Ch-1 : Fundamentals of Algorithm.

* Algorithm - step by step solution
  - set of rules to solve Problem
  - Finite Instruction

- Characteristics - Clear
  - ~~I/O or O~~
  - Input / Output
  - Finiteness
  - Effectiveness
  - Feasible
  - Definiteness.

Ch-2 : Analysis of Algorithm

* Types of Analysis of Algorithm

1 Wort Case - Worst Condition, Maximum input, maximum time.

2 Best Case - Best Condition, Best time, No Possible in real time.

3 Average Case - Average Condition, Average running time, real time possible.

* Asymptotic Analysis

1 O notation : $F(n) \leq c\,g(n)$

2 Ω notation : $F(n) \geq c\,g(n)$

3 Θ notation : $F(n) = c\,g(n)$

* Insertion Sort :

```
For j ← 2 to n                          time * Cost
    do key ← A[j]                       c1 n
    i ← j - 1                           c2 (n-1)
    while i > 0, A[i] > key             c3 (n-1)
                                        Σ_{j=2}^{n} tj      c4
     do A[i+1] ← A[i]                   Σ_{j=2}^{n} tj-1    c5
     i ← i - 1                          "                   c6
    A[i+1] ← key                        c7 (n-1)
```

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} t_j - 1 + c_6 \sum_{j=2}^{n} t_j - 1 + c_7(n-1)$$

- Best Case : Take $t_j = 1$ ($\Theta(n)$ or $O(n)$)
- Worst Case : Take $t_j = j$ ($\Theta(n^2)$ or $O(n^2)$)

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

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

* Bubble Sort :

	time + cost
for i ← 1 to n	n $c_1(n+1)$
do for j ← length[A] to i + 1	$\sum_{i=1}^{n} n+1-i$ c_2
if A[i] > A[j]	$\sum_{i=1}^{n} n-i$ c_3
A[i] ↔ A[j]	$\sum_{i=1}^{n} n-i$ c_4

$$T(n) = c_1(n+1) + (c_4+c_3)\left(\sum_{i=1}^{n} n-i\right) + c_2 \sum_{i=1}^{n} n+1-i$$

- Best case : take i = 1 ($\Omega(n)$ or $\Theta(n)$)
- Worst case : take i = n ($\Theta(n^2)$ or $O(n^2)$)

* Selection Sort :

	cost + time
For i ← 1 to n-1 do	c_1 n
min j ← 1, ~~to~~ max ← a[i]	$c_2(n-1)$
For j ← i+1 to n do	$c_3 \sum_{i=1}^{n} n-i+1$
if a[j] < max	$c_4 \sum_{i=1}^{n} n-i$

min j ← j	$C_5 \sum_{i=1}^{n} n-i$
max ← a[i]	$C_6 \sum_{i=1}^{n} n-i$
a[min j] ← a[i]	C_7 n-1
a[i] ← max	C_8 n-1

-) Best Case : Take i = 1 ($\Omega(n^2)$ or $\Theta(n^2)$)

- Worst Case : Take i = n ($\Theta(n^2)$ or $O(n^2)$)

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

Ch-3 : Divide and Conquer Algorithm.

* Divide - Divide the Problem into sub-Problem
Conquer - Find solution of sub-Problem
Combine - Combine solution of sub-Problem

* Substitution Method :

Ex. $T(n) = T(n-1) + 1$ then

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

Put the substitu of n into the n-1 and Find one Pattern with K value

$\therefore T(n) = T(n-2) + 2$
$\therefore T(n) = T(n-3) + 3$
$\therefore T(n) = T(n-k) + k$

Find the value of K and put into the $T(n)$.

* Recurrence Tree Method :

IF $T(n) = n\,T\left(\frac{n}{2}\right) + n^2$ then

$n/2$ is become division factor
n^2 is Root node
n is become a number of division.

Page No.
Date : / /

* Master Method :

$$T(n) = a\,T\left(\frac{n}{b}\right) + \Theta(n^k [\log n]^p)$$

If $\log_b a > k \rightarrow \Theta(n^{\log_b a})$

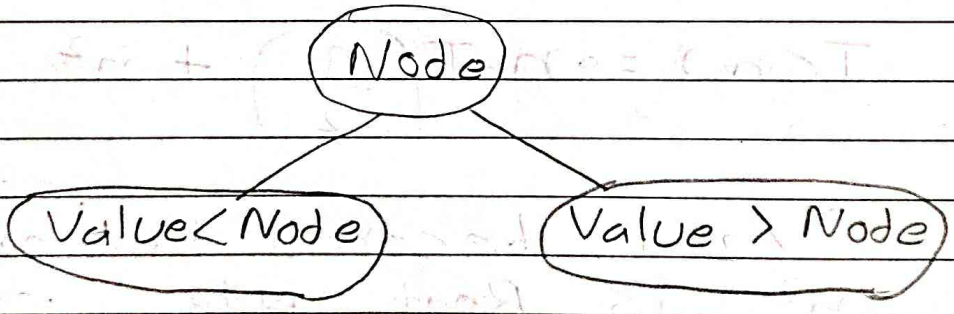
If $\log_b a < k$,

If $P < 0 \rightarrow O(n^k)$
If $P \geq 0 \rightarrow \Theta(n^k (\log n)^p)$

If $\log_b a = k$,

If $P = -1 \rightarrow \Theta(n^k \log(\log n))$
If $P < -1 \rightarrow \Theta(n^k)$
If $P > -1 \rightarrow \Theta(n^k (\log n)^{P+1})$

