{35}{3}$
 $\times \frac{10}{10}$
 $\frac{2}{2}$
 $\frac{21}{21}$
 $\frac{0}{0}$
 $\frac{0}{0}$
 $\frac{7}{7}$
 $\frac{1}{1}$
 $\frac{2}{2}$
 $\frac{5}{5}$
 $\underline{\underline{28125}}$

$m\varphi_{1,5}^J$ when $R=2$

$$= m\varphi_{1,2}^J + m\varphi_{3,5}^J + P_0 \cdot P_2 \cdot P_5$$

$$\approx 15750 + \frac{2500}{2500} + (30)(15)(20)$$

$$= 25750 - 27250$$

$\frac{15}{3}$
 $\times \frac{1}{1}$
 $\frac{5}{5}$
 $\frac{1}{1}$
 $\frac{9}{9}$
 $\frac{0}{0}$
 $\frac{0}{0}$
 $\frac{2}{2}$
 $\frac{3}{3}$
 $\frac{0}{0}$
 $\frac{0}{0}$
 $\frac{1}{1}$
 $\frac{6}{6}$
 $\frac{5}{5}$
 $\frac{0}{0}$
 $\frac{1}{1}$
 $\frac{5}{5}$
 $\frac{7}{7}$
 $\frac{5}{5}$
 $\underline{\underline{27250}}$

$m\varphi_{1,5}^J$ when $R=4$

$$= m\varphi_{1,4}^J + m\varphi_{5,5}^J + P_0 \cdot P_4 \cdot P_5$$

$$= 9375 + (30)(10)(20)$$

$$= 15375$$

$\frac{9375}{2000}$
 $\underline{\underline{15375}}$

$m\varphi_{1,5}^J$ when $\boxed{R=3}$

$$= m\varphi_{1,3}^J + m\varphi_{4,5}^J + P_0 \cdot P_3 \cdot P_5$$

$$= 78.75 + \frac{100}{2500} + (30)(5)(20)$$

$$= 13375 - 11875$$

$\frac{15}{2}$
 $\frac{1}{1}$
 $\frac{0}{0}$
 $\frac{0}{0}$
 $\frac{8}{8}$
 $\frac{0}{0}$
 $\frac{0}{0}$
 $\frac{2}{2}$
 $\frac{5}{5}$
 $\frac{0}{0}$
 $\frac{7}{7}$
 $\frac{5}{5}$
 $\underline{\underline{375}}$

$\frac{900}{100}$
 $\frac{0}{0}$
 $\frac{0}{0}$
 $\frac{28}{28}$
 $\frac{7}{7}$
 $\frac{8}{8}$
 $\frac{7}{7}$
 $\underline{\underline{11875}}$

$m\varphi_{2,6}^J$

$R=2, 3, 4, 5$

$m\varphi_{2,6}^J = \min(m\varphi_{2,6}^J, m\varphi_{3,6}^J, m\varphi_{4,6}^J, m\varphi_{5,6}^J)$

$R=2$

$R=4$

$R=5$

$m\varphi_{2,6}^J$ when $R=2$

$$= m\varphi_{2,2}^J + m\varphi_{3,6}^J + P_1 \cdot P_2 \cdot P_6$$

$$= 5875 + (35)(15)(25)$$

$$= 18500$$

$\frac{35}{3}$
 $\frac{15}{15}$
 $\frac{2}{2}$
 $\frac{3}{3}$
 $\frac{25}{25}$
 $\frac{25}{25}$
 $\frac{1}{1}$
 $\frac{25}{25}$
 $\frac{18}{18}$
 $\frac{6}{6}$
 $\frac{0}{0}$
 $\frac{18}{18}$
 $\frac{1}{1}$
 $\frac{8}{8}$
 $\frac{1}{1}$
 $\frac{5}{5}$
 $\frac{3}{3}$
 $\frac{7}{7}$
 $\frac{5}{5}$
 $\underline{\underline{18500}}$

$m\Omega_{2,6}^f$ when $R=3$

$$= m\Omega_{2,3}^f + m\Omega_{4,6}^f + P_1 \cdot P_3 \cdot P_6$$

$$= 2625 + 3500 + (35)(5)(25)$$

$$= 10500$$

$$\begin{array}{r} 952 \\ 25 \\ \hline 70 \\ 1375 \\ 3500 \\ \hline 10500 \end{array}$$

$m\Omega_{2,6}^f$ when $R=1$

$$= m\Omega_{2,4}^f + m\Omega_{5,6}^f + P_1 \cdot P_4 \cdot P_6$$

$$= 4375 + 5000 + (35)(10)(25)$$

$$= 11125$$

$$\begin{array}{r} 450 \\ 5000 \\ \hline 4875 \\ 11125 \end{array}$$

$m\Omega_{2,6}^f$ when $R=5$

$$= m\Omega_{2,5}^f + m\Omega_{6,6}^f + P_1 \cdot P_5 \cdot P_6$$

$$= 7125 + (35)(20)(25)$$

$$= 10625$$

$$\begin{array}{r} 1951 \\ 3500 \\ \hline 7125 \\ 10625 \end{array}$$

$m\Omega_{1,6}^f$

$$i=1 \quad j=6$$

$R = 1, 2, 3, 4, 5$

$$m\Omega_{1,6}^f = \text{min}_{R=1}^{R=5} (m\Omega_{1,6}^f, m\Omega_{1,6}^f, m\Omega_{1,6}^f, m\Omega_{1,6}^f, m\Omega_{1,6}^f)$$

$m\Omega_{1,6}^f$ when $R=1$

$$= m\Omega_{1,1}^f + m\Omega_{2,6}^f + P_0 \cdot P_1 \cdot P_6$$

$$= 10500 + (30)(35)(25)$$

$$= 15750$$

$$\begin{array}{r} 1951 \\ 5250 \\ \hline 10500 \\ 15750 \end{array}$$

$m\Omega_{1,6}^f$ when $R=2$

$$= m\Omega_{1,2}^f + m\Omega_{3,6}^f + P_0 \cdot P_2 \cdot P_6$$

$$= 15750 + 5875 + (30)(15)(25)$$

$$= 32375$$

$$\begin{array}{r} 252 \\ 125 \\ 25 \\ \hline 375 \\ 11250 \\ 5875 \\ \hline 15750 \\ 32375 \end{array}$$

$m[1,6]$ when $R=3$

$$\begin{aligned}
 &= m[1,3] + m[4,6] + P_0 P_3 P_6 \\
 &= 7875 + 3500 + (30)(5)(25) \\
 &= 15125
 \end{aligned}$$

$$\begin{array}{r}
 15125 \\
 1251 \\
 \hline
 3750 \\
 3500 \\
 \hline
 7875 \\
 15125
 \end{array}$$

$m[1,6]$ when $R=4$

$$\begin{aligned}
 &= m[1,4] + m[5,6] + P_0 P_4 P_6 \\
 &= 9375 + 5000 + (30)(10)(25) \\
 &= 21875
 \end{aligned}$$

$$\begin{array}{r}
 25 \\
 7500 \\
 5000 \\
 9375 \\
 \hline
 21875
 \end{array}$$

$m[1,6]$ when $R=5$

$$\begin{aligned}
 &= m[1,5] + m[6,6] + P_0 P_5 P_6 \\
 &= 11875 + (30)(20)(25) \\
 &= 26875
 \end{aligned}$$

$$\begin{array}{r}
 75 \\
 15000 \\
 11875 \\
 \hline
 26875
 \end{array}$$

(ii) Find optimal soln from optimal value.

16/9/2015

*- Matrix (m) of matrix chain multiplication

problem gives the minimum number of scalar multiplication need to compute the product of chain of matrices A_1 to A_g , i.e., $A_1 \dots A_g$.

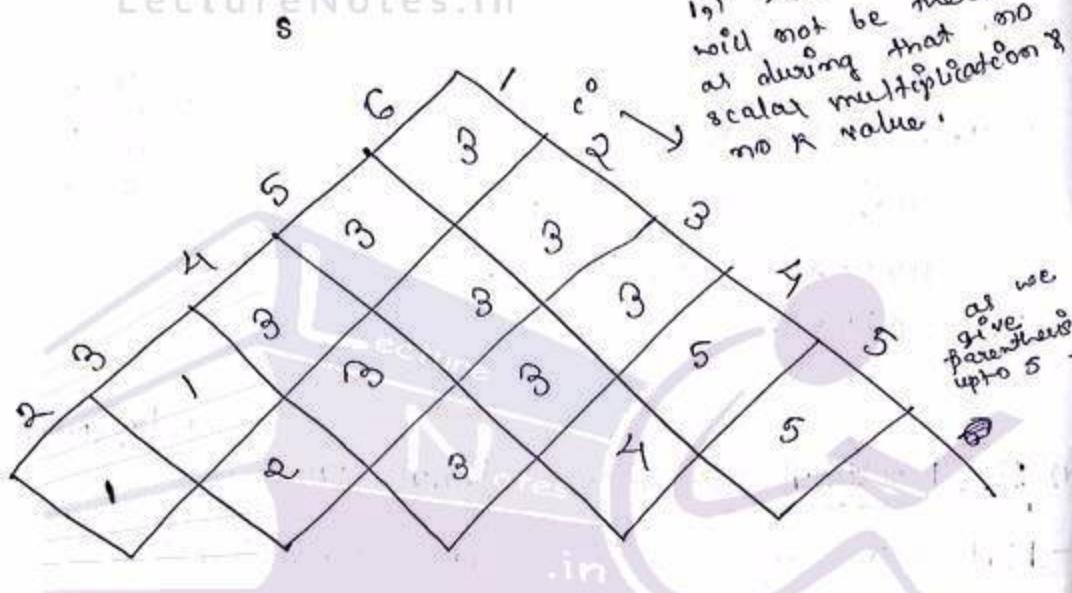
$m[1,6]$ i.e., $m[1,6]$ in the example gives the minimum number of scalar multiplication needed to compute the product of chain $A_1 \dots A_6$, i.e., $A_{1000} Q$.

$\therefore m[1,6] = 15125$

→ optimal value.

To get the optimal solution, another table $S[1000, 1000]$ is constructed.
 Every $S[i,j]$ will contain the value of k for which $m[i,j]$ gives minimum scalar multiplication.

LectureNotes.in



Algorithm to compute optimal parenthesis

1 Point optimal parenthesis (S, i, j)

2 If $i == j$

LectureNotes.in

3 Then print A_i

4 Else print 'C'

5 Point_optimal_parenthesis($S, i, S[i, j]$)

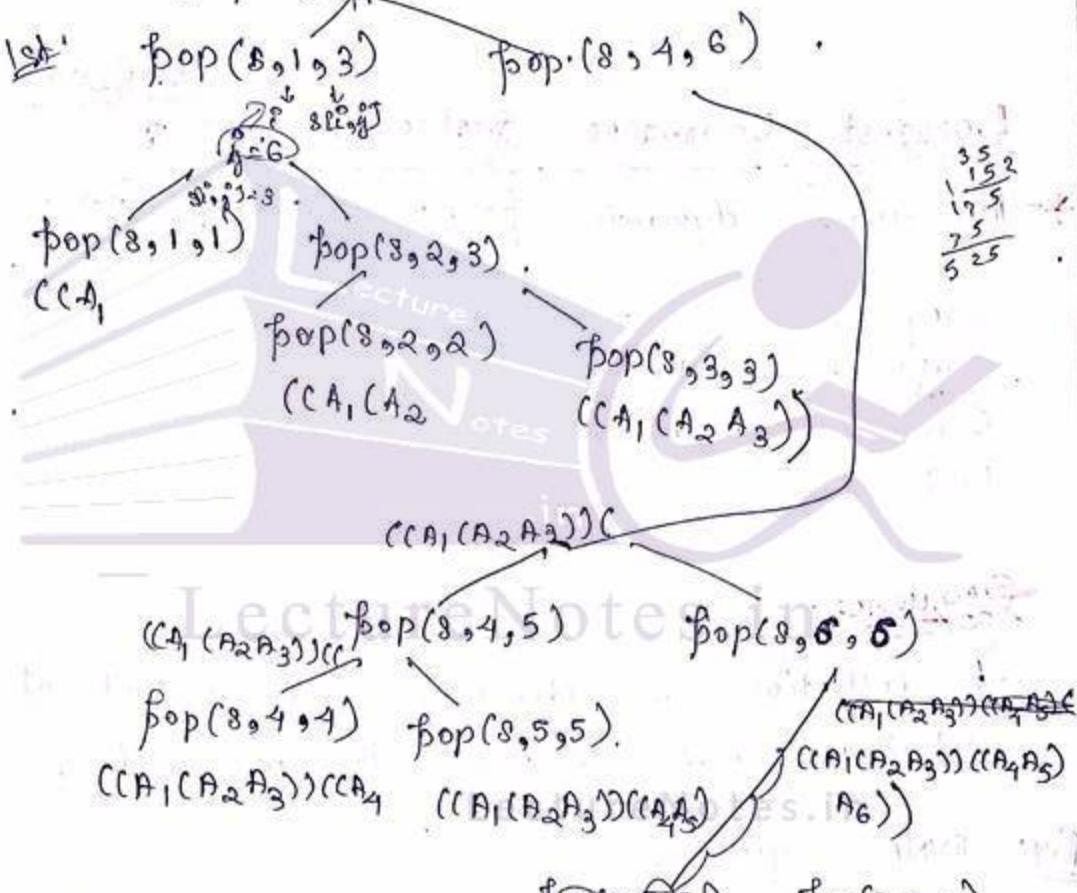
6 Point_optimal_parenthesis($S, S[i, j]+1, j$)

7 Print ')

17889232

* To find the optimal solution i.e., optimal parenthesization of $A_1 \dots A_n$ i.e., the given matrix chain algorithm now $\text{pop}(s, 1, n)$ (pop : find-optimal parenthesis) will be invoked first.

So, $\text{pop}(s, 1, n) \rightarrow \text{input}$.



$$\begin{array}{r}
 35 \\
 152 \\
 125 \\
 75 \\
 375 \\
 525 \\
 \hline
 15075
 \end{array}$$

Verification:

$$\begin{array}{c}
 (A_2 A_3) \\
 (A_1 (A_2 A_3))
 \end{array}$$

$$\begin{array}{c}
 ((A_1 (A_2 A_3)) \\
 ((A_1 (A_2 A_3)) ((A_4 A_5) A_6)))
 \end{array}$$

Optimum cost = $m[1, 6] \approx 15125$

Exercise

15.2.1

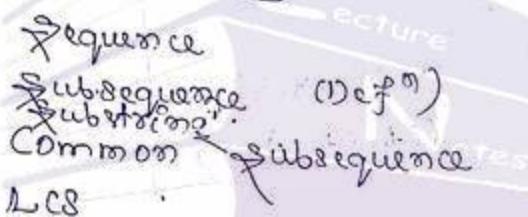
Find the optimal parenthesization of the matrix chain product whose sequence of dimensions is $\{5, 10, 3, 12, 5, 50, 6\}$.

LectureNotes.in

21/9/2015

Longest Common Subsequence (LCS)

*- By using dynamic programming approach.



Sequence

A collection of characters over a set of alphabets and arranged in an order.

Ex: DNA sequence

aatcgacctgggtcc

defined on alphabet set = {a, t, c, g}

Sub-sequence

$s = A \ B \ C \ - \ I \ C \ D \ B$

$s_1 = -IBA$ (Subsequence of s)

$s_2 = AIDB$ (Subsequence of s)

$s_3 = DIBAIB$ \times (order of the characters should be maintained).

Given a sequence

$$M = \langle m_1, m_2, \dots, m_m \rangle$$

another sequence

$$K \leq m$$

$$X = \langle x_1, x_2, \dots, x_K \rangle$$

is called subsequence of M if there exist a strictly increasing sequence $\langle i_1, i_2, \dots, i_K \rangle$ of index of M for $j=1 \dots m$.

*- In a Subsequence of a sequence the characters may not be continuous, but they have to be contiguous.

Ex: $s = aatcgc$

$s_1 = aatc$ (it is a substring & subsequence)

$s_2 = atgc$ (it subsequence)
but not substring.

