IMP

formal def
graph
E.g

O ⟶ Upper Bound
Ω ⟶ Lower Bound
Θ ⟶ Tight Bound

12/07/2024

* Asymptotic Notations

- Mathematical way of Representing time complexity.

1) Big-Oh-O (order of function)
2) Big-Omega Ω
3) Theta Θ

- Common Runtime time complexities are

$O(2^n)$, $O(n^3)$, $O(n)$, $O(\log n)$, $O(n \log n)$, $O(n^2)$

1) Big-Oh

- When we do priore analysis of algorithm without execution no. of time instructions executed is represer by Asyntotic Notation.

-(Upper Bound of running time)

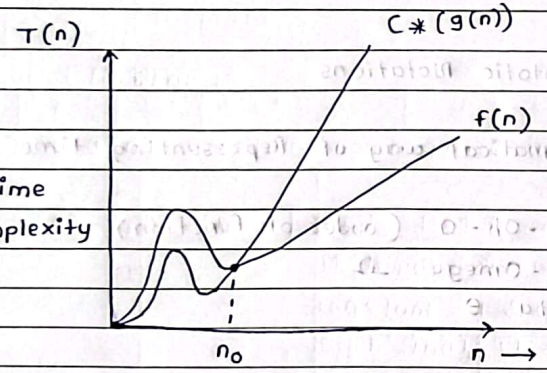
- Using Big-Oh we can use longest amount of time taken by algorithm to complete.

formal definition of Big-Oh

Given : two functions $f(n)$ & $g(n)$ we say that $f(n)$ is $O(g(n))$ if there is exists constant $c > 0$ & $n_0 \geq 0$ such that

$$f(n) \leq c * g(n) \text{ for all } n \geq n_0$$

__/__/____



f(n) is asymtotically bounded above by g(n) up to constant factor c

Example) $f(n) = 3n^3 + 2n^2 + 5n + 1$

Step 1 : find the dominant term (highest order of term)
Examine the function & find the term having highest order of growth. Ignore the constant & lower order term

Step 2 : Determine the order of growth.

Step 3 : Write the Big-Oh notation, where f(n) represents the dominant term

1) $3n^3$
2) n^3
3) $O(n^3)$

__/__/____

Example) $f(n) = 1000n = O(n)$

Example) $f(n) = n^2 + 3n + 1 \longrightarrow O(n^2)$

Example) $3n^2 - 100n + 6 = O(n^2)$

$f(n) = 3n^2 - 100n + 6$

$g(n) = n^2$

$3n^2 - 100n + 6 \leq c.n^2$ for some c

$c = 3$

2) Big - Omega Notation (Ω)

formal definition :

This Notation is used represent the ~~dash (-)~~ lower Bound of algorithm running time

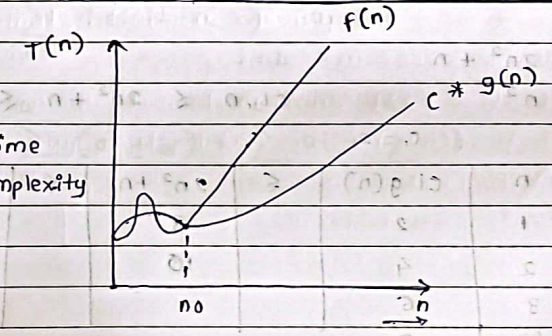
Using omega Notation we can denote shortest time of algorithm.

The function f(n) is said be in Omega g(n)

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

If the f(n) is bounded below by some positive constant multiple of g(n) such that

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



15/07/2024

3) Theta θ

Example)

formal definition

f(n) & g(n) are two non-negative integer function

$f(n) = \Theta\, g(n)$ | Read. f of n is theta of g(n) if there exist two positive constant C1 & C2 & value no such that $C1 \cdot g(n) \leq C2 \cdot g(n)$ for all $n \geq n_0$

That is it is bounded from above & below by the same function g(n).

