DESIGN AND ANALYSIS OF ALGORITHM

21/3/2024

Algorithm

- Step by step Process (Unambiguous)
- Algorithm must change his behaviour while increasing n value
- Set of procedures to solve a problem

◈ - Understanding the given problem

Program comes under **Complexity**

* TIME → How much time
* SPACE → How much space needed to compute the output

Performance Analysis

↳ BestCase (mt, ms)
↳ Avg Case (At, As)
↳ WorstCase (Mt, Ms)

M - Maximum
m - minimal

A = x , "t" // 500 Kb
B = x , = $t_1 + t_2$ // 500 Kb

Notation of Algorithm

problem → Input → Algorithm → output

* Computational thinking
* Algorithimic thinking

Problem Classification

◈ Classification

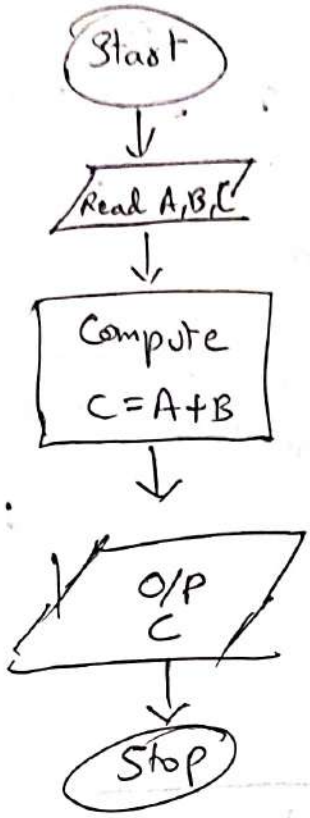
⊡ Computational Problem
↳ Formalization
↳ Characterization

Ex: (Remodeling the answer)
Structured Problem (eg-Sorting)
Linear search / Binary search → Searching Problem
Construction Problem (Conditional Problem)
Decision Problem
Optimization Problem (Getting Best Value/ Solution)

Every intermediate time you will get a solution which is known as feasible solution (Initial Part)

The intermediate results →
The final results → Optimal solution
The last result is →