Common Subsequence

$S_1 = ABBBCDDEA$

$S_2 = ADDBADECA$

$S_1 = ABD$ (Subsequence of S_1 and S_2)
 $S_2 = ABC$ so, common subsequence.

$S_3 = ABDDEA$ (longest common subsequence)
of S_1 and S_2 .

Objective of LCS Problem

To find the LCS between two given strings.

Longest Common Subsequence Problem

Given

$X = \langle x_1, x_2, \dots, x_n \rangle$

and

$Y = \langle y_1, y_2, \dots, y_m \rangle$

The LCS problem tries to find max length common subsequence of X and Y .

Brute force approach

Find all subsequence of X .

Check if it is a subsequence of Y ,

then find the largest among them.

* Given a string of length(m), it can have 2^m subsequences.
Finding LCS of two given strings take exponential time which is impractical for long strings.

Solution:

Solve it by dynamic programming approach.

Prefix:

Given a string

$$X = \langle x_1, x_2, \dots, x_m \rangle,$$

the i th prefix of X is

$$x_p = \langle x_1, x_2, \dots, x_i \rangle.$$

Ex $X = \langle A \ B \ C \ . \ B \ D \ A \ B \rangle$

$$x_4 = \langle A \ B \ C \ B \rangle \quad (\text{4th prefix of } X)$$

$$x_5 = \langle A \ B \ C \ B \ D \rangle$$

x_0 = empty prefix of X .

Theorem 15.1

Let $X = \langle x_1, x_2, \dots, x_m \rangle$

and

$Y = \langle y_1, y_2, \dots, y_n \rangle$

be sequences and let

LectureNotes.in

$Z = \langle z_1, z_2, \dots, z_k \rangle$

be the LCS of X and Y

(i) if $x_m = y_n$,

then $z_k = x_m = y_n$ and z_{k-1} is

~~Lecture Notes~~
 x_{m-1} and y_{n-1} .

the LCS of

(ii) if $x_m \neq y_n$,

then $z_k \neq x_m$ implies Z is LCS of

~~Lecture Notes~~
 x_{m-1} and y_{n-1} .

(iii) if $x_m \neq y_n$,

then $z_k \neq y_n$ implies Z is LCS of X and

~~Lecture Notes~~
 y_{n-1} .

Proof:

~~Ex.~~ $X = ABCBDA$

$Y = BDCAABA$

$Z = BCBBA$

$x_m = A$

$y_n = A$

$z_n = A$, such that A does not contain x

$z_{n-1} = BCB$

$x_m = ABCBD$

$y_m = BDCAIB$

LCS = BCB

$\therefore z_{n-1} \neq \text{LCS}$

binomial theorem;

(ii) Let us proof it by

Let $z_n \neq (x_m = y_m)$.

Then, we could append $(x_m = y_m)$ to z_n to obtain a common subsequence of x and y of length $(n+1)$ which contradicts the supposition that z_n is the longest common subsequence of x and y .

Thus, $x_m = y_m = z_n$

To prove z_{n-1} is the LCS of x_m and y_m when $x_m = y_m$ & $z_n = x_m = y_m$.

Let z_{n-1} is not the LCS of x_m and y_m , and there exist another subsequence ω which is LCS of x_m and y_m and of length greater than $(n-1)$.

$x_m = y_m$ can be appended to last of ω

to obtain LCS of X and Y and of length greater than which is a contradiction.

Z^0, Z_{k-1} is the LCS of X_{m1} and Y_{n1} .

Proof:

(Step) $X = ABCBABCAC$
 $Y = BDCABABAD$

If $x_m \neq y_n$ and $x_k \neq y_m$, then Z is LCS of X_{m1} and Y .

(Step) $X = ABCBABCAC$
 $Y = BDCABABAD$

$Z = X_{m1} \text{ or } Y_{n1} \text{ or } X_{m1} \text{ or } Y_{m1}$

~~Step-1~~
(Optimal characterization of the problem)

Let $X = \langle x_1, \dots, x_m \rangle$

and $Y = \langle y_1, \dots, y_n \rangle$

be two given sequences.

To find the LCS of X and Y ,

if $x_m = y_n$,

then find X_{m1} and Y_{n1} ; then append

($X_m = Y_m$) to last to get LCS of X and Y .

Ex: $X = ABCBDBA$

$Y = BIDCAIBA$

$X_m = Y_m = A$

To find LCS of X and Y .

\Leftarrow , find LCS of $X_{m-1} = ABCBDB$

$Y_{m-1} = BDCAIB$

$Z_{m-1} = BCIB$

To find LCS of X and Y , append $X_m = Y_m$
i.e., \rightarrow at end of Z_{m-1} i.e., $BCIB$.

$\therefore Z = BCIBA$

If $X_m \neq Y_m$,

then find LCS of X_{m-1} and Y .

find LCS of X and Y_{m-1} .

Whichever ever is greater becomes LCS of X and Y .

Ex: $X = ABCBDBACB$

$Y = BIDCAIBAED$

$X = ABCBDBACB$

$Y = BDCAIBAED$

$LCS1 = BCIBAIB$

$X_{m-1} = ABCBDBACB$

$Y = BDCAIBAED$

LCS 2 = BCBBA

LCS 1 > LCS 2

∴, LCS1 is LCS of X and Y.

Step-2 (Define Recursive Solⁿ)

Let $C(i, j)$ is the length of LCS of X_i and Y_j ,

then

$$C(i, j) = \begin{cases} 0 & , i=0 \text{ and } j=0 \\ C(i-1, j-1)+1 & , \text{ if } X_i = Y_j \\ \max(C(i-1, j), C(i, j-1)) & \text{if } X_i \neq Y_j \end{cases}$$

Step-3 (Constructing optimal value).

* The optimal value $C(m, n)$ is obtained by constructing a dynamic programming table of size $(m+1) \times (n+1)$.

* Each value of the table is computed by using the recurrence.

Ex $X = ABCBDBAB$

$Y = BDCAABA$

$$|X| = 7$$

$$|Y| = 6$$

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1 A	0	0	0	0	1	1	1
2 B	0	1	4	1	1	1	2
3 C	0	1↑	1↑	2	2	2	2
4 D	0	1↑	1↑	2↑	2↑	3	3
5 E	0	1↑	2↑	2↑	2↑	3↑	3↑
6 F	0	1↑	2↑	2↑	3↑	3↑	4
7 G	0	1↑	2↑	2↑	3↑	4↑	4

$C(m, n)$ gives the optimal length of LCS.

Step 4: Find optimal soln.

* To get the optimal solution starting from $C(m, n)$ cell and back-tracing to the cell containing the zero value using the arrows.

Take the up arrow (left).

(diagonal arrow)

$BCBA$ is the LCS of both the strings and $BCAB$ also.

Time complexity

Greedy Algorithm | Greedy Approach

* Another way of solving the optimization problem.

* Greedy algorithm make choice of solution which is best at that moment. At ~~the~~ ~~moment~~ they look for the solution which is best locally.



Initially, Greedy chooses the path P_4 . Greedy algorithm never guarantees optimal solution. But, for many problems it has been proved that it gives optimal sol.

In dynamic programming, the sub-problems are dependent non-overlapping sub-problems.

Activity Selection Problem

* Let a set of activities

$$\mathcal{A} = \{a_1, \dots, a_n\}$$

i.e., n number of act. activities available who wants to share a common resource.

* The activity selection problem aims to find maximum ^{size sub-} set of ^{mutually} compatible activities.

* Each activity a_i has a start time (s_i) and finish time f_i i.e., $\{s_i, f_i\}$.
with $0 \leq s_i < f_i$

↓ included
may or
may not
included

* Two activities a_i and a_j are called mutually compatible if $a_i = \{s_i, f_i\}$

$$\text{and } a_j = \{s_j, f_j\}$$

if $s_j \not\gg f_i$ or $s_i \not\gg f_j$.

* Compatible means they don't overlap.

28/9/2015

EDM

Bu: $S = \{a_1, \dots, a_{11}\}$

$n = 11$

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
a_i	4	5	6	7	8	9	10	11	12	13	14

$$S_1 = \{a_1, a_4, a_8, a_{11}\}$$

$$S_2 = \{a_2, a_4, a_8, a_{11}\}$$

$$S_3 = \{a_2, a_1, a_9, a_{11}\}$$

$$S_4 = \{a_2, a_6, a_{11}\}$$

$$S_5 = \{a_3, a_7, a_{11}\}$$

$$S_6 = \{a_3, a_8, a_{11}\}$$

Goal:

$$S_1 = \{a_1, a_4, a_8, a_{11}\}$$

$$S_2 = \{a_2, a_4, a_8, a_{11}\}$$

$$S_3 = \{a_2, a_4, a_9, a_{11}\}$$

any one of the following).

*- First find the optimal substructure of the problem by using dynamic programming approach, then convert it to solve by greedy approach.

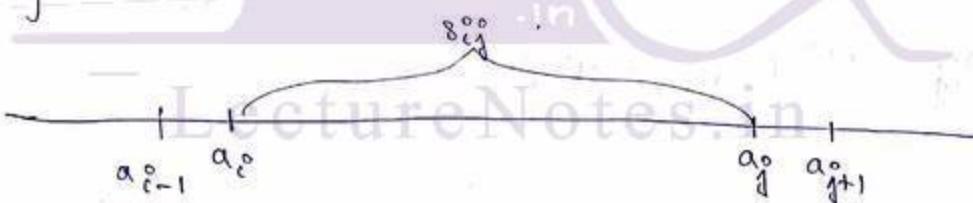
(Defn)

Let S_{ij}^o is the subset of S which is defined as

$$S_{ij}^o = \{a_k : f_i \leq s_k < f_k \leq s_j\}, S_{ij}^o \subseteq S$$

activity of s .

S_{ij}^o contains all jobs which are mutually compatible with a_i^o and a_j^o and all the jobs which are finishing before a_i^o and all the jobs which are starting after a_j^o finishes.



s_k : start time of a_k

f_k : finish time of a_k .

S^o, S can be written as $S_{0, n+1}$ for some fictitious activities a_0 and a_{n+1} such that

$$f_0 \leq s_1 < f_1 \dots f_n \leq s_{n+1}.$$

Let's assume the activities are sorted in monotonically increasing order of their finish time.

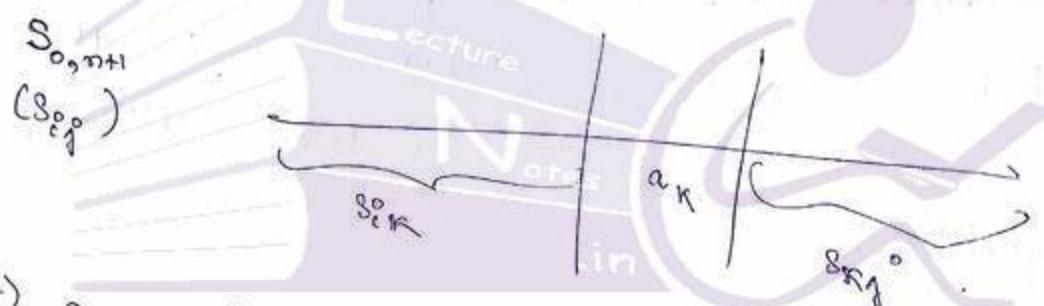
$$f_0 < f_1 < f_2 < \dots < f_{n+1}$$

$$\text{If } i >$$

$$S_{ij}^o = \emptyset$$

$S_{ij}^o \rightarrow$ collection of activities which are mutually compatible with a_i and a_j

*- Find optimal sub-structure



$$\Rightarrow S_{ij}^o = S_{0k}^o (S_{ik}^o) + S_{kj}^o (S_{kj}^o) + a_k$$

Consider some non-empty sub-problem (S_{ij}^o) .

Suppose a solution to S_{ij}^o includes some activity a_k such that

$$f_0 < s_{ik} < f_k < f_j$$

\Rightarrow activity (a_k) generates two sub-problems

(i) S_{ik}^o (activities that start after a_i finishes and finish a_k starts).

29/9/2015

\cap S_{kj} (activities that starts after a_k finishes
and finish before a_j starts).

* Each of S_{ik} and S_{kj} is a consists of a
subset of activities in S_{ij}^o .

\therefore the solution to $S_{ij}^o = (S_{ik}^o \text{ to } S_{ik} + a_k + S_{kj}^o \text{ to } S_{kj})$

* Let A_{ij}^o is the solution of S_{ij}^o , i.e., A_{ij}^o is the
maximum sized subset of S_{ij}^o containing
mutually compatible activities.

$$S_{ij}^o, A_{ij}^o = A_{ik}^o + a_k + A_{kj}^o$$

\downarrow Sol^o of S_{ik}^o \downarrow Sol^o of S_{kj}^o

Where A_{ik}^o is the solution of S_{ik}^o and
 A_{kj}^o is the sol^o of S_{kj}^o .
It will vary from 0 to $(n+1)$.

* Define the sequence

Let $c_{ij}^{o,ij}$ be the number of activities in
a maximum sized subset of mutually
compatible activities in S_{ij}^o .

If $S_{ij}^o \sim \phi$ i.e., $i > j$

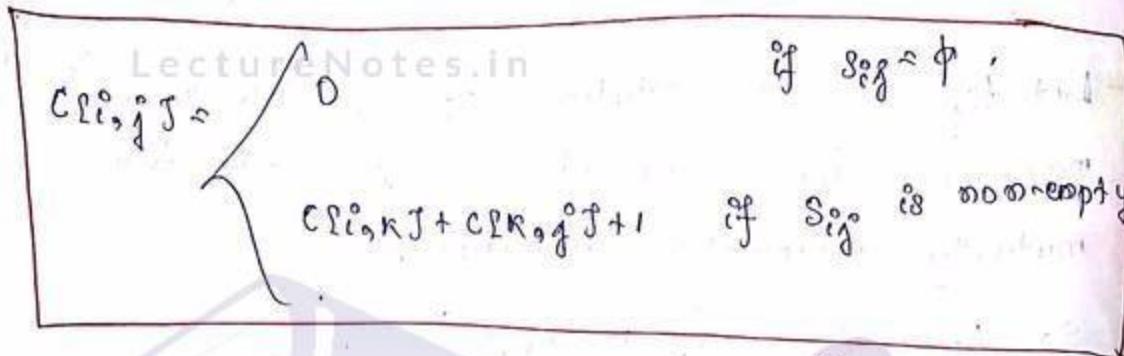
$$c_{ij}^{o,ij} = 0$$

If S_{ij}^o is non-empty,

$$C_{L_i, j}^o = C_{L_i, Kj}^o + C_{LK, j}^{o+1}$$

$C_{L_i, Kj}^o \rightarrow$ no. of activities in L_i

$C_{LK, j}^{o+1} \rightarrow$ no. of activities in L_{Kj} ,



*- Converting the dynamic programming ^{solt} to
Speedy ^{solt}.

Theorem 16.1

Consider any non-empty sub-problem (S_{ij}^o) and let a_m be the activity in S_{ij}^o with the earliest finish time.

If f_m be the finish time of a_m , then

$$f_m = \min \left\{ f_K : a_K \in S_{ij}^o \right\}$$

Then,

- (i) activity (a_m) is used in some minimum sized subset of mutually compatible activities of S_{ij}^o . (a_m must be

included in the solⁿ of S_{ij}^o)
($S_{im} = \emptyset$)
③ the sub-problem (S_{im}^o) is empty so,
choosing a_m leaves the sub-problem S_{mj}^o
is the only free one that may be
non-empty.

($a_m \rightarrow$ 1st activity of A_{ij}^o)

Algorithm:

Recursive-Activity-Selection (s, f, i, j)

1. $m \leftarrow i+1$
2. while $m < j$ and $s_m < f_i$
do $m \leftarrow m+1$
3. if $m < j$
then return $\{a_m\} \cup$ Recursive-Activity-Selection
(s, f, m, j)
4. else return \emptyset .

$s \rightarrow$ array of start time of all the activities,
 $f \rightarrow$ array of finish time of all the activities.

* - For the first call, the algorithm will be called for $i=0, j=m+1$.

Recursive-Activity-Selection ($s, f, 0, m+1$)

i	1	2	3	4	5	6	7	8	9	10	11
S	1	3	0	5	3	5	6	8	8	2	12
f	4	5	6	7	8	9	10	11	12	13	14

* The Greedy approach for solving the activity selection problem chooses the activity (a_m) for solving the subproblem of a greedy activity with earliest finish time.