Example) $f(n) = 4n + 1$

$g(n) = n$

$C1 \cdot n \leq 4n + 1 \leq C2 \cdot n$

$C_1 = 4$ $C_2 = 5$

n	C1.g(n)	≤	4n+1	≤	C2.g(n)
1	4		5		5
2	8		9		10
3	12		13		15
4	16		17		20

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

$g(n) = n^2$ $C1 \cdot n \leq 2n^2 + n \leq C2 \cdot n$

$C_1 = 2$ $C_2 = 3$

n	C1.g(n)	≤	$2n^2+n$	≤	C2.g(n)
1	2		3		3
2	4		10		6
3	6				9
4	8				12

* Mathematical Analysis of non-recursive Algorithm

Algorithm

- Iterative / Non-Recursive
  - loops
- Recursive
  - functions

1) Decide the input size base on parameter n.
2) Identify Algorithms basic operations.
3) Check How many times basic operation is executed.
4) then find whether the execution of basic operation depends on the input size n.
5) Determine the best; worst, Average case time complexity for n.
6) If the basic operation depends on best; worst, average case then it has to be analysic seperately.
7) Set of same for no. of times basic operation is execute
8) Simply the sum using standard formula & rules for manipulation.

* Rules :

i) $\sum_{i=1}^{u} C\, a_i = C \sum_{i=1}^{u} a_i$

ii) $\sum_{i=1}^{u} (a_i \pm b_i) = \sum_{i=1}^{u} a_i + \sum_{i=1}^{u} b_i$

__/__/____

* Summation formala :

1) $\sum_{i=1}^{n} 1 = 1+1+1+1 \ldots n = n = \Theta(n)$

2) $\sum_{i=l}^{u} 1 = u - l + 1$ where $l \le u$ are some lower & Upper Bound.

3) $\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = 1+2+\ldots+n = \frac{n(n+1)}{2} \approx \frac{1}{2} n^2 = \Theta(n^2)$

4) $\sum_{i=1}^{n} i^k = 1+2^k+3^k+\ldots n^k = \frac{n^{k+1}}{k+1} = \Theta(n^{k+1})$

Example) Consider the problem finding the value of largest element from the list of n number.

Pseudocode:

⟶ Algorithm MaxElement (A[0 ... n]

// Determine the value of the largest element in a given array

// input : An array A [0 ... n-1] of integer

// Output : the largest value from array A

maxvalue ⟵ A[0]

For i ⟵ 1 to n-1

if A[i] > maxvalue

maxvalue ⟵ A[i]

return maxvalue

Steps :

two operation :

i) A[i] > maxvalue

ii) maxvalue ⟵ A[i]

Array ele not equal - Distinct

__/__/____

Let's us denote C(n) no. of time comparison executed.

$l = 1 \quad u = n-1$

Apply formula ;

$\sum_{i=1}^{n-1} 1 = u - l + 1$

$= n-1-1+1$

$= n-1$

$= n$

* Summation formula 19/07/2025

5) $\sum_{i=1}^{n} a^i = 1+a+\ldots a^n = \frac{a^{n+1}-1}{a-1}$

* Element Uniques Problem :

Example) Check whether all the element in the array are distinct element

⟶

Algorithm Uelement (A[0 ... n-1])

// input : array

// Output : Array element are distinct or not.

For i = 0 to n-2 do

j = i+1 to n-1 do

if A[i] == A[j] return False

return true

i ⟶ n-2

0	1	2	3	4
10	20	30	40	50

n = 5

i = 0	J	
A[i] = 10	1 = 20	T
	2 = 30	T
	3 = 40	T
	4 = 50	T
i = 1		
A[i] = 20	2 = 30	T
	3 = 40	T
	4 = 50	T
i = 2		
	3 = 40	T
	4 = 50	T
i = 3		
	4 = 50	T

1) Problem size = n
2) Basic Operation = comparison
3) Best Case & worst case are different
4) Best Case = first two elements are equal

worst case = If all elements are unique then all sequences will be executes.

$C_{worst}(n)$ = Outerloop, inner loop

$$\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 \qquad \therefore u - l + 1$$

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

$$= n - 1 - i - 1 + 1$$

$$= n - i - 1$$

$$\sum_{i=0}^{n-2} n - i - 1$$

Substitute value of i

$$= (n-0-1) + (n-1-1) + (n-2-1) + (n-3-1) + \ldots$$

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

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

$$= \theta(n^2)$$

Example) Matrix Multiplication

```
Algorithm matmul (A[0... n-1, 0... n-1], B[0...n-1, 0...n-
// matrix multiplication
// input : two matrices A & B
// Output : C Matrix Containing multiplication of A&B
    for i ⟵ 0 to n-1 do
        for j ⟵ 0 to n-1 do
            c[i,j] ⟵ 0
        For k ⟵ 0 to n-1 do
            c[i,j] = c[i,j] + A[i,k] * B[k-j]
    return c
```

__/__/____

1) Input size =

2) Order of matrix = n

3) Basic operation in inner most loop = Addition & Multiply.