Flowchart → To Understand the flow of process

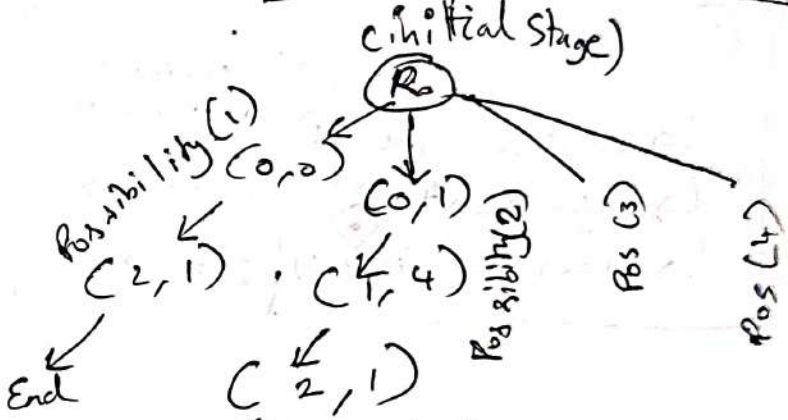


ALGORITHM

START
READ (A, B, C)
COMPUTE (C = A + B)
// END

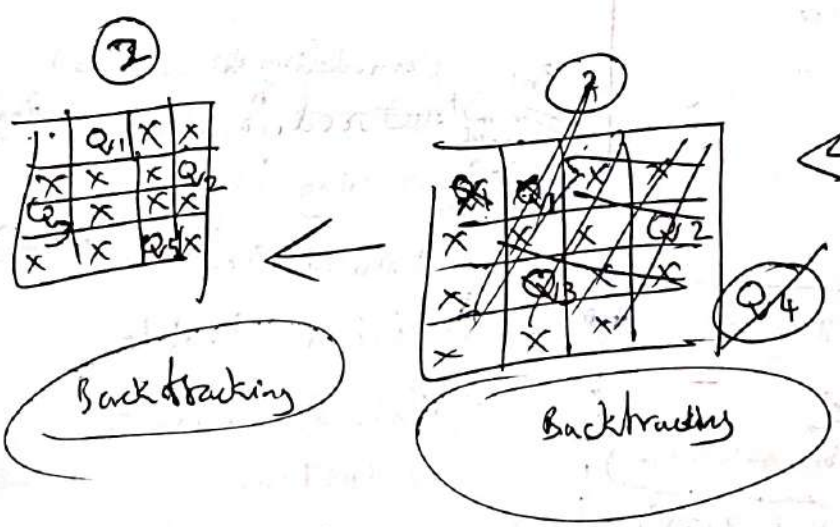
Searching Problem

STATE SPACE TREE



Construction Problem (Conditional Problem)

4 Queens Problem



Q_1 Q_2 Q_3 Q

BACKTRACKING

(3)

Non-Computational Problem

↳ Deterministic algorithm / tractable

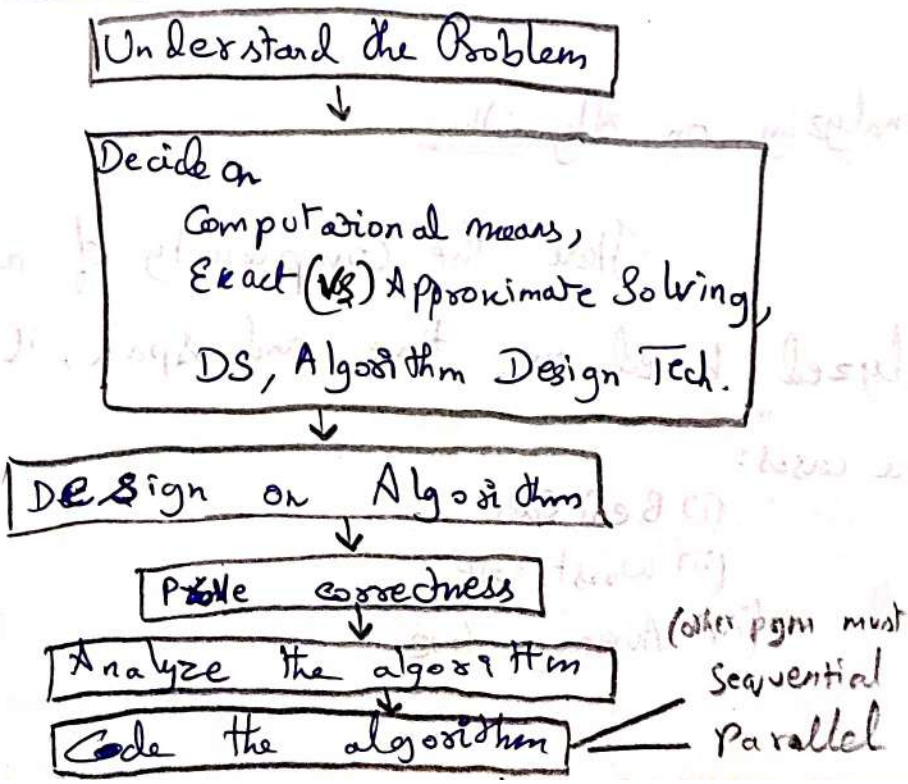
↳ Non-Deterministic algorithm / intractable

- Deterministic
  - Determining problem datas.
  - Possible to solve the problem is defined as Deterministic
  - Able to determine the time is tractable

- Non-Deterministic
  - Not possible to determine the problem is defined as non-deterministic.
  - Not Possible to determine the time is intractable

23-3-2024

Analysis of Algorithms



classifications of Algorithm

↳ By types of Problems.