* The activity picked is of a greedy choice in the sense that it leaves as much opportunity as possible for the remaining activities to be scheduled.

* The greedy choice is the one that maximizes the amount of remaining time.

$$\text{IAS}(S, f, 0, m+1) \quad m = 12$$

i	j	m
0	12	

$$f_0 = 0$$

sol^a set = $\{a_1, a_3, a_5, a_7\}$

RAS(3, 4, 1, 2)

$$\frac{1}{1} \quad \frac{1}{12} \quad \frac{m}{2} \rightarrow 3 \rightarrow 4$$

if ($m < j$)

return a_4

RAS(3, 4, 1, 2)

$$\frac{1}{4} \quad \frac{1}{12} \quad \frac{m}{5} \rightarrow 6 \rightarrow 7 \rightarrow 8$$

RAS(3, 4, 8, 12)

$$\frac{1}{8} \quad \frac{1}{12} \quad \frac{m}{9} \rightarrow 10 \rightarrow 11$$

RAS(3, 4, 11, 12)

$$\frac{1}{11} \quad \frac{1}{12} \quad \frac{m}{12}$$

return \emptyset

Time Complexity

The algorithm needs f in sorted order.

So, if the sorting takes $O(n \log n)$.

Then, time taken by activity selection

problem will be

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

\downarrow
RAS

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

Huffman Codes.

- *- Huffman codes are efficiently and effectively used for data compression. It follows greedy approach for compression of data.

Data Compression → reducing the size of file

Text file



It contains characters

1,00,000 characters,

$$(1,00,000 \times 8) \text{ bits} = 100,000 \text{ bytes}$$

$$= 100 \text{ KB}$$

1 bit → store 2 ch^{es}
codes (0, 1)

2 bits → 4 ch^{es}

3 bits → 8

4 bits → 16

n bits → store ch^{es}



ch ^{es} s	a	b	c	d	e	f
frequency	0.45	13	12	18	9	5

- *- If the characters are represented using binary code, then to represent 6 characters, only 3 bits are enough.

- *- If 3 bit code is used to represent each character code present in file.

Then, the file size is

$$\text{Size } (3,00,000) \text{ bits}$$

37.5 KBS

So, for compressing a file, binary character code is used to represent the characters of a file.

- * The binary character code can be of
 - i) fixed length code
 - ii) variable length code.

Fixed length code means the length of binary code to represent each character is same for all the characters. (uses more memory)

In variable length code, the frequent characters are represented by short length code and infrequent characters are represented by long codewords.

Ex: (Variable length code)

a	b	c	d	e	f
45	13	12	16	9	5

<u>Character</u>	<u>Codeword</u>
a	0
b	101
c	100

d 111

e 1101

f 1100

File size

$$= 45 \times 1 + 13 \times 3 + 12 \times 3 + 9 \times 4 + 5 \times 4 + 16 \times 3$$

$$= 45 + 39 + 36 + 36 + 20 + 48$$

$$= 224,000 \text{ bits}$$
 (as all are taken in thousand
45000, 13000)

Q & A

15
39
36
20
30

Prefix Code

Encoding

Encoding is the process of representing the characters of original file by fixed or variable length binary codeword.

File

abcd

Binary code words

Fixed length
or
Variable length

File is decoded size

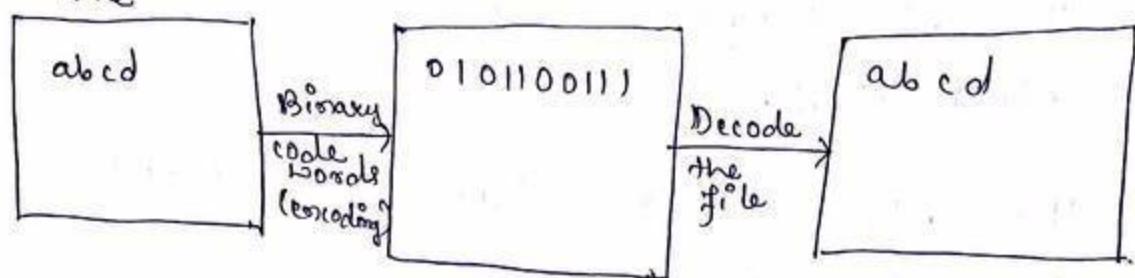
0.101.100.111

= 010110011

(Encoded file)

(. dot is used for concatenation).

To read the file, the receiver has to decode the file.



LectureNotes.in

- * No codeword will be a prefix of another codeword.

a - 0	
b - 101	
c - 100	
d - 111	
e - 1101	
f - 1100	

→ Prefix code.

0 0 10111
↓ ↓ F ↓
a b e d

5/10/2015.

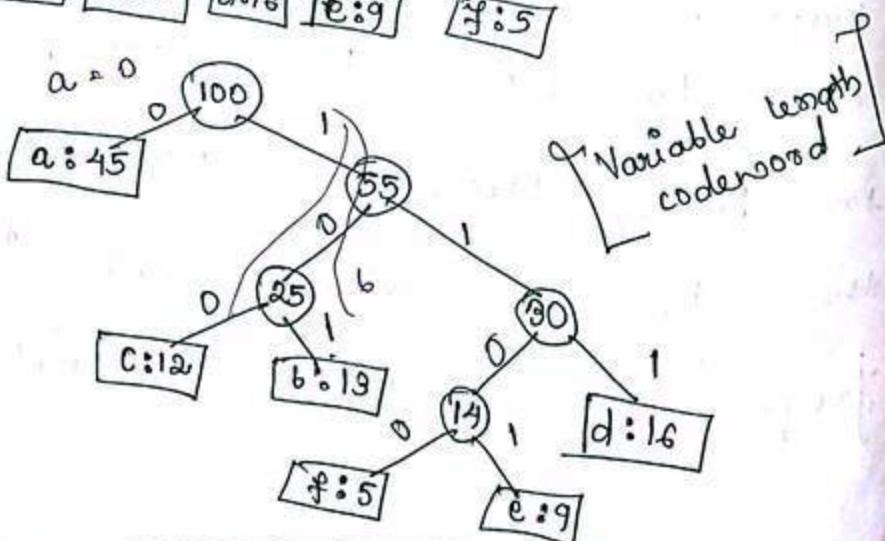
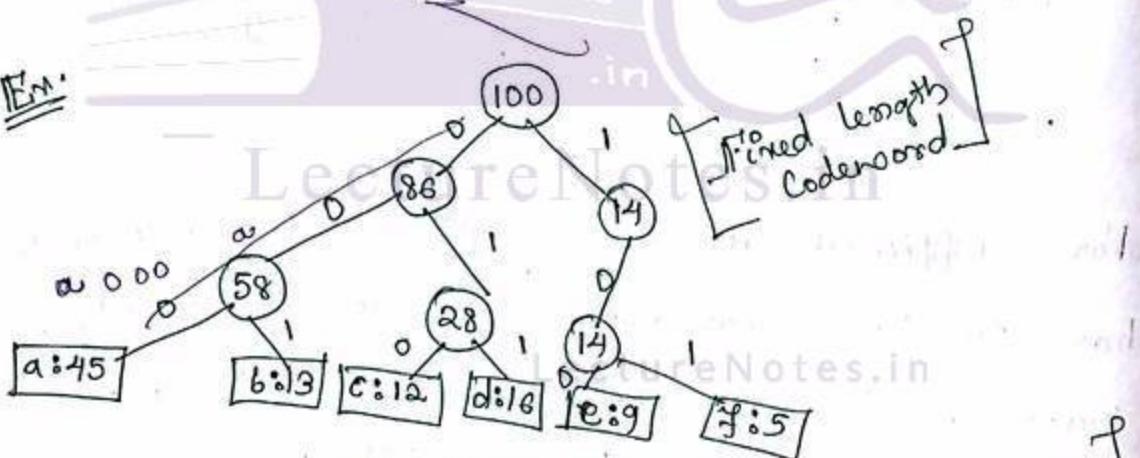
- * For efficient decoding process, the codewords has to be represented in an efficient manner. One of such representations is done by using binary tree.

- * In binary tree, the leaves represents the characters along with their frequency and each internal node

of the tree contains the cumulative or sum of frequency of its left and right child.

- *- The edges of the tree are labelled by using 0 and 1.
0 means $\xrightarrow{0}$ to left child and 1 means $\xrightarrow{1}$ to the right child.

- *- The codeword of a character is the concatenation of the edges of level starting from 01001 to itself.



Full binary tree / binary 2-Tree
every internal node has no child or 2 children

- * An optimal code for a file is always represented by a full binary tree in which every non-leaf node has two children.
- * If "C" is the alphabet from which characters are derived and the characters have positive frequency, then the tree for an optimal prefix code has exactly $|C|$ leaves one for each letter of the alphabet and has exactly $|C|-1$ number of internal nodes.

$C = \{a, b, c, d, e, f\}$

- * Given a tree (T) corresponding to a prefix code, then the number of bits required to encode a file

$$B(T) = \sum_{c \in C} f(c) d_T(c)$$

$f(c) \rightarrow$ frequency of c.

$d_T(c) \rightarrow$ Depth of c from root.

No. of bits required to encode the file.

$$\begin{aligned}
 \text{Ex: } B(T) &= 45*1 + 13*3 + 12*8 + 16*3 + 9*1 \\
 &\quad + 5*4 \\
 &= 2,24,000 \text{ bits}
 \end{aligned}$$

HUFFMAN ALGORITHM

- * maintains a priority queue (Min Priority Queue)
- * HUFFMAN invented a greedy algorithm that constructs an optimal prefix code called HUFFMAN code.
- * Let " C " is a set of alphabets with "n" number of characters. Each character in " C " $c \in C$ has a positive frequency.
- * The algorithm builds the tree (T) in a bottom-up approach. It begins with $|C|$ number of leaves and performs a sequence of $|C| - 1$ number of merging operation to create the final tree.
- * A min priority queue (Q) keyed on frequency (f) is used to identify

the two least frequent objects, to be merged together.

*- The result of merging[?] operation of two objects is a new object where the frequency of the sum of the frequencies of the two objects.

1. $n \leftarrow |C|$
2. $Q \leftarrow C$ (constructs a min heap out of Q)
3. $\text{for } i \leftarrow 1 \text{ to } (n-1)$
4. do allocate a new node (Z)
5. left(Z) $\leftarrow x \leftarrow \text{EXTRACT-MIN}(Q)$
6. right(Z) $\leftarrow y \leftarrow \text{EXTRACT-MIN}(Q)$
7. $f(Z) \leftarrow f(x) + f(y)$
8. $\text{INSERT}(Q, Z)$
9. return $\text{EXTRACT-MIN}(Q)$
10. end

2nd step constructs a minheap which works in $O(n)$ time
from 3 to 8, $(n-1)$ times it applies extract min from the min-heap which takes $O(\log n)$.
So, the total time taken from 3 to 8 is $O(n \log n)$.

9th step will obviously take constant time

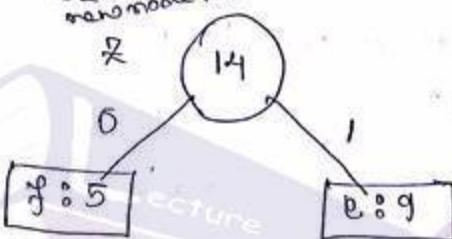
Total time taken by ANUPRAMAN code

$$T(m) = O(m) + O(m \log m)$$

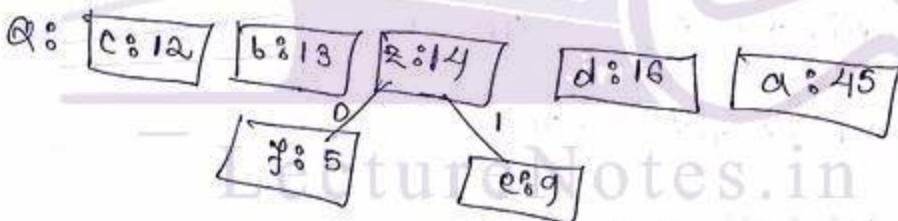
∴ $T(m) = O(m \log m)$



Create a new node (x)

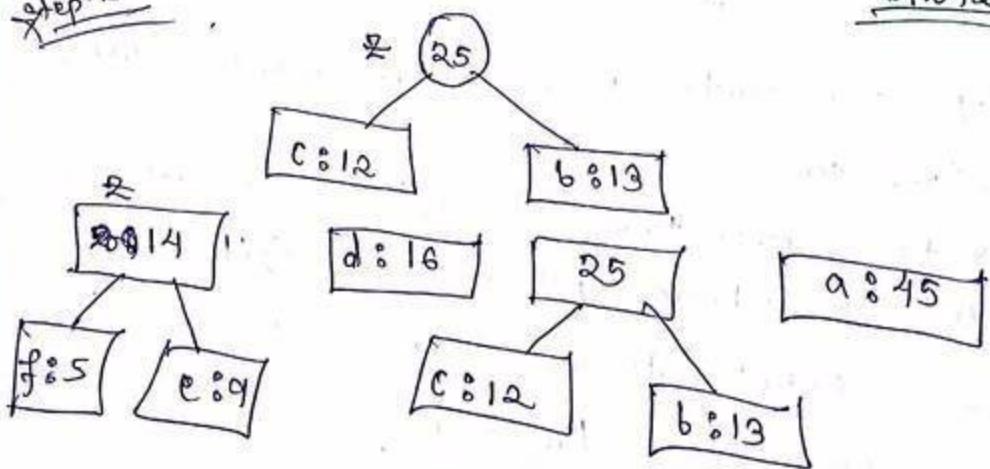


Insert x into priority queue

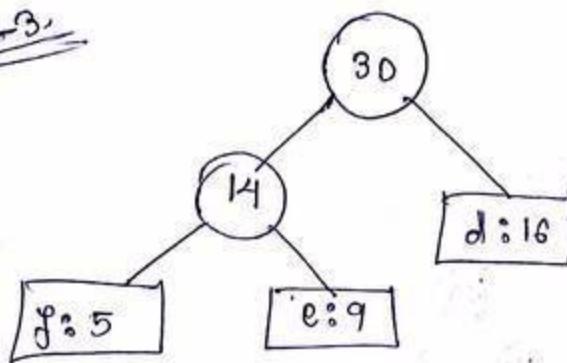


Step-11

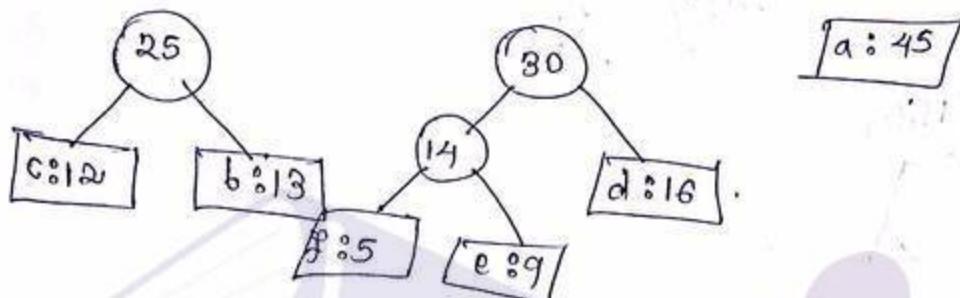
LectureNotes.in 6/10/2015



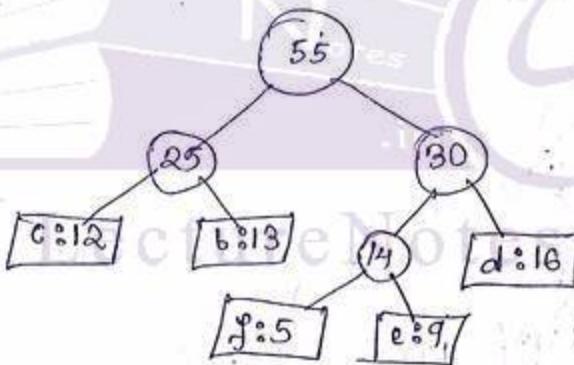
Step-3



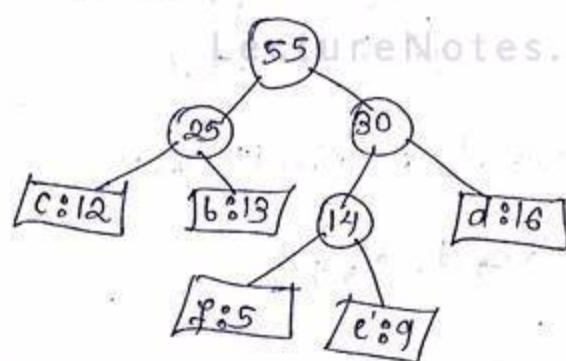
LectureNotes.in



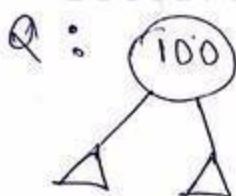
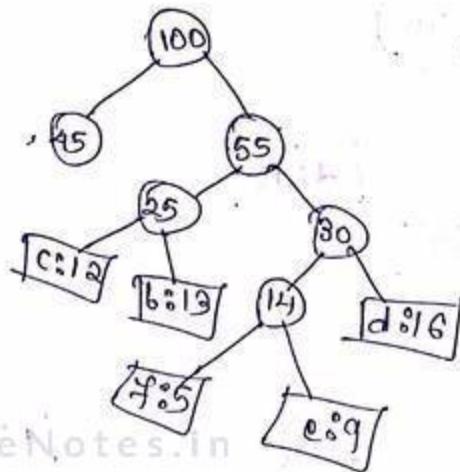
Step-4