$C(n)$ = Outer loop × inner loop × innermost loop

i - for loop × j - for loop × k - for loop

$$C(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} (1)$$

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (n)$$

$$= \sum_{i=0}^{n-1} (n)^2$$

$$= (n)^3$$

Example) Summation all element

Algorithm Sum (n;)

// input : A

$S \leftarrow 0$

for $i = 0$ to n do

$S \leftarrow S + i$

return S

Order of growth = n

Time complexity = $O(n)$

24/07/24

Example 1) $T(n) = 4T\left(\frac{n}{2}\right) + n^2$

Step 1 : identify a, b, f(n)

$a = 4$

$b = 2$

$4\sqrt{n} = n^2$

Step 2 : Compute $\log_b^a = \log 2^2 = n^{\log_b a} = n^2$

$= 2$

Step 3 : Compare F(n) with $= n^{\log_b^a}$

Step 4 : Determine case : 2

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

Step 5 : Apply master method

$T(n) = \theta(n^2 \log n)$

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

Step 1 : identify a, b, f(n)

$a = 9$

$b = 3$

$f(n) = n$

Step 2 : Compute $\log_b^a = \log_3^9$

$= 2$

Step 3 : Compute F(n) with $= n^{\log_3^9}$

$= n^2$

Step 4 : Determine Case : 2

$f(n) = n^2$

Step 5 : Apply Master Method

$T(n) = \theta(n^2 \log n)$

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

Step 1 : identify a, b, f(n)

$a = 2$

$b = 2$

$f(n) = n^2$

Step 2 : Compute $\log_b^a = \log_2^2$

$= 1$

Step 3 : Compute F(n) with $= n^{\log_b^a}$

$= n$

Step 4 : Determine Case : 2

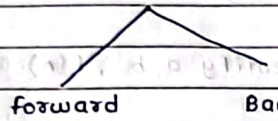
$f(n) = n \log_b^a = n^1$

Step 5 : Apply master method - case 3

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

26/07/2024

* Substitution method to solve recurrance Relations :



- One of simplest method for solving simple recurrance relation.
- We solve the recurrance relation for n = 0, 1, 2 ... until we see a pattern then we make a guess work & predict the running time.

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

- we can calculate the running time for n = 0, 1, 2 as follows.

n	T(n)
1	1
2	T(2-1) + 1 = 2
3	T(3-1) + 1 = 3
4	T(4-1) + 1 = 4

- Where the value of n = k & T(n) = k & also the T(n) = n

* Backwork Substitution

- In Backwork substitution we do opposite of forward substitution.

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

__/__/____

$$T(n) = T\left(\frac{n}{2}\right) + c \quad \text{——} (1)$$

Put $n = n/2$

$$T\left(\frac{n}{2}\right) = T\left(\frac{n}{4}\right) + c \quad \text{——} (2)$$

Put $n = \frac{n}{4}$

$$T\left(\frac{n}{4}\right) = T\left(\frac{n}{8}\right) + c \quad \text{——} (3)$$

Sustitute $T\left(\frac{n}{2}\right)$ in eqn (1)

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

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

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

Substitute $T\left(\frac{n}{4}\right)$ in eqn — (3)

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

$$= T\left(\frac{n}{2^3}\right) + 3c \quad \text{——} \text{ Ans}$$

$$T(n) = T\left(\frac{n}{2^4}\right) + 4c \quad \text{——} \text{ Ans}$$

$$T(n) = T\left(\frac{n}{2^5}\right) + 5c \quad \text{——}$$

$n = 2k$

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

$$= 1 + \log n \cdot c \quad \therefore 1 \text{ \& } c \text{ are constant}$$

__/__/____

$n = 2^k$ ①

$\log n = \log 2^k$

$= k \log 2$

$\log n = k$ (1)

Put k values

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

Given T(n) we can calculate the value of T(n/2)

From above Recurrence

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

29/07/2024

* Backward Substractor

$T(n) = T\left(\frac{n}{2}\right) + a(1)$

$\boxed{T(n) = T\left(\frac{n}{2}\right) + c}$ ← constant —— (1)