↳ By Implementation

↳ By Design Techniques — Divide/Conquer, Back tracking, Greedy tech, Dynamic prg

↳ By Design Complexity (time/space) — Three cases: best, avg, worst

CHARACTERISTICS OF ALGORITHM

* I/P (n) — Size of Input
* O/P (Unambiguous)
* Definiteness — clearly defined
* Finiteness — finite amt of time
* Effectiveness — should be effective

Study of Algorithm

① How to devise Algorithm (How to plan the Algorithm)

② Validation

③ Analyze

④ Testing

↳ 2 phases

(i) Debugging

(ii) Profiling. (Performance) Calculated by < time / Space (Effectiveness)

Analyzing an Algorithm

Here the complexity of algorithm is analyzed based on time and space, it categorized into three cases:

(i) Best Case

(ii) Worst Case

(iii) Average Case

(i) Let (D_n) - Domain of a problem where n is the size of input

(ii) Let ("$I \in D_n$") - Here where an Instance of an Problem taking from the domain (D_n)

(iii) $T(I)$ - The Computation time of the Algorithm

(i) BestCase Analysis (B(n))

- An algorithm has bestcase running time (B)(n) for the any input of size n is the minimum computed time of the algorithm with respective all instance from the respective domain.

- Mathematically it defined as

$$B(n) = \min \{ T(I) \mid I \in D_n \}$$

(ii) Average Case Analysis (A(n))

- An algorithm has average case running time (An) for any input of size n is the avg computed time to algorithm with respect to All instance from the respective domain

- Mathematically it defined as

$P(I)$ = Probability of encountering a specific input.

$$A(n) = \sum_{I \in D_n} P(I)\, T(I)$$

Worst Case Analysis ($W(n)$)

- An algorithm has worst case running time ($W(n)$) for any input of size n is the maximum computed time of the algorithm with respective all instance from the respective domain

$$W(n) = \max\{T(I) \mid I \in D_n\}$$

Example :

- Assume the two Algorithms A and B for a given Problem P. The Time Complexity of A & B is

Algo. A = $3n$

Algo. B = 2^n (exponential type)

n - size.

which algorithm should be selected assuming that all other conditions remaining the same for both all other algorithms -

Size 'n'	A = 3n	B = 2^n
1	3	2
2	6	4
3	9	8
4	12	16

Conclusion

The algorithm A which is $3n$ should be selected

* For size ≤ 3 b is fastes
* For size ≥ 3 A is fastes

Analysis of Iterative Algorithm

(same step repeated again until answer found)

Steps involved in Mathematical Analysis of Iterative Algorithm

* Measure the input size (Length of input data)
* Measure the running time using either < step count / Operation count
* In case of operation count find the [worst / best & / avg case] efficiencys of the algorithm.
* Identify the rate of growth, this step determines the performance of algorithm when the input size is scaled to higher value.
  (increasing the input value.)
* The Behaviour of algorithm determine only after observing its performance when the input data are scaled to higher value

Measuring Input size

-> Generally Algorithm ~~analysis~~ analysis is depend on size of Input (Binary format).

-> The efficiency of algorithm is always expressed as a function of input size.

STEP COUNT

-> The idea is to count the number of instructions or steps that are used by an algorithm.

-> Find steps per execution and Frequency of statements.

-> Executable statement's = 1

Non - Executable statement's = 0

25-3-24

Step count

(i) Declarative statements with Initialization = 1

Declarative statements with no initialization = 0

(ii) Expression have -> 1

Combination of operator & operations)

c = a + b

(9)

(iii) Comments, brackets (Begin, if, End if, while) → 0

(iv) Return statement, function Invocation, Assignment statement (Break, Continue, goto) → 1

Eg:

Consider the following Algorithm Segment find Step Count.

```
Algorithm Simple (A, B, C)
Begin
    A = B + 1
    C = A + 2
End
```

(v) Frequency of an algorithm is denoted, and the number of times the instruction is executed, (How many time the instruction will be executed) therefore the total count can be obtained by multiplying the frequency and the stepcount

~~Step no~~ ~~Algorithm~~

~~Sol:~~

Sl.:

Step No	Algorithm	Steps per