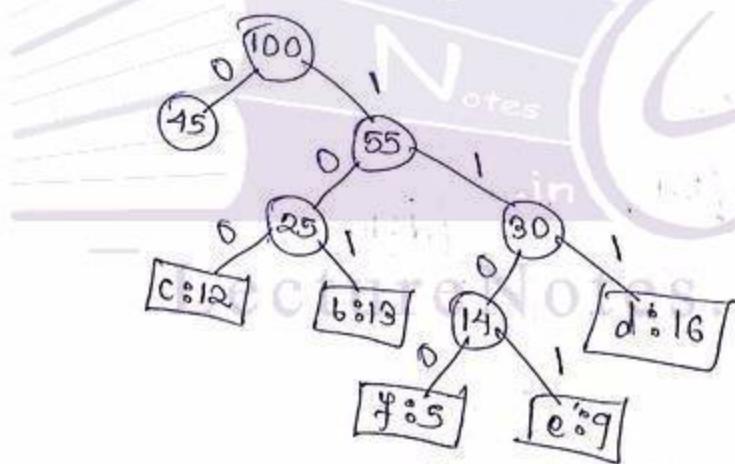
a:45



~~Step 5~~



Huffman Code Tree :



No. of merging required $\Theta(n)$.

$$C = \{a, b, c, d, e, f\}$$

$$|C| \approx 6$$

* Number of leaves in the prefix code representation tree is 6 i.e., n ,

* No. of nonleaf nodes $= 5$ ($n-1$) .

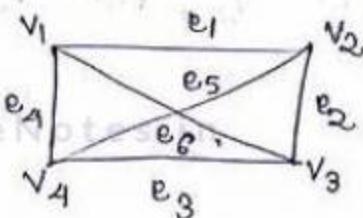
Graph

Algorithms

Graph

$$G = (V, E)$$

Lecture Notes



Graph

$$G = (V, E)$$

is a collection of non-zero set of vertices and edges.

$$V = \{v_1, v_2, v_3, \dots, v_n\}$$

if $|V| < \infty$.

$$E = \{e_1, e_2, e_3, \dots, e_m\}$$

if $|E| < \infty$.

$$e_i = (u, v)$$

In a graph (G) if

$$(v_1, v_2) = (v_2, v_1)$$

that graph is called undirected graph.

If the graph is directed, then

$$e = (v_1, v_2) \neq (v_2, v_1)$$

Adjacent Vertices

(V_i, V_j) , the vertex V_i is adjacent to V_j if there is an ^{directed} edge from V_i to V_j .

V_1 is adjacent to V_2 , but V_2 is not adjacent to V_1 .

LectureNotes.in

Complete Graph

In a complete graph (G), any vertex is adjacent to any other vertex.

Connected Graph

Every vertex is connected to any other vertex of the graph.

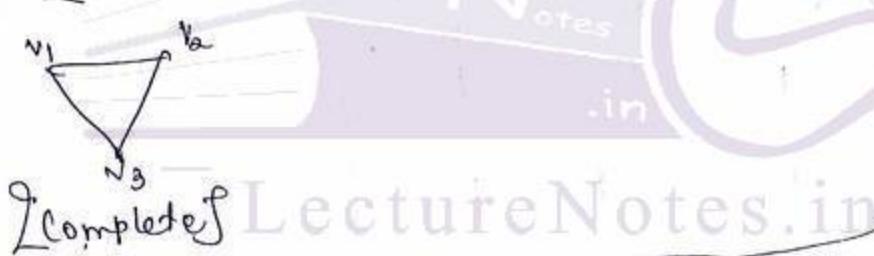
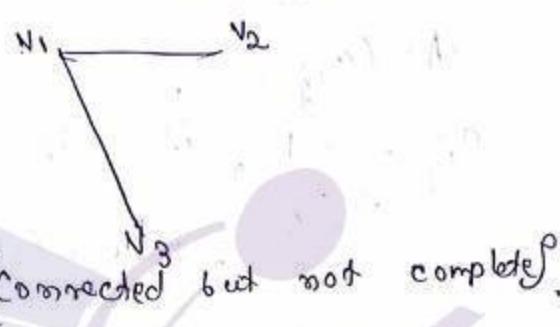
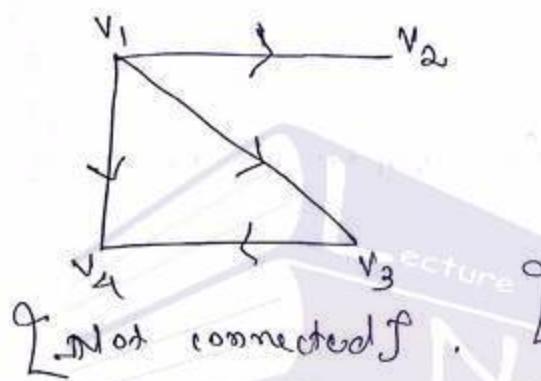
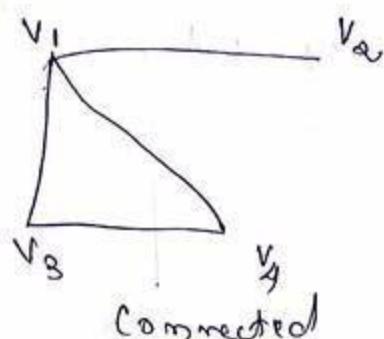
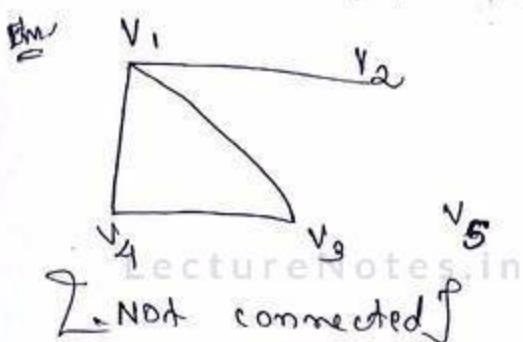
Connected

V_1 is connected to V_m if there exist a path from V_1 to V_m .

Path

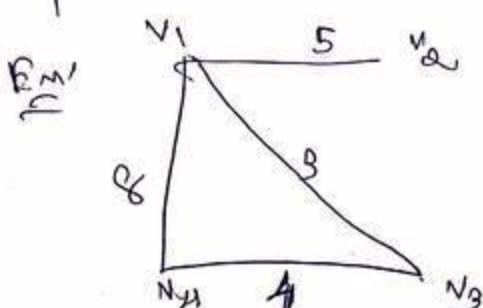
A path from V_1 to V_m is a collection of vertices $\{V_1, V_2, V_3, \dots, V_{m-1}, V_m\}$ where V_1 is adjacent to V_2 , V_2 is adjacent to V_3 , ..., V_i is adjacent to V_j , V_j is adjacent to V_k , ..., V_{m-1} is adjacent to V_m .

* If the path starts and ends at the same vertex, it is a cyclic graph.



If each edge is associated with a non-zero value, then it is a weighted graph.

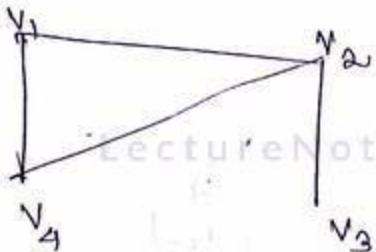
④ Weight can be ~~tre~~ or ~~ve~~.



Representation of a Graph

Adjacency Representation:

Represented by adjacency matrix and adjacency list.

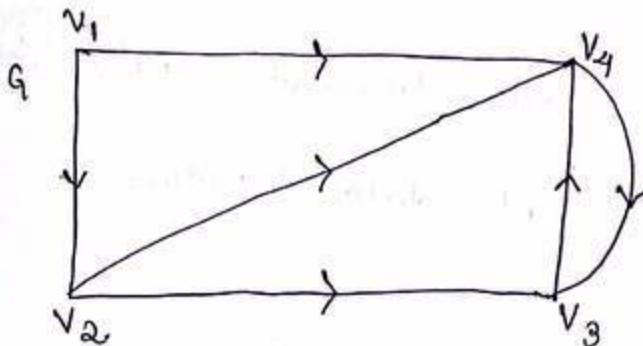


$$A = (a_{ij})$$

$a_{ij} = 1$, if v_i is adjacent to v_j .
 $a_{ij} = 0$, otherwise

	v_1	v_2	v_3	v_4
v_1	0	1	0	1
v_2	1	0	1	0
v_3	0	1	0	0
v_4	1	0	0	0

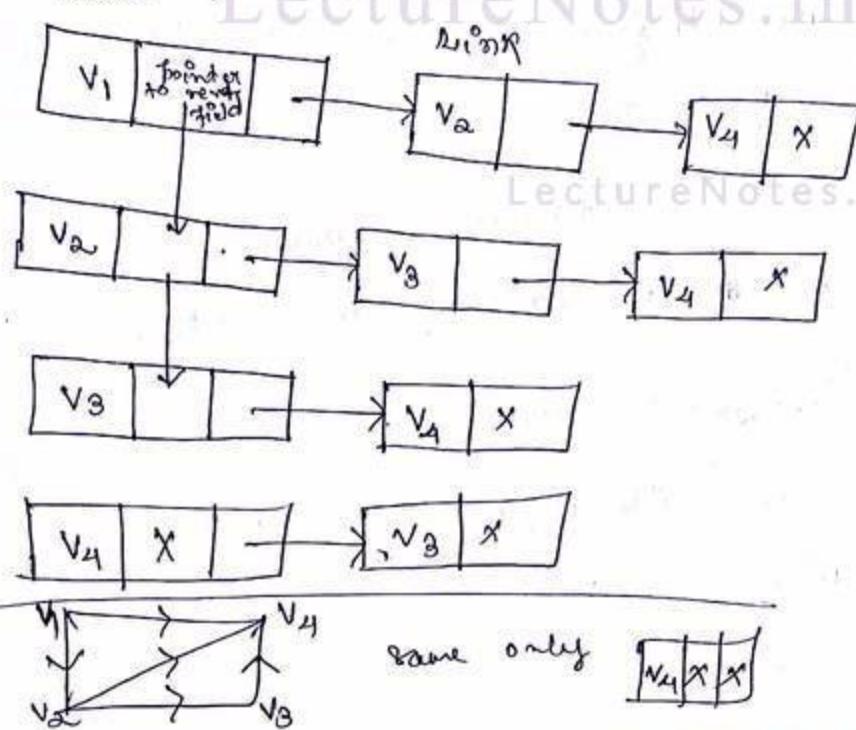
Linked Representation



Nodes	Adjacency Matrix
v ₁	v ₂ v ₄
v ₂	v ₃ v ₄
v ₃	v ₄
v ₄	v ₁ v ₂ v ₃

Mode

LectureNotes.in

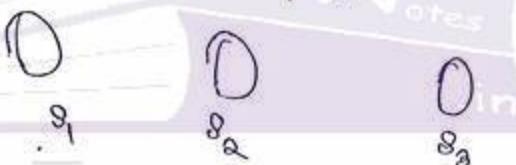


Data Structures for Disjoint Sets

* Grouping n distinct elements into disjoint sets s_1, s_2, \dots, s_n addressed by disjoint set data structure.

* To group n distinct element into disjoint sets normally involves the operation of identifying which object belongs to which set and uniting them.

or elements \rightarrow Graph containing n vertices.



s_1, s_2, s_3 are disjoint \Rightarrow there will be no overlapping elements.

* A disjoint set data structure maintains a collection (set) of disjoint dynamic sets,

$$S = \{s_1, s_2, \dots, s_n\}$$

* Each set is identified by a representative which is a member of same set.

Disjoint set operations

* In dynamic set implementation, each element of a set is represented by an object.

Let M denotes an object then, the following operations are used:

- (i) Make-set(x): This operation creates a new set whose only member is x . (Since the sets are disjoint so, x should not be there in any other set).
- (ii) Find-set(x): This operation returns the representative of the set which contains x .
- (iii) UNION(x, y): Let s_x be a dynamic set containing x and s_y be a dynamic set containing y . Then, this operation unites s_x and s_y ($s_x \cup s_y$) only when their representatives are not same. That means they are two disjoint sets. Then, after uniting these two sets, s_x and s_y will be removed from s .
 $s = \{s_1 \rightarrow s_2 \rightarrow (x, y)\}$.

- * Since sets are disjoint, each union operation reduces the ^{no. of} sets by 1.
- * Then, after $(n-1)$ steps or $(n-1)$ union operations, all the objects will belong to a single set.

Application of Disjoint Sets

1. Find Connected Component of an undirected graph.

Algorithm.

CONNECTED-COMPONENT(G).

= for each vertex $v \in V[G]$,

2: do $\text{findset}(v)$,

3: for each edge $(u,v) \in E[G]$,

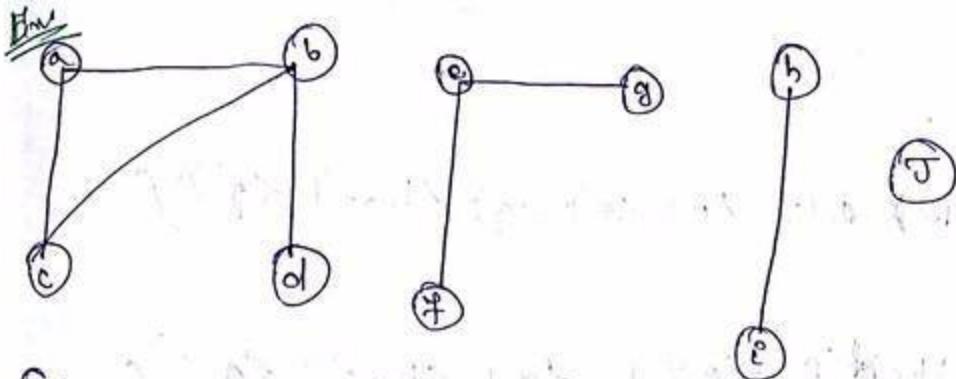
4: do if $\text{findset}(u) \neq \text{findset}(v)$

5: then $\text{union}(u, v)$

6: end.

: $E[G] \rightarrow$ set of edges of G .

$V[G] \rightarrow$ set of vertices of G ,



G: LectureNotes.in

V E G T . R E G T .

~~Step 1~~
S = {a} {b} {c} {d} {e} {f} {g} {h} {i} {j}

(a, b)

rep. (a) = a rep. (b) = b

union {a, b}

S = {~~{a, b}~~} {c} {d} {e} {f} {g} {h} {i} {j}

(a, c)

rep. {a, b} = a or b, rep. {c} = c

union {a, b, c}

S = {~~{a, b, c}~~} {d}, {e}, {f}, {g}, {h} {i} {j}

(b, d), do nothing

(b, d)

S = {a, b, c, d} {e} {f} {g} {h} {i} {j}

Objective of the algorithm :

to find {~~{a, b, c, d}~~} {e} {f} {g} {h} {i} {j}

the disjoint sets.