Put $n = n/2$

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

Put eqn (1)

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

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

Put $n = n/2$ in eqn (1)

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

$T(n) = \left[T\left(\frac{n}{8}\right) + 2c\right] + c$

$\boxed{T(n) = T\left(\frac{n}{8}\right) + 3c}$

Put $n = n/2$ in eqn (1)

$T\left(\frac{n}{8}\right) = \left[T\left(\frac{n}{16}\right) + c\right]$

$T(n) = \left[T\left(\frac{n}{16}\right) + 3c\right] + c$

$\boxed{T(n) = T\left(\frac{n}{16}\right) + 4c}$ ....

__/__/____

Generalize eqⁿ :

$T(n) = T\left(\frac{n}{2^k}\right) + k.c$

$n = 2k$

$T(n) = T\left(\frac{n}{2^k}\right) + k.c$

$= T(1) + k.c$ ——— ①

$n = 2^k$ ∴ Take log of both sides

$\log n = k \log 2$

$k = \log_2 n$

$T(n) = T(1) + k.c$

$= 1 + \log_2 n . c$

↑ Discard ↑ constants

$T(n) = \log_2 n$

Example) $T(n) = 2T\left(\frac{n}{2}\right) + c$ ——— ①

Put $n/2$

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

Put eqⁿ ①

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

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

__/__/____

$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + 3c$

Put eqⁿ (1)

$T(n) = 2T\left[4T\left(\frac{n}{8}\right) + 2c\right] + c$

$\boxed{T(n) = 8T\left(\frac{n}{8}\right) + 5c}$

$T\left(\frac{n}{8}\right) = 2T\left(\frac{n}{16}\right) + 3c$

Put eqⁿ ①

$T(n) = 8T\left[2T\left(\frac{n}{16}\right) + 3c\right] + c$

$\boxed{T(n) = 16T\left(\frac{n}{16}\right) + 4c}$

Example $T(n) = T\left(\frac{n}{3}\right) + c$ ——— ①

Put $n = n/3$ in eqⁿ ①

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

eqⁿ (1)

$$T(n) = \left[T\left(\frac{n}{9}\right) + c\right] + c$$

$$\boxed{T(n) = T\left(\frac{n}{9}\right) + 2c} \quad \text{——— ②}$$

Put $n = n/9$

$$T\left(\frac{n}{9}\right) = T\left(\frac{n}{27}\right) + c$$

eqⁿ (1)

$$T(n) = \left[T\left(\frac{n}{27}\right) + 2c\right] + c$$

$$\boxed{T(n) = T\left(\frac{n}{27}\right) + 3c}$$

Ans : $T(n) = c.\log_3 n + 1$

* Divide & Conquer

```
Algorithm DC (P)
{
    if P is too small then
        return soln of P
    else
        Divide (P) and obtain P1, P2, P3 ... Pn
        where n >= 1
            Apply DC to each subproblem
                return combine (DC(P1), DC(P2) ....)
}
```

Recurrence Relation

$$T(n) = \begin{cases} g(n) & \text{if } n \text{ is small} \\ T(n_1) + T(n_2) + \dots T(n_r) + f(n) & \end{cases}$$

* Binary Search Recursive Algorithm

```
Algoritlm BinSearch (A, key, low, high)
// This algorithm is to search an element using Binary
   Search
// input : Array : A & key
// output : Index of Search key if key is found
low = 0
high = n - 1
mid = (low + high) /2
if (key = A[mid]) then return mid {
else
    if (key < A[mid] them
        BinSearch (A, key, low, mid-1)
```

31/07/24

```
else
    If (key BinSearch (A, key, mid+1, high)
}
```

Analysis :

$C_{worst}(n) = C_{worst}\left(\frac{n}{2}\right) + 1$ ← if key is at mid (one comparison)

$C_{worst}(1) = 1$ ← Time Required to compare left & Right list.