* Binary Search Tree :



Best Case - O(log n)

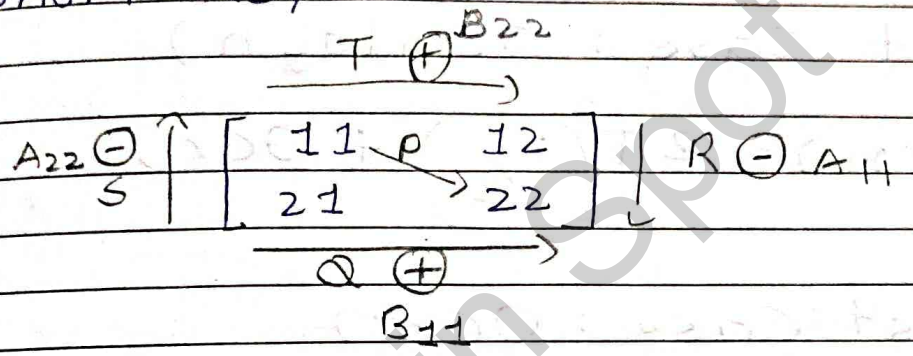
Worst Case - O(n)

* Stranssen's Matrix Multiplication :

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$A \times B = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

- Short Key :



$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$Q = B_{11}(A_{21} + A_{22})$$
$$R = A_{11}(B_{12} - B_{22})$$

$$S = A_{22}(B_{21} - B_{11})$$
$$T = B_{22}(A_{11} + A_{12})$$

$$U = (A_{21} - A_{11})(B_{11} + B_{12})\ (S+T)$$
$$V = (B_{21} + B_{22})(A_{12} - A_{22})\ (Q+R)$$

$$C_{11} = P + S - T + V$$
$$C_{12} = R + T$$
$$C_{21} = Q + S$$
$$C_{22} = P + R - Q + U$$

* Merge Sort :

$T(n) = 2T(n/2) + n \quad , \quad n > 1$

Time Complexity - $O(n \log n)$

* Quick Sort :

- Best Case : $\Theta(n \log n)$

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

- Worst Case : $O(n^2)$

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

Ch - 4 : Greedy Algorithm

* Greedy Algorithm - At a time best solution, Faster to execute, efficiency.

* Prims Algorithm :

Steps :

1 Create edges list and skeleton
2 Select minimum edge
3 select ~~ed~~ end point of edge
Repeat until n-1 edges are add.

* Kruskal's Algorithm :

Steps :

1 Create edges list and skeleton.
2. Sort the edge list and take First edge from edge list
3 select second edge from edge list
Repeat until n-1 edges are add.

* Fractional Knapsack Problem.

Steps :

1 Find the Density of every item.
2 Sort the table according to Density in ascending order.
3 set total weight and total Profit =0.

4 Pick the item, ~~uti~~ until, you reach the maximum weight condition

5 To add last Item,

Profit = Last Profit + (Item density x Value)

This is last total Profit.

Unit : 5 : Dynamic Programming

* Dynamic Programming - Optimal Solution and better solution to the Greedy approach.

* Matrix Chain Multiplication.

i/j	1	2 ----
1		
:		

if i = j → then take 0

if i > j → then X

else Min (MCi, k) + MCk+1, j) + di-1, dk dj))

i <= k < j

->Answer will get top of the table corner.

K = value of i which is increment untit K<j

* Longest Common Subsequence:

i\j		0	1
	Ai/Bj	String	String - - -
0	String		
1	String		

If i = 0 or j = 0 -> take 0

If $A_i = B_j$ -> C[i-1, j-1]+1 ↖

If $A_i \neq B_j$ -> Max[C[i, j-1]←, C[j-1, j]↑)

-> Answer will be Bottom corner of the table.

Note: In all the Problem last Solution is Reaming. This is only table create Method

Page No.
Date : / /

* 0-1 Knapsack Problem:

i\M	0	1	- - -
0			
1			
:	:	:	

IF $i = 0$ or $M = 0$ → take 0

IF $w_i > M$ → $C[i-1, M]$

else. Max $[v_i + C[i-1, M-w_i],$
$C[i-1, M]]$

Answer will be Bottom corner of the table.

* Making Change Problem:

d_i	i/j	0	1 - - -
-	0		
1	1		
:	:	:	

D = Coin value.

If $i = 1 \rightarrow 1 + C[1, j - d_1]$

If $j < d_i \rightarrow C[i-1, j]$

else $\min(C[i-1, j], 1 + C[i, j - d_i])$

→ Answer will be Bottom Corner of the table.