(~~abc~~)

(h, i)

$\delta = \langle habc \rangle \wedge \langle i \rangle \wedge \langle j \rangle \langle h, i \rangle \langle j \rangle$

(e, f)

$\delta = \langle hacb \rangle \langle e \rangle \langle f \rangle \langle g \rangle \langle h, i \rangle \langle j \rangle$

(e, g)

$\delta = \langle habcd \rangle \langle e \rangle \langle f \rangle \langle g \rangle \langle h, i \rangle \langle j \rangle$

= $\langle j \rangle$ do nothing

8/10/2015

Minimum Spanning Tree (MST)

- *- The design of electronic circuit having equipments needs inter-connection of the pins of the equipments. This problem can be mapped to an undirected graph

$$G = (V, E)$$

Where V is set of pins, and E is the set of inter-connections among the pins.

- *- The objective of such kind of problem is to find an acyclic subset (T)

T \subseteq Ω (set of interconnections)

that connects all of the vertices whose total weight

$$w(T) = \sum_{(u,v) \in T} w(u,v)$$

is minimized.

LectureNotes.in

$w(u,v) \rightarrow$ cost of interconnection of u with v .

(u,v) is an edge and $w(u,v)$ is the cost or weight.

* Since T is acyclic and connects all of the vertices, it is called a spanning tree and the problem of determining $w(T)$ is called minimum spanning tree problem.

Algorithm to find MST.

Generic MST (G, w) .

- 1: $A \leftarrow \emptyset$
- 2: While A does not form a spanning tree .
 - 3: do find edge (u,v) that is safe for A .
 - 4: $A \leftarrow A \cup (u,v)$.
 - 5: return A .

Cut

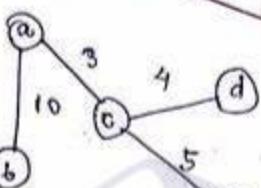
A cut of a graph

$$G = (V, E), (S, V-S)$$

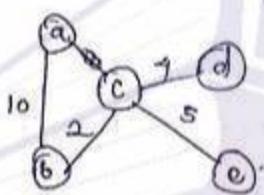
Spanning

Tree.

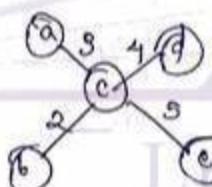
(minⁿ no. of edges which connects all the vertices of a graph).



$$T(c) = 10 + 3 + 4 + 5 \\ = 22$$



$$T(c) = 10 + 1 + 5 + 2 \\ = 21$$



$$T(c) = 3 + 4 + 3 + 2 \\ = 14$$

*- Spanning tree with minimum cost \rightarrow MST

Known as minimum spanning tree

Sum of the weight of edges of a tree is known as cost of the tree.

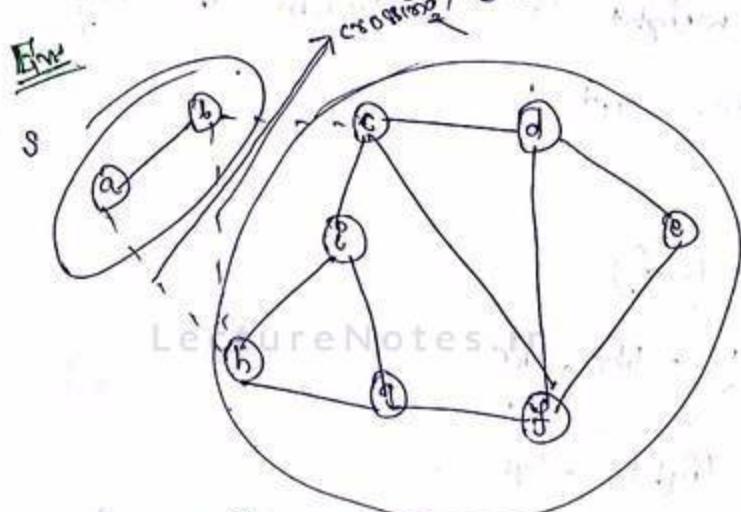
Cut

A cut $(S, V-S)$ of an undirected graph

$$G = (V, E)$$

13/10/2015

is a partition of V .



Crossing Edge: edge which connects two sets S and $V-S$.

($u, v \in E$) crosses the cut $(S, V-S)$

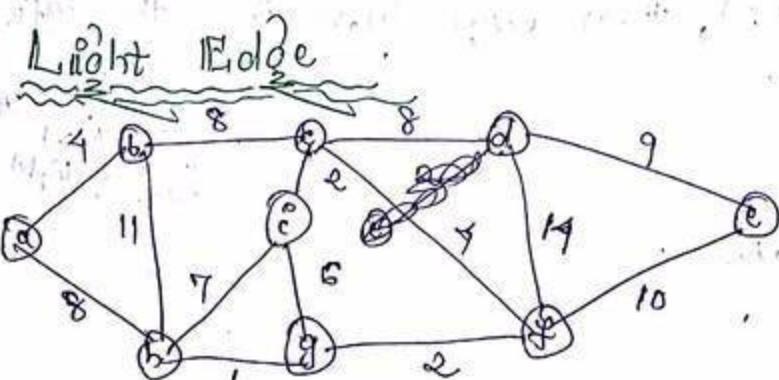
if one of its end point is in S , and another is in $V-S$.

* A cut respects a set G of edges if all the edges of G does not cross the cut.

Ex: $A = \{a, b\}$ respects $S = \{a, b\}$,

$S = \{a, b\}$ respects $A = \{(c, g), (c, f)\}$,

$S = \{a, b\}$ not respects $A = \{(a, b), (a, h), (i, g)\}$,



A edge is a light edge if it violates a cut and its weight is minimum of any edge crossing the cut.

$$S = \{a, b\}$$

$$V-S = \{c, d, e, f, g, h\}$$

Crossing edge = ah, bh, bc

light edge

Kruskal's algorithm (use greedy approach).

Theorem 23.1

Let $G = (V, E)$ be a connected, undirected graph with real valued weight functions (w) defined on E .

Let T be a subset of the tree that is included in some minimum spanning tree of G .

Let $(S, V-S)$ be the cut that respects T and let (u, v) be a light edge crossing $(S, V-S)$, then edge (u, v) is the safe edge for T .

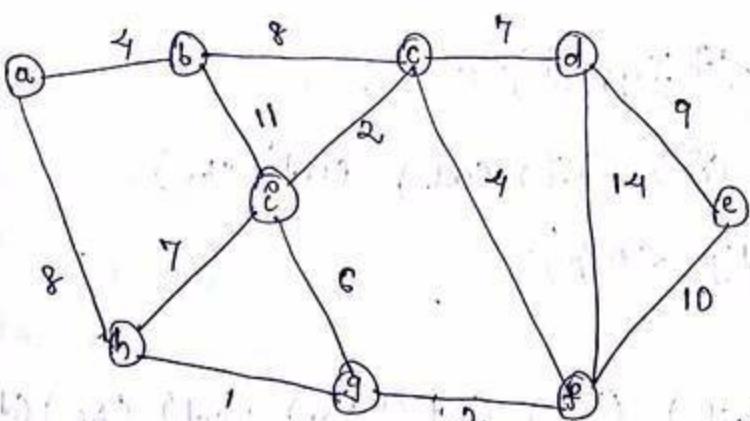
Algorithm

MST-Kruskal (G, w)

$\Leftarrow T \leftarrow \emptyset$

cut respects
area (u, v)
of the light edge

- 2: for each vertex (v) $\in V[G]$
- 3: do make-set(v)
- 4: sort the edges of E in non-decreasing order.
- 5: for each edge $(u, v) \in E$
- 6: do if $\text{Find_set}(u) \neq \text{Find_set}(v)$
- 7: then $A \leftarrow A \cup \{(u, v)\}$
- 8: Union (u, v)
- 9: return A
- 10: end.
- set of edges will be containing the edges included in minimum spanning tree.



Step 1 $\rightarrow \{b\}$

$S = \langle \langle a \rangle \langle b \rangle \langle c \rangle \langle d \rangle \langle e \rangle \langle f \rangle \langle g \rangle \langle h \rangle \langle i \rangle \rangle$

Step 2

$E = \langle \langle hg \rangle, (gf), (ic), (ef), (ab), (ig), (cd), (hi), (bc), (ah), (de), (cf), (bi), (df) \rangle$

$A = \langle (hg) \rangle$ (find_set(h) ≠ find_set(g))

LectureNotes.in

$S = \langle \langle a \rangle \langle b \rangle \langle c \rangle \langle d \rangle \langle e \rangle \langle fg \rangle \langle hg \rangle \langle i \rangle \rangle$

$A = \langle (hg) \rangle, (gf), (ic)$

$S = \langle \langle a \rangle \langle b \rangle \langle c \rangle \langle d \rangle \langle e \rangle \langle fgh \rangle \langle i \rangle \rangle$

$A = \langle (hg) \rangle, (gf), (ic)$

$S = \langle \langle a \rangle \langle b \rangle \langle cd \rangle \langle e \rangle \langle fgh \rangle \langle i \rangle \rangle$

$A = \langle (hg) \rangle, (gf), (ic), (ef)$

$S = \langle \langle a \rangle \langle b \rangle \langle cd \rangle \langle e \rangle \langle icfgh \rangle \rangle$

$A = \langle (hg) \rangle, (gf), (ic), (ef), (ab)$

$S = \langle \langle ab \rangle \langle cd \rangle \langle e \rangle \langle icfgh \rangle \rangle$

cg (not)

$A = \langle (hg) \rangle, (gf), (ic), (ef), (ab), (cd)$

$S = \langle \langle ab \rangle \langle cd \rangle \langle e \rangle \langle icfgh \rangle \rangle$

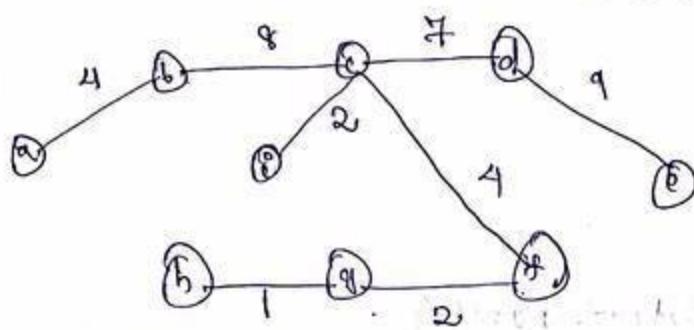
$A = \langle (hg) \rangle, (gf), (ic), (ef), (ab), (cd), (bc)$

$S = \langle \langle abc \rangle \langle fgh \rangle \langle e \rangle \rangle$

ah (not)

$A = \langle (hg) \rangle, (gf), (ic), (ef), (ab), (cd), (bc), (de)$

$\delta = \langle ab\{c\}ghde\rangle^*$



LectureNotes.in
 $T(c) = 4 + 8 + 2 + 7 + 9 + 4 + 2 + 1$
 $= 37$

14/10/2015

Algorithm

Time Complexity of Kruskal's

Line 1 $O(1)$

Line (2~3) $O(N)$

Line 4 $O(E \log E)$

Line (5~8) $O(N)$

$$T(n) = O(N) + O(E \log E) + O(N)$$

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

$$|E| \leq |V| - 1$$

Prim's Algorithm: (use greedy approach)

Algorithm:

MST_Prim's (G, h, π)

for each $u \in V \setminus \pi$,

keyList $\leftarrow \infty$.

3.

$\pi_{\text{Luf}} \leftarrow \text{NIL}$

4. $\text{key}_{\text{Luf}} \leftarrow 0$

5. $Q \leftarrow VEGJ$

6. while $Q \neq \emptyset$

7. do $u \leftarrow \text{Extract-MIN}(Q)$

LectureNotes.in

for each $v \in \text{Adj}_{\text{Luf}}$

8.

do if $v \in Q$ and $w(u, v) < \text{key}_{\text{Luf}}$

9.

10.

then $\pi_{\text{Luf}}[v] \leftarrow u$

11.

$\text{key}_{\text{Luf}} \leftarrow w(u, v)$

12. end.

$G \rightarrow$ Graph

$w \rightarrow$ real valued weight

$\pi \rightarrow$ root of the tree,

$VEGJ \rightarrow$ set of vertices

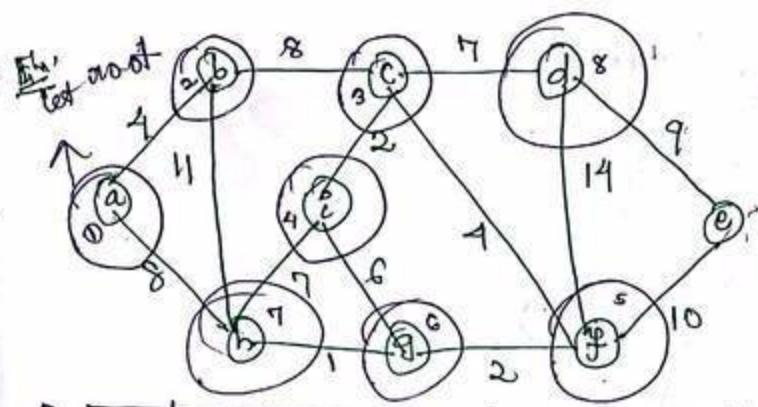
$\text{key}_{\text{Luf}} \rightarrow$ the minⁿ weight associated with the vertex (u)

$\pi_{\text{Luf}} \rightarrow$ parent of the node (u)

$Q \rightarrow$ priority queue (min-priority queue)

$\text{Adj}_{\text{Luf}} \rightarrow$ all adjacent nodes of u .

$w(u, v) \rightarrow$ weight associated with u, v .



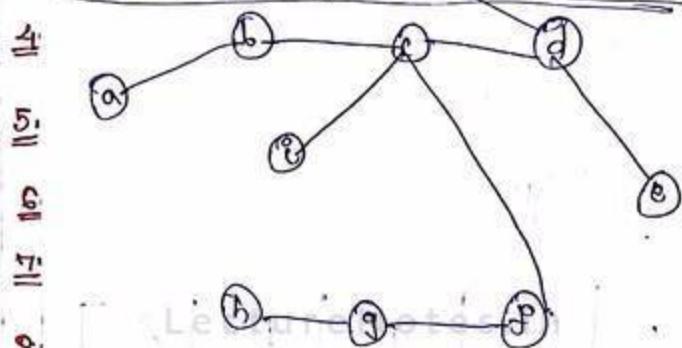
a	b	c	d	e	f	g	h	i
---	---	---	---	---	---	---	---	---

Nodes	Key	π	Adjacent nodes
a	∞ 0 x	N/A	b, h
b	∞ 4 x	N/A	a, c, d, e
c	∞ 8 x	N/A	d, e, f
d	∞ 7 x	N/A	c, f, e
e	∞ 9 x	N/A	d, f
f	∞ 4 x	N/A	g, c, d, e
g	∞ 2 x	N/A	h, i, f
h	∞ 1 x	N/A	a, b, c, g
i	∞ 2 x	N/A	c, g, h

2. Extract the node with minimum key value,
so extract a.

x	x	x	a	f	y	g	k	e
---	---	---	---	---	---	---	---	---

3. Minimum Spanning tree :



9. Time Complexity.

$$\text{Line (1-3)} \approx O(V)$$

~~$$\text{Line - 4} = O(1)$$~~

$$\text{Line - 5} \approx O(V \log V)$$

$$\text{Line - 6} \approx O(V)$$

$$\text{Line - 7} \approx O(V \log V)$$

$$\text{Line (8-12)} \approx O(V-1) \\ = O(E)$$

Total time

$$= O(V) + O(V \log V) + O(V) + O(V \log V) + O(E)$$

$$= O(V \log V)$$

Shortest Path Problem

to find the shortest path between two vertices.