	Best	Avg ~~worst~~	Worst
Successfull Search			
UnSuccessful Search			

Assume $n = 2^k$ then,

$C_{worst}(2^k) = C_{worst}\left(\frac{2^k}{2}\right) + 1$

$$C_{worst}(2^k) = C_{worst}(2^{k-1}) + 1 \quad \text{---} (1)$$

Using Backword substitution,

$$C_{worst}(2^{k-1}) = C_{worst}(2^{k-2}) + 1$$

$$C_{worst}(2^k) = \left[C_{worst}(2^{k-2}) + 1\right] + 1$$

$$C_{worst}(2^k) = C_{worst}(2^{k-2}) + 2$$

$$C_{worst}(2^{k-3}) = C_{worst}(2^{k-3}) + 1$$

$$C_{worst}(2^k) = \left[C_{worst}(2^{k-3}) + 3\right]$$

$$C_{worst}(2^k) = C_{worst}(2^{k-k}) + k.3$$

$$= C_{worst}(2^0) + k.3$$

$$= C_{worst}(1) + k.c$$

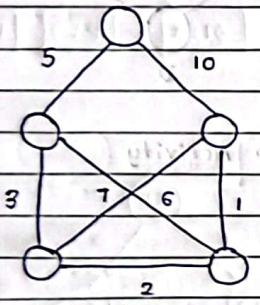
$$= 1 + \log_2 n \cdot c$$

Discard ↑ ↑ constants

$$C_{worst}(2^k) = \log_2 n$$

iv) Greedy method works in stages only on input considered at each time, & based on these input it is considered or decide whether particular input gives optimal solution.

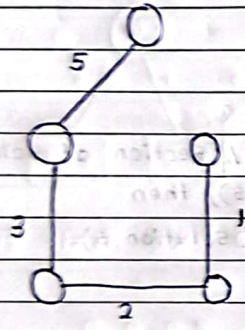
Example)



Graph G

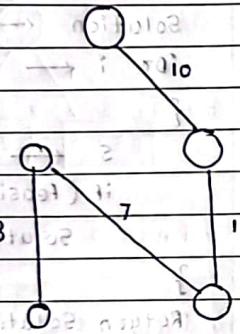
→

1)



$w(T_1) = 11$

2)



$w(T_1) = 21$

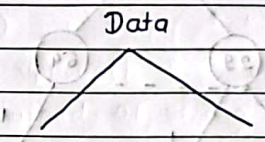
* Haffman Coding

Data → | a | b | c | d | e | f | g | h | h | a | a |

↑ 8 bits

11 x 8 = 88 bits

- Haffman encoding contain the data in the form of sequence of character
- Create a table having Frequency of occurances of each character in
- From this table Haffmans tree is constructed.



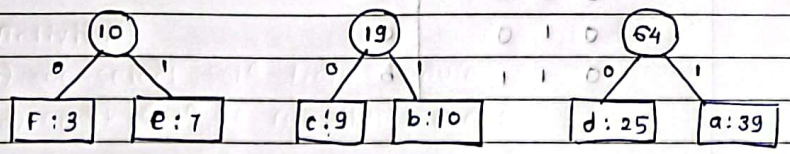
E.g). a : 39 b : 10 c : 9 d : 25 e : 7 f : 3

Step 1 :

i) Arrange this symboles in ascending order.

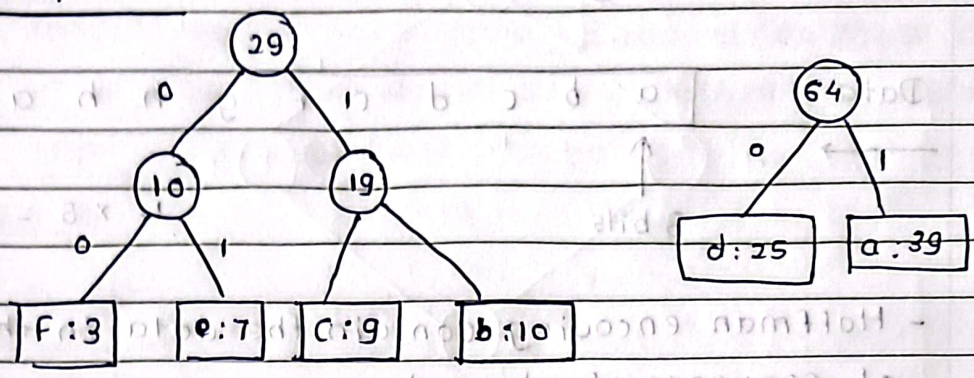
f : 3	e : 7	c : 9	b : 10	d : 25	a : 39

Step 2 :

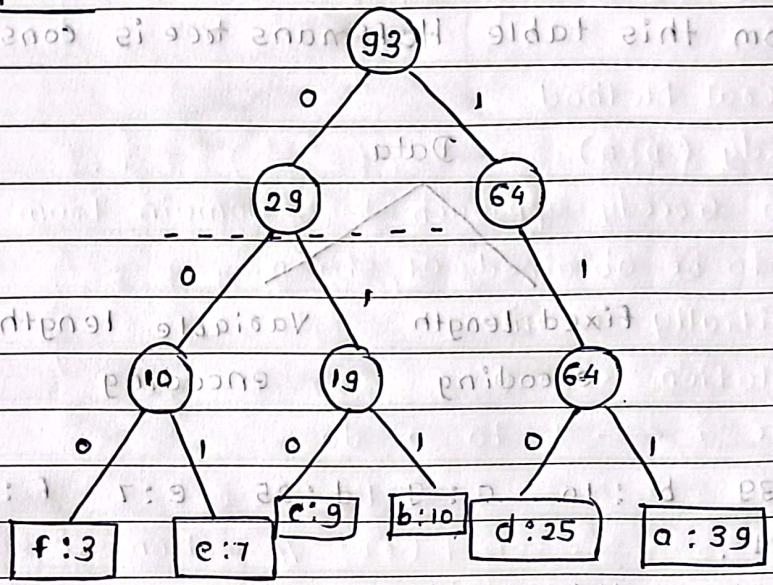


__/__/____

Step 3 :



Step 4 :



as there are 6 character $2^3 = 8$ combinan so free

0 0 0 ~~0~~	f
0 0 1 ~~0~~ x	e
0 1 0	c
0 1 1	b

09/08/24

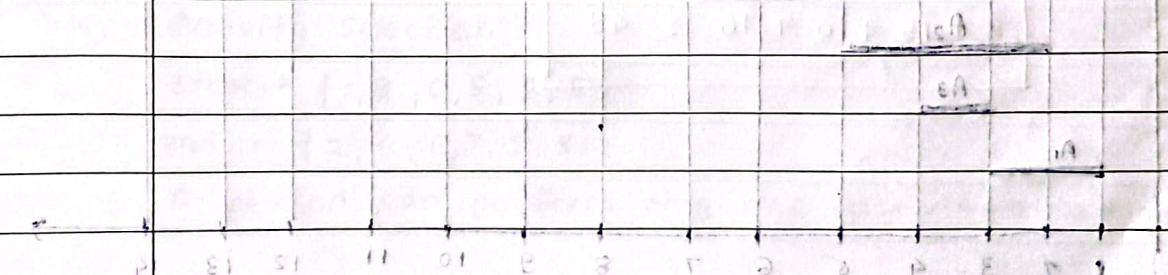
* Activity Selection using greedy approach.

- Greedy algorithm strategies builds up soln piece by piece always choosing the next piece that offers most & obvious and immediate benefits.
- Greedy algorithm are used for optimization problem having following property.
  - At Every step we can make a choice that looks based at the moment. & we get optimal solution to the complete problem.

You are given 'n' activities with their start & finish time. Select the maximum no. of activities that can be performed by single person assuming that a person can only work on single activity at a time.

⟶ Suppose $S = \{1, 2, 3 \ldots n\}$ is the set of proposed activities, the activities shared the resource which can be used by only one activity at a time. ex - Tennis court, lecture Hall

- each activity 'i' has starts & finish time where $S \leq f$ (s) (f)
- activities i & j are compatible if the intervals do not overlap
- The activity selection problem chosen the maximum set of mutually consitent activities.



Given 10 activities :