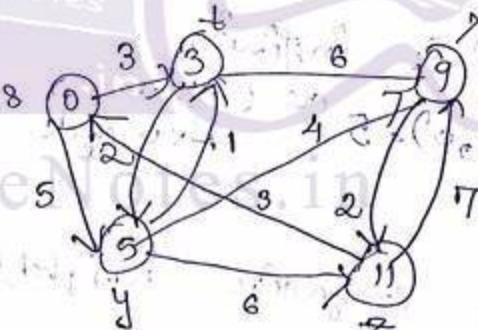
Input: directed graph $G = (V, E)$

Weight function $w: E \rightarrow \mathbb{R}$

$$P = \langle V_0, V_1, \dots, V_k \rangle$$

$$w(P) = \sum_{i=1}^k w(V_{i-1}, V_i)$$

shortest path $(u, v) = \min_P w(P) : u \xrightarrow{P} v$ if there exists a path from u to v
 otherwise
 there might be multiple shortest paths from u to v



single source shortest paths

$G = (V, E) \Rightarrow$ find a shortest path from a given source vertex s to each vertex $v \in V$.

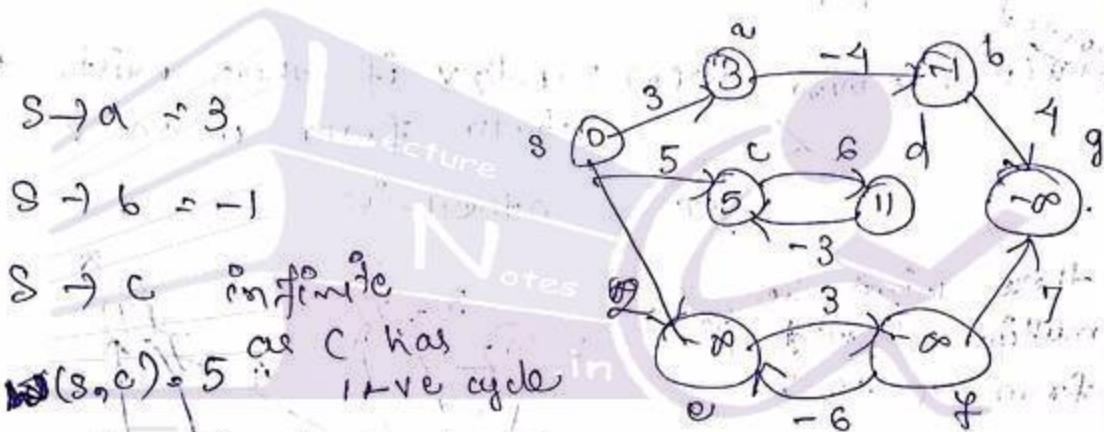
Single dest

given vertex s from each vertex $v \in V$

single pair.
 shortest path from u to v for
 given vertices $u \neq v$, with test mode
 and standard edges. Returns min length of
 a shortest path.

negative weight edges may form negative
 weight cycles.

If such cycles are reachable from source,
 then $d(u, v)$ is not properly defined...



$s \rightarrow e$ (negative weight) $3 + (-6) = -3$

$d(s \rightarrow e) = -\infty$ (no shortest path)

can find paths from s to e with

arbitrarily large negative weights,

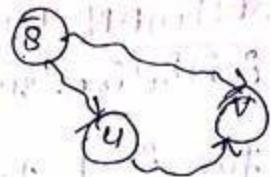
Cycles

Zero-weight

Triangle Inequality

For all $(u, v) \in E$,

$$d(v, u) \leq d(v, u) + d(u, v)$$



$d(v, u)$ = length of shortest path to the vertex v .

$\pi(v)$ = parent of v .
(predecessor)

4/11/2015

$d(u, v) \rightarrow$ length of shortest path,

Initialize-single-source(V, s)

1 for each $v \in V$

do $d(v) \leftarrow \infty$

2 $\pi(v) \leftarrow \text{NIL}$

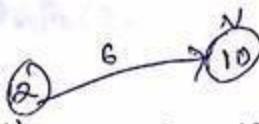
3 $\leftarrow d(s) \leftarrow 0$

Relaxation (u, v) \rightarrow to test whether the shortest path can be improved.

if $d(v) > d(u) + w(u, v)$.

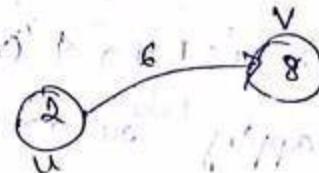
$\Rightarrow d(v) = d(u) + w(u, v)$

$\Rightarrow \pi(v) \leftarrow u$



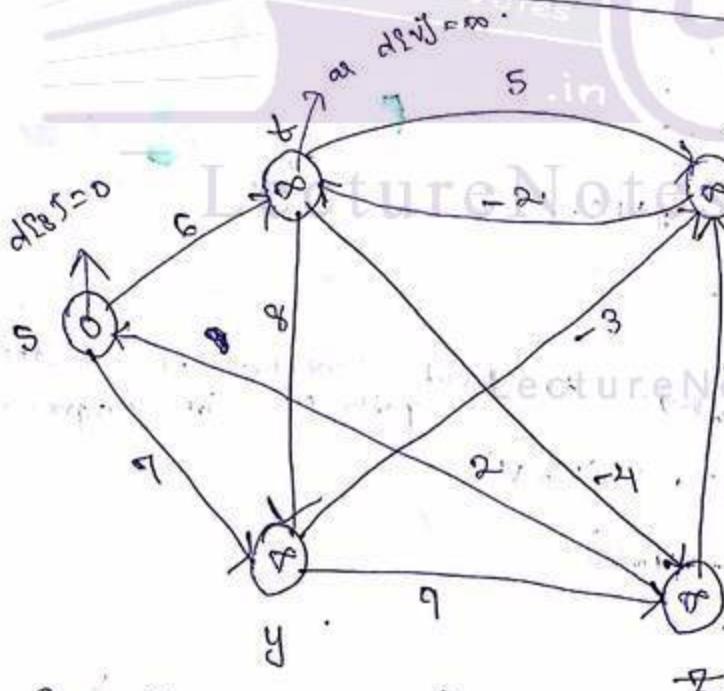
$$d(v) \geq d(u) + w(u, v)$$

$$\Rightarrow 10 > 6 + 2$$

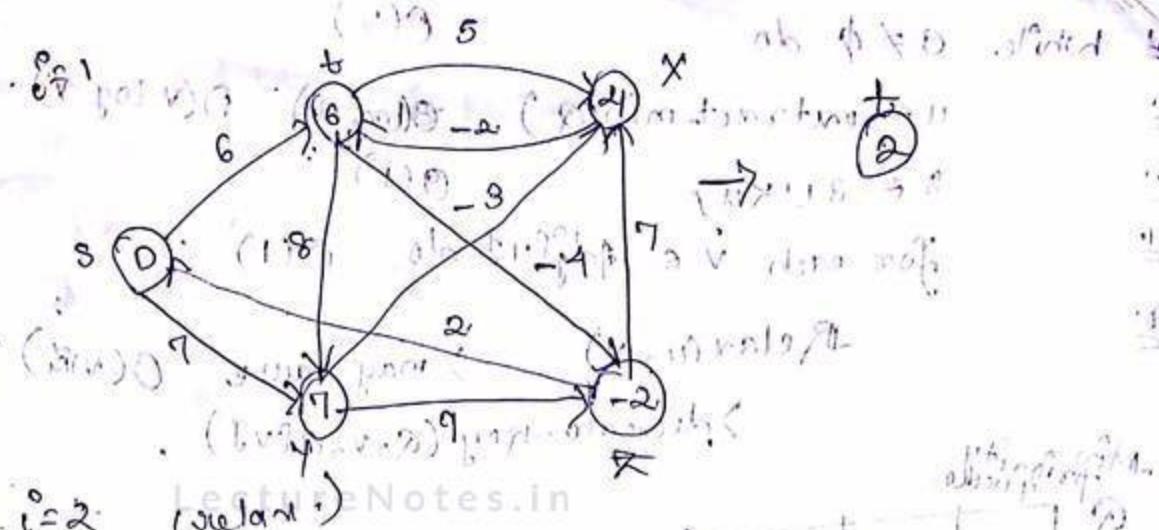


- weights can be -ve
must not contain cycle
- BELLMAN-FORD (V, E, w)
1. initialise-single-source (V, s) $\leftarrow O(V)$
 2. for $i \leftarrow 1$ to $|V|-1$ $\leftarrow O(V)$
 3. do for each $(u, v) \in E$ $\leftarrow O(E)$
 4. do relax(u, v, w)
 5. for each edge $(u, v) \in E$ $\leftarrow O(E)$
 6. do if $d[v] > d[u] + w(u, v)$ $\leftarrow O(E)$ } check -ve cycle.
 7. then return false
 8. return true.

$$T(n) = O(V+E+R) = O(VE)$$



2. for $i \leftarrow 1$ to $|V|-1$
3. apply relax.



$c=3$, $c=4$, no change, last time to $t \rightarrow s$, $\delta(g, t) = -9$.

δ^0 , $\delta(s, t) = 6$, $\delta(s, y) = 9$, $\delta(y, t) = -9$.
It gives the shortest path by maintaining.

10/11/2015

Dijkstra's Algorithm: finds single source shortest path.

non-negative edge weight.

If all edge weights are equal, then we use this algorithm.

Use a priority queue based on d_{VJ} values.
(note: BFS uses FIFO).

Dijkstra(G, s)

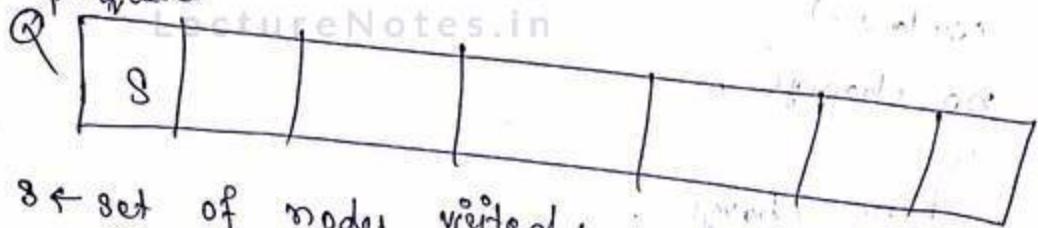
1. Init(G, s) OCV

2. $S \leftarrow \emptyset$ set of discovered nodes. $\Theta(1)$

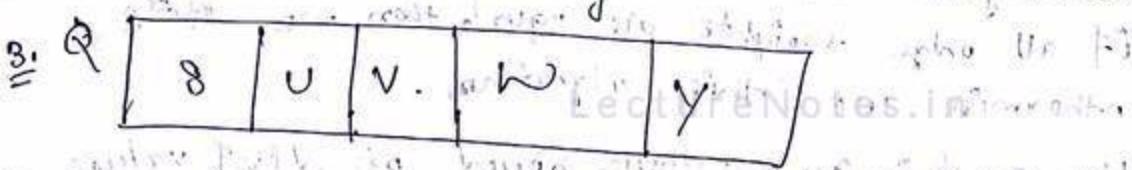
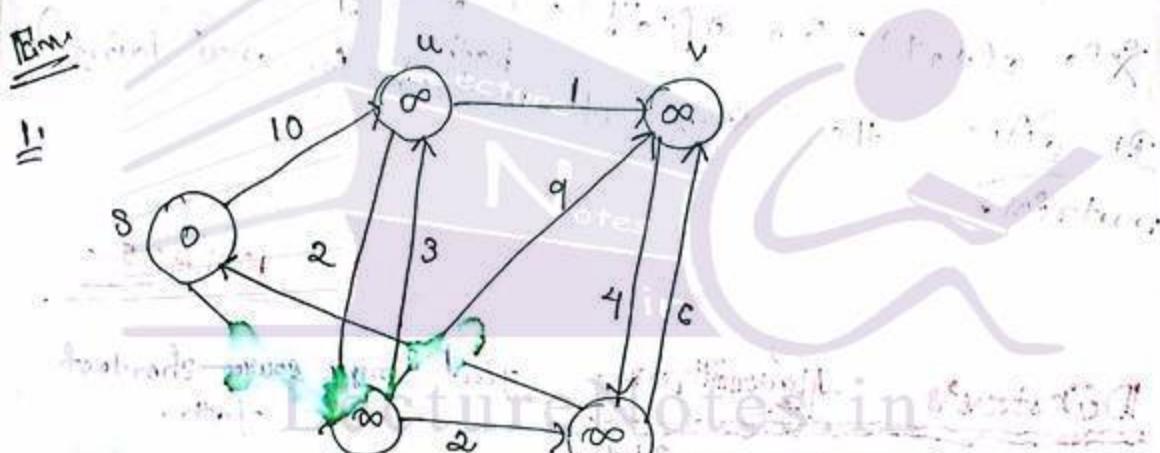
3. $Q \leftarrow V[G]$ $O(\log V)$

2 while $a \neq \phi$ do $O(v)$
 3 $u \leftarrow \text{extract-min}(a)$ $\Theta(\log v) \rightarrow O(v \log v)$.
 4 $S \leftarrow S \cup \{u\}$ $O(1)$.
 5 for each $v \in \text{Adj}_G(u)$ do $O(1)$
 6 $\text{Relax}(u, v)$ \rightarrow may cause $O(vk)$.
 7 $\rightarrow \text{decrease-key}(Q, v, d_{v,u})$.

Min priority queue



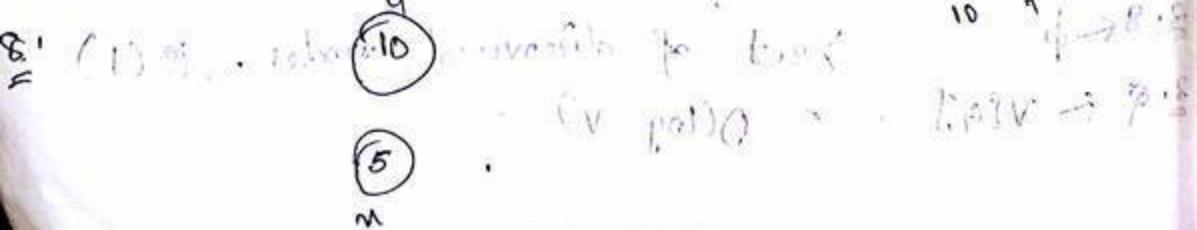
$S \leftarrow$ set of nodes visited.



$S = \{\phi\}$.

$S = \{s\}$.

$\text{Adj} = u, v, w$.