$S = \{A1, A2, A3, \ldots A10\}$

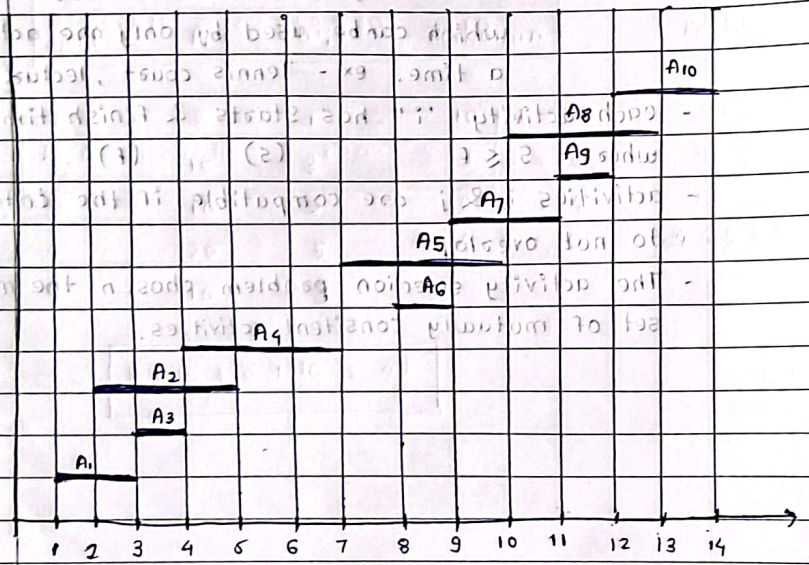
$S_i = \{1,2,3,4,7,8,9,9,11,12\}$

$F_i = \{3,5,4,7,10,9,11,13,12,14\}$

Compute the Schedule where the greatest no. of activities take place.

Step 1 :

Activity	A_1	A_3	A_2	A_4	A_6	A_5	A_7	A_9	A_8	A_{10}
Start	1	3	2	4	8	7	9	11	9	12
Finish	3	4	5	7	9	10	11	12	13	14



Greedy activity Selector (s, f, n)

1) n ← length {s}
2) A ← {1}
3) j ← 1
4) for i ← 2 to n
5) do if $s_i \geq f_i$
6) then A ← A ∪ {i}
7) j ← i
8) Return A

12/08/2024

* Solve the Following :

Q.1) n = 5 items
w = 3 3 2 5 1
P = 10 15 10 20 8
knapsack Capacity = 10 kg

Q.2) Give items a {value, weight} {40, 20} {30, 10} {20, 5}
knapsack Capacity = 20 Find max value o/p

Q.3) knapsack capacity = 40 find maximum value o/p assuming items to be divisible & non-divisible
{60, 20} {50, 25} {20, 5}

Q.4) Activity Selection
start = {1, 3, 0, 5, 8, 5}
end = {2, 4, 6, 7, 9, 9}
A person can perform only one activity at a time.

Step 1 :

Object	1	2	3	4	5
weight	3	3	2	5	1
Profit	10	15	10	20	8
pi/wi	~~3.3~~	~~5~~	~~5~~	~~4~~	~~8~~
	3.3	5	5	4	8

Find pi/wi ratio

Step 2 : Arrange in decreasing order

	1				
Obj	5	2	3	4	5
W	1	3	2	5	3
P	8	15	10	20	10

Step 3 :

Object	Pi	Wi	Remainin C	Profit
1	8		10 - 1 = 9	8
2	15	3	9 - 3 = 6	15
3	10	2	6 - 2 = 4	10
4	20	5	4 - 4 = 0	5 % 5 x 20 = 16
5	10	3		

8 + 15 + 10 + 16 = 49

Q.2) Step 1 : find Pi/wi ratio

(Pi) value	40	30	20
(wi) Weight	20	10	15
Pi/wi	2.1	3	~~3~~ 4

Step 2 : Arrange desending order

Pi	~~30~~	~~40~~	20	30	40
wi	10	20	15	10	20

Step 3 :

Pi	Wi	Remaining Capacity	Profit
~~30~~	~~10~~	20 - 10 = ~~10~~	~~30~~
~~40~~	~~20~~	~~10 - 10~~ = ~~0~~	~~10 % 20 X 40 =~~
20	15		
20	5	20 - 5 = 15	20
30	10	15 - 10 = 5	30
40	20	5 - 5 = 0	5 % 20 X 40 = 10

20 + 30 + 10 = 60

Q.3)

Step 1 : Ratio P_i / w_i

P_i	60	50	20
W_i	20	25	5
P_i/w_i	3	2	4

Step 2 : Descending order

P_i	20	60	50
w_i	5	20	25

Step 3 :

P_i	W_i	Remaining Capacity	Profit
20	5	40 - 5 = 35	20
60	20	35 - 20 = 15	60
50	25	15 - 15 = 0	15 / 25 × 50 = 30

divisible = 20 + 60 + 30 = 110

Non-divisible = 60 + 20 = 80