Q	m	u	v	n	y	16	7	2
---	---	---	---	---	---	----	---	---

$$S = \{s, n\}$$

$$\text{adj}[\Sigma] = u, v, y$$

$$\text{relax}(u) = 8$$

$$v = 14$$

$$y = 7$$

LectureNotes.in

(8)



(14)

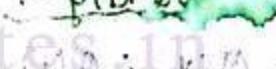


(7)

(y)



(w)



Q

Q	y	u	v	n	w	in
---	---	---	---	---	---	----

$$S = \{s, n, y\}$$

$$\text{adj}[\Sigma] = s, n$$

$$\text{relax}(n) = 18$$

Q

Q	u	v	.
---	---	---	---

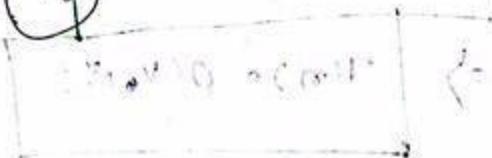
$$S = \{s, n, y, u\}$$

$$\text{adj}[\Sigma] = n, v$$

$$\text{relax}(n) = 5$$

$$v = 9$$

(19)



(19)

(5)



$S = \{s, u, y, u, v\}$

$$\text{Adj } LvS = y$$

$\text{relax}(y) = \pi$

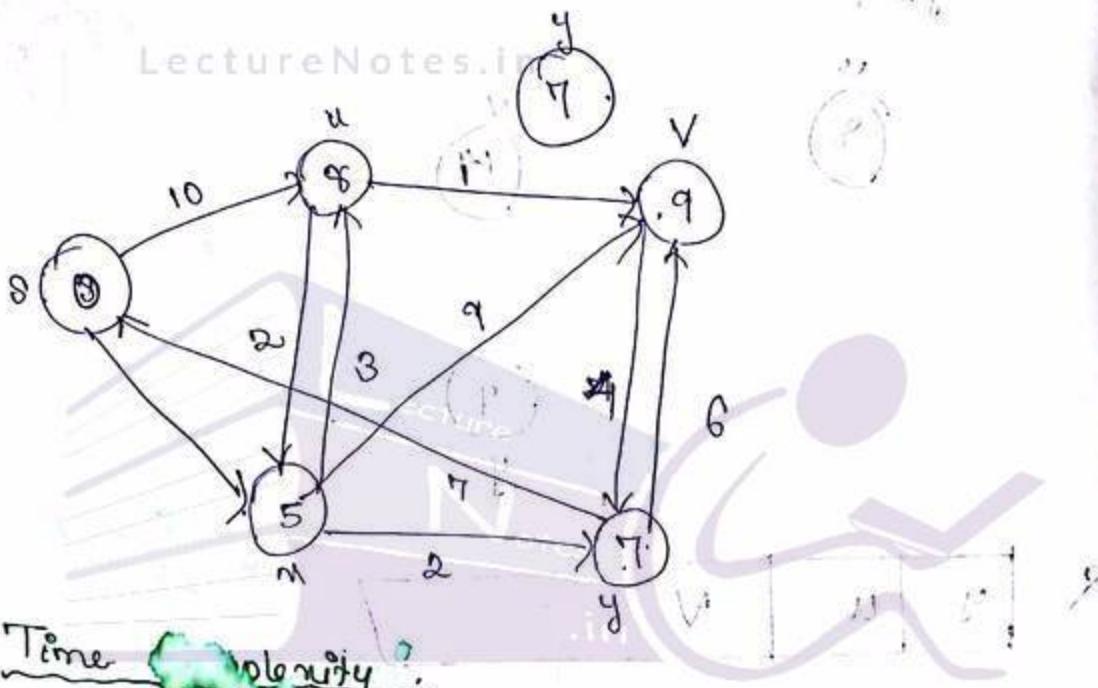
$\pi \in R \geq 0$

parallel -> parallel

parallel relaxation

$\pi_1 = \pi$

$\pi_2 = \pi$



$$T(n) = O(v) + O(1) + O(\log v) + O(v) + O(v \log v) \\ + O(1) + O(1) + O(v \cdot E)$$

$$T(n) = O(v \cdot E)$$

$$\Rightarrow T(n) \approx O(v \cdot E)$$

Flow Graph.

Flow is the state that material moves through the network.

Source vertex (s) → from where material is produced.

Sink vertex (t) → where material is consumed.

Goal: determine max¹ state of material flow from source to sink.

For all other vertices: what goes in must go out.

Formal Max Flow Problem

Graph $G = (V, E)$ → a flow network

- * directed, each edge has capacity $c(u, v) \geq 0$
- * two special vertices $s \neq t$.
- * for any other vertex (v) , there is a path $s \rightarrow \dots \rightarrow v \rightarrow \dots \rightarrow t$.

Flow → a fun² $f: V \times V \rightarrow \mathbb{R}$.

- * capacity constraint: for all $u, v \in V$: $f(u, v) \leq c(u, v)$
- * flow symmetry: for all $u, v \in V$: $f(u, v) = -f(v, u)$
- * flow conservation: for all $u \in V - \{s, t\}$:

$$\sum_{v \in V} f(u, v) - f(v, u) = 0$$

or, $\sum_{v \in V} f(v, u) - f(u, v) = 0$

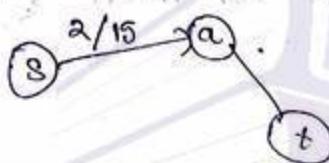
$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v) \quad (\text{flow conservation})$$

Another approach

Cancellation of flows -
 Avoid the flows in opposite dirⁿ b/w the same pair of vertices.
 Such flows cancel each other due to open symmetry.

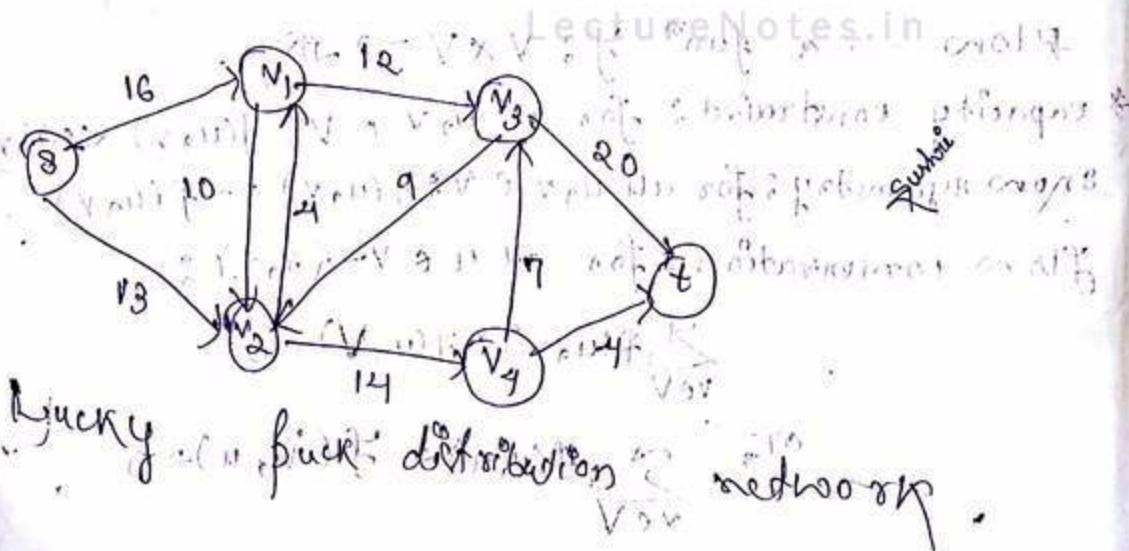
$$f(s, a) = 2$$

$$f(a, s) = 2$$



Main Flow

Find a flow of maxⁿ value from the source to the sink. Denoted by $|f|$.



Ford-Fulkerson Method

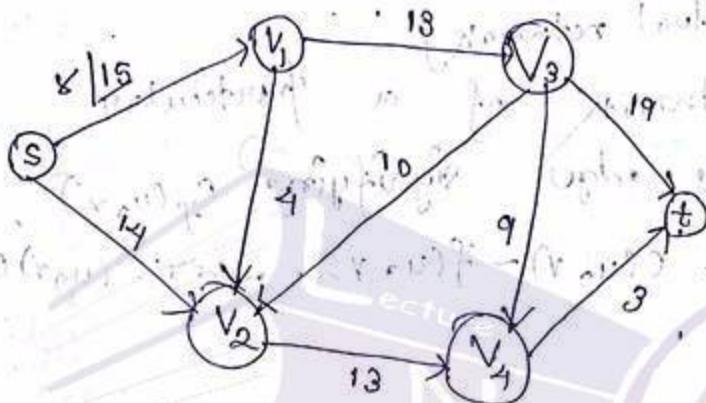
13/11/2015

Contains several net algorithms:

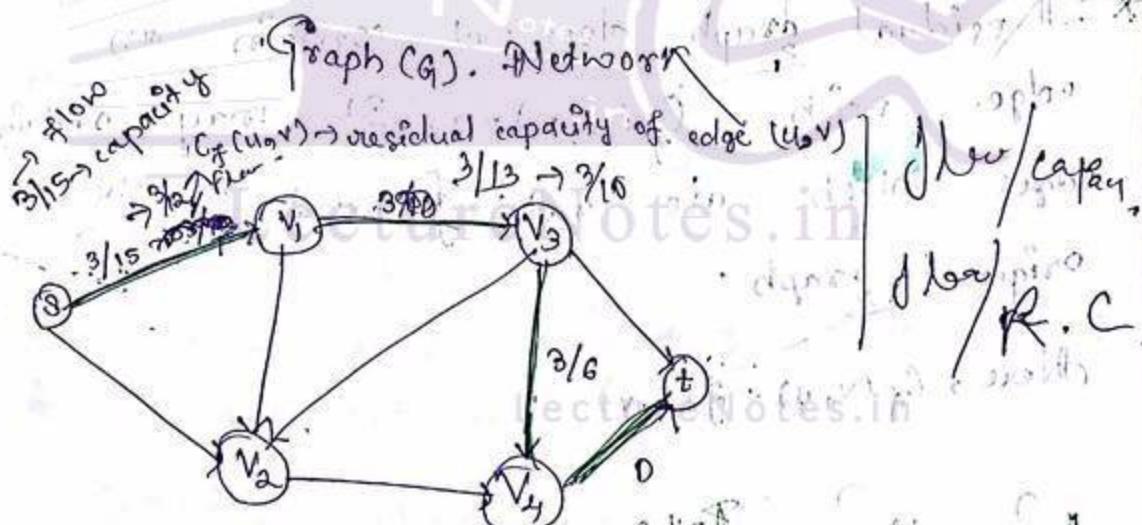
- residue networks
- augmenting paths
- symmetry

Residual Networks

LectureNotes.in



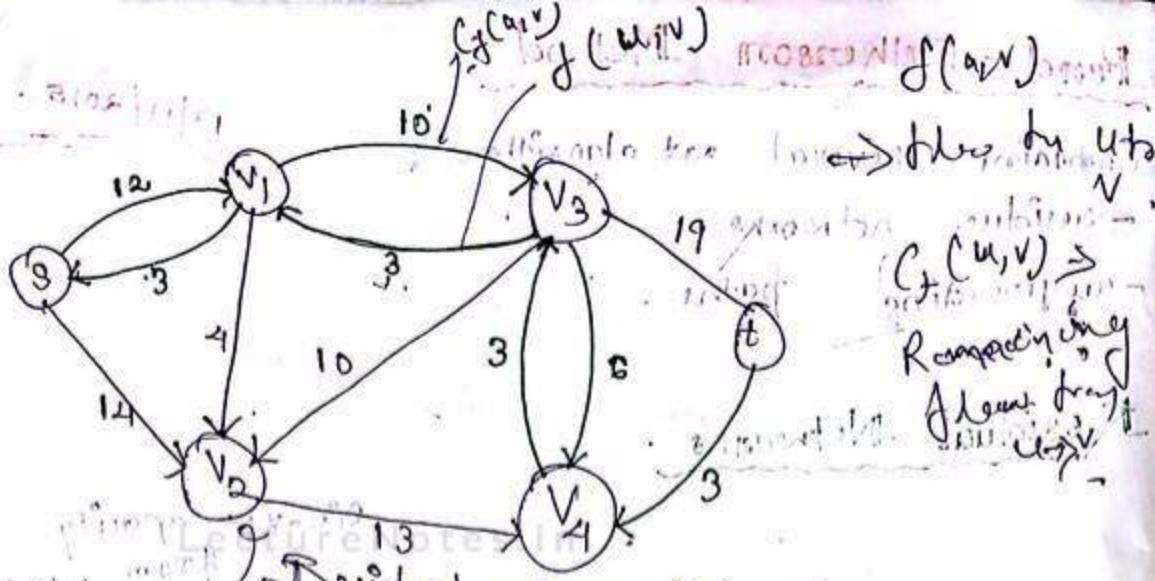
$c(u, v)$ = capacity from u to v



flow/capacity
flow/R.C.

$$C_f(u, v) = C(u, v) - f(u, v)$$

enqueued with on hold at first teeth
on off for with a number of col



* Residual network of a particular graph contains edges signifying $c_p(u, v)$, i.e., $c_p(u, v) = c(u, v) - f(u, v)$, where $(u, v) \in E$.

* Residual graph does not contain an edge with $c_p(u, v) < 0$. It may contain edges which are not present in the original graph.

$$\text{Now, } c_p(v, u) = f(u, v)$$

Augmenting Paths

The augmenting path of a residual graph decides the amount of flow that will be added to the previous flow to increase the net flow.

* The Ford-Fulkerson algorithm terminates when the residual network carries no path from source to destination by returning minimum flow.

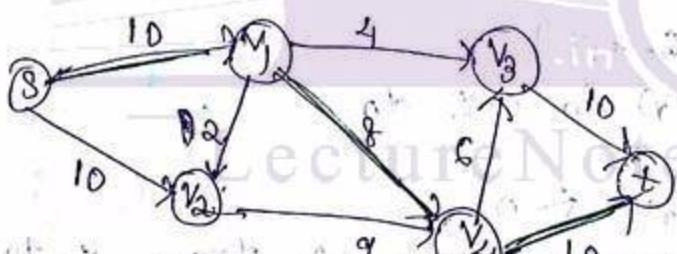
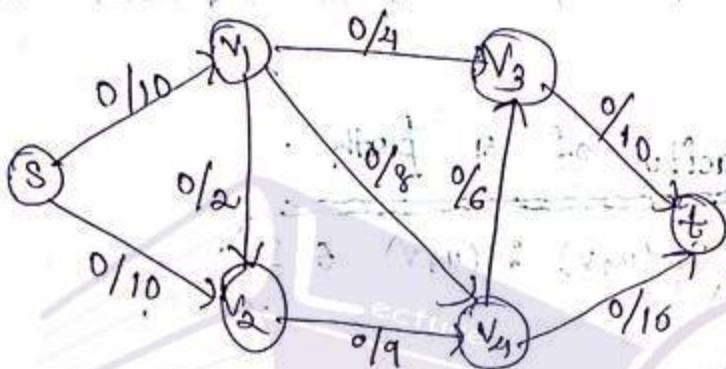
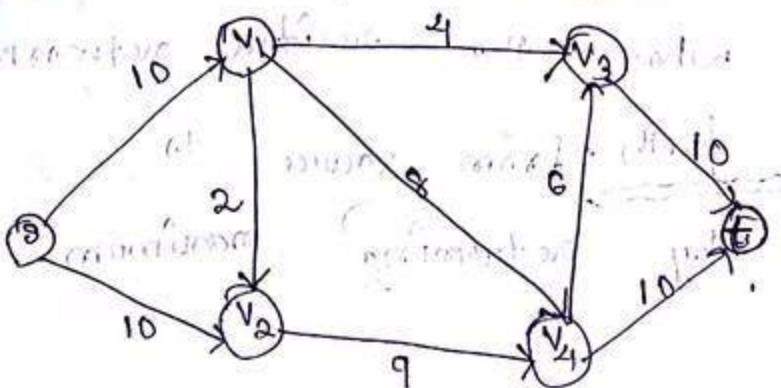
* Find a path P from s to t , such that there is

Residual Capacity of a path

$$C_f(P) = \min_{\downarrow \text{residual capacity of path}} \{C_f(u, v) : (u, v) \in P\}$$

Ford-Fulkerson (G, s, t)

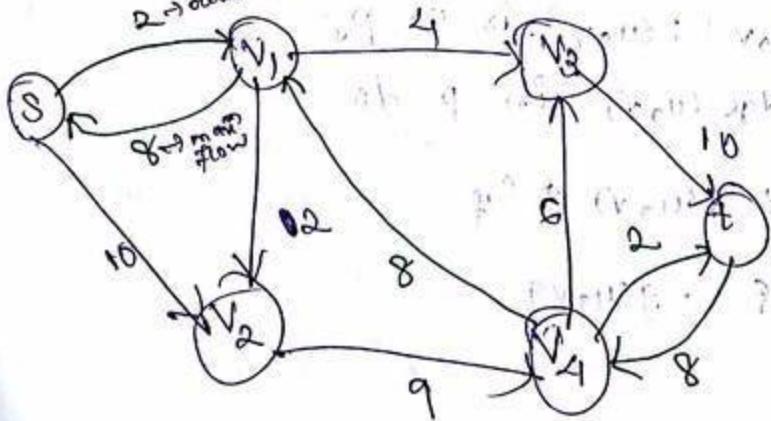
- 1 for each edge (u, v) in G do
- 2 $f(u, v) \leftarrow f(v, u) \leftarrow 0$
- 3 while there exists a path P from s to t in residual network (G_f) do
- 4 $C_f = \min \{C_f(u, v) : (u, v) \text{ is in } P\}$
- 5 for each edge (u, v) in P do
- 6 $f(u, v) \leftarrow f(u, v) + C_f$
- 7 $f(v, u) \leftarrow -f(u, v)$
- 8 return f .



maximum amount of data from source to destination
= 8
residual capacity of the path = 8.

$$C_f(p) = 8$$

\Rightarrow residual capacity.



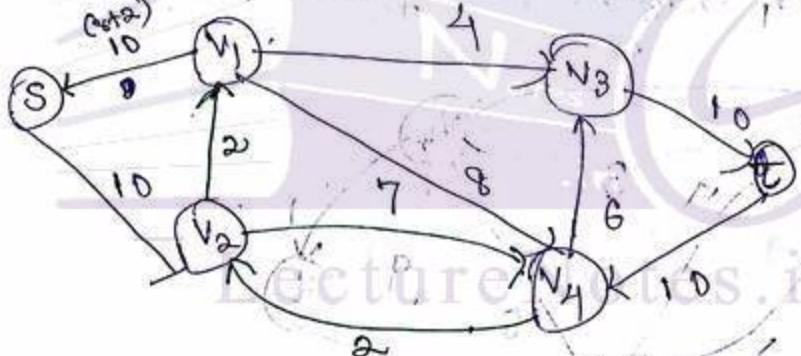
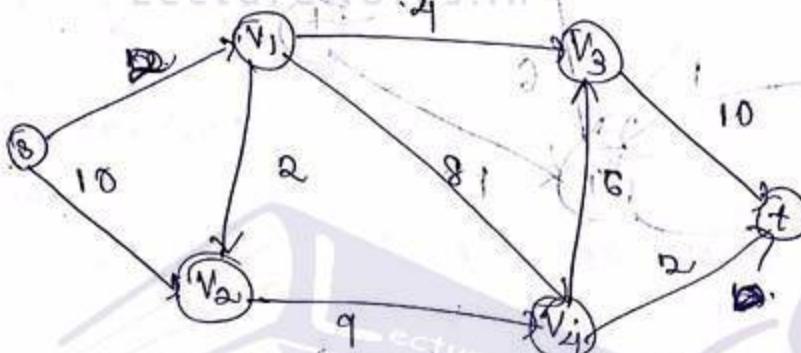
Consider path. (1) $S \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow t$

$$c_f(p) = 2$$

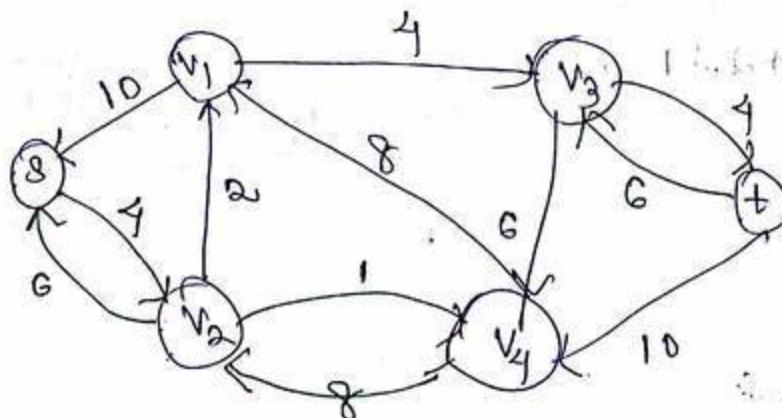
Flow of the network

$|f| = \sum$ residual flow of each path.

$$|f| = 0 + 8 + 2$$



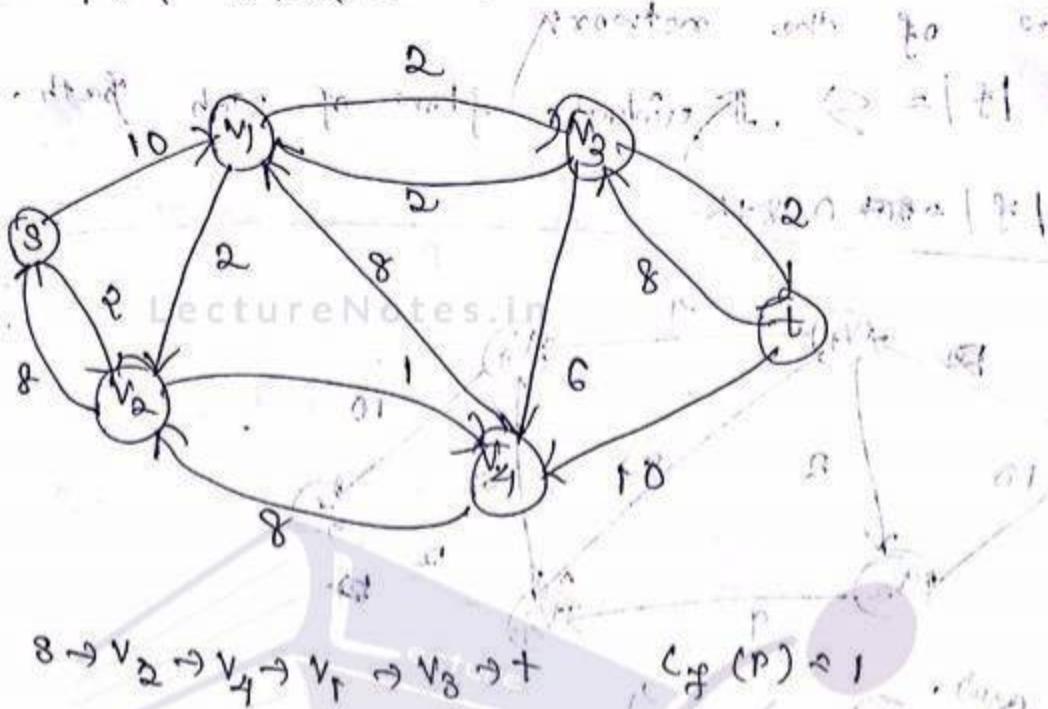
P_3 $S \rightarrow v_2 \rightarrow v_1 \rightarrow v_3 \rightarrow t$ $c_f(p) = 6$



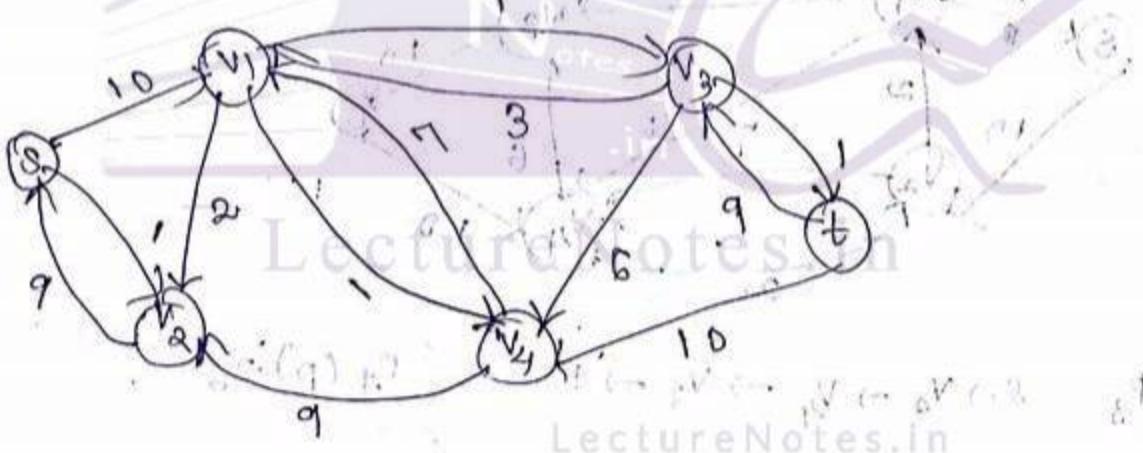
10
8
1
6
4
9
p

$$s \rightarrow v_2 \rightarrow v_1 \rightarrow v_3 \rightarrow t \quad c_f(p) \approx 5^+$$

$$|f| = 8+2+6+2$$

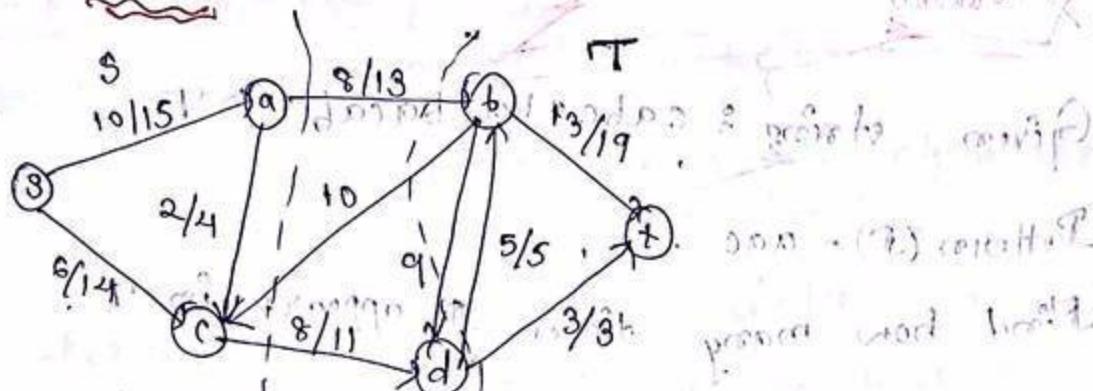


$$s \rightarrow v_2 \rightarrow v_4 \rightarrow v_1 \rightarrow v_3 \rightarrow t \quad c_f(p) \approx 1$$



$$|f| = 8+2+6+2+1 \\ = 19$$

Cut.



$$S = \{s, a, c\}$$

$$T = \{b, d, t\}$$

net flow of cuts sum of flows from $a \rightarrow b$ and $c \rightarrow d$.

$$= (13 + 11) - 10 = 14$$

$$= (13 + 11) - 10 = 14$$

↓

sum of flows from $s \rightarrow T$ sum of flows from $T \rightarrow s$

Capacity of the cut

$$= 2, \text{ccu, v} \quad u \in S \quad v \in T$$

(57)

Theorem: Max flow min cut

Corollary: $\max_{\text{flow}} \leq \min_{\text{cut}}$

$\max_{\text{flow}} \geq \min_{\text{cut}}$

String Matching

Given string : cabcaba, pattern = T

Pattern (P) = aac

Find how many times P appears in T.

Ex: Nucleotides : A + C G T . (DNA sequence)
P = $\langle \text{aacgg} \rangle$, (motif)

Let t_0 = left three ch^{gs}

Match do with P.

If not, then shift by 1

$t_1 = \langle \text{abc} \rangle$

Match with P,
upto (m-m+1). /T/ and /P/ or m.

Naive String matching (T, P)

1. $n \leftarrow \text{length}(T)$

2. $m \leftarrow \text{length}(P)$

3. for $s \leftarrow 0$ to $(n-m)$ with step 1

4. if $P[1:m] == T[s+1:s+m]$

ff ("s is a valid shift");

$T(n) = O((n-m)*m)$

$= O(n^2) = O(n*m)$

Rabin-Karp → Algorithm

Σ is set of alphabets

$$d = |\Sigma|$$

for English alphabets

$$d = |\Sigma|^{0.26} \approx 9$$

assume $\Sigma = \{0, 1, \dots, 9\}$ so that each ch is a decimal digit.

$$\text{Ex: } P = 31415 \text{ (pattern)}$$

It's corresponding decimal formula

Norvig's formula

$$P = P[m] + 10(P[m-1]) + 10(P[m-2]) + \dots + 10(P[1]) + 10(P[0])$$

$$= 5 + 10 + 40 + 10 + 30 = 95$$

$$= 5 + 10(1 + 10(4 + 10(1 + 10(3))))$$

$$= 5 + 10(1 + 10(4 + 10(81)))$$

$$= 5 + 10(1 + 10(314))$$

$$= 5 + 10(314)$$

$$= 5 + 31410$$

$$= 31415$$

$T \rightarrow$ string

$$t_0 = T[m] + 10(T[m-1]) + 10(T[m-2]) + \dots + 10(T[1]) + 10(T[0])$$

$$t_8 \approx 10(t_8 - 10^{m-1}T[m-1]) + T[m+1]$$

((the window is one character))

modulo q

$$P \rightarrow \dots, t_0, m \quad \text{finding } T \{ s + mt + j \} \bmod q$$

$$t_{s+1} = (d(t_s - T \{ s + 1 \} h) + T \{ s + mt + 1 \}) \bmod q$$

q is a prime no. chosen by user.

$$h = d^{m-1} \pmod{q}$$

Rabin-Karp matcher (T, P, d, q)

1. $m \leftarrow \text{length } \{ T \}$.
Getting $d^{\text{length } T} \equiv P \pmod{q}$.
2. $m \leftarrow \text{length } \{ P \}$.
Initial comparison of P .
3. $b \leftarrow d^{m-1} \pmod{q}$.
Initial shift of window.
4. $\# \leftarrow 0$.
Number of matches.
5. $t_0 \leftarrow 0$.
($t_0 \equiv \text{char}_0 \text{ of } T \pmod{q}$).
6. for $i \leftarrow 1$ to m (Preprocessing).
 $t_i \leftarrow (dt_{i-1} + T[i]) \bmod q$.
7. $t_0 \leftarrow (dt_0 + T[0]) \bmod q$.
8. for $s \leftarrow 0$ to $m-m$ (Matching).
if $p = t_s$
if $P[1:m] = T[s+1:s+m]$
if pattern occurs with shift s .
10. if $s < (m-m)$
 $t_{s+1} \leftarrow (d(t_s - T[s+1:h]) + T[s+m+1]) \bmod q$
11. $t_0 \leftarrow (t_0 + d^{m-1} \cdot (T[0] - T[s+1:h])) \bmod q$
12. $T(n) = O(n) + O(m(n+m)/q)$

$$\text{Ex: } P = 31415 \quad \text{Vinegar} \rightarrow \text{H}_2\text{O}$$

$$P_0 = 0 \quad \text{Initial amount}$$

$$t_0 = 0 \quad \text{Initial time}$$

$$d = 10 \quad \text{Dissolution rate}$$

$$q = 11 \quad \text{Rate of evaporation}$$

$$\frac{P}{P_0} \leftarrow 3 \cdot 0 \cdot 11 \approx 3 \quad P = \boxed{31415}, t = 1000 \text{ sec}$$

$$\frac{t_0}{t_0} \leftarrow 2 \cdot 0 \cdot 11 \approx 2 \quad t_0 = \boxed{2350}, \text{ min}$$

$$\frac{C}{C_0} \leftarrow 30 \cdot 1 \approx 31 \quad \text{Final concentration}$$

$$t_0 \leftarrow 20 + 3 = 23 \quad \text{Time}$$

$$\frac{P}{P_0} \leftarrow 31 + 4 \approx 35 \quad P = \boxed{31415}, t = 2000 \text{ sec}$$

$$q \leftarrow 23 + 5 \approx 28 \quad t_0 = \boxed{2350}, \text{ min}$$

$$\frac{P}{P_0} \leftarrow 3140 + 1 \approx 3141 \quad P = \boxed{31415}, t = 2000 \text{ sec}$$

$$q \leftarrow 2350 + 9 = 2359 \quad t_0 = \boxed{2350}, \text{ min}$$

$$\frac{P}{P_0} \leftarrow 3141 \times 10 + 5 = 31415 \quad P = \boxed{31415}, t = 2000 \text{ sec}$$

$$q \leftarrow 23590 + 0 = 23590 \quad t_0 = \boxed{2350}, \text{ min}$$

If $n \rightarrow$ polynomial

⇒ Degree bound $\approx \Theta(n^3)$.

Degree \rightarrow highest power of the polynomial.

$$\text{Ex: } P(x) = 3x^3 + 2x^2 + 1$$

∴ Degree = 3.

Degree bound ≈ 4 .

$\Theta(n^2)$ → Evaluation of polynomial.

Eff

$\Theta(n \log n)$.

To evaluate value of a polynomial.

$$P = \underline{\underline{173}}, \quad T = \underline{\underline{31592918}}$$

$$t_0 = 3^{15}; \quad t_1 = 159. \quad P = \underline{\underline{173}}$$

$$m = 3$$

$$s = 2$$

$$P = P[3] + 10(P[2] + 10(P[1] + 10P[0]))$$

$$0 \text{ to } n-m$$

$$= 173$$

$$n-m \times m$$

$\Theta(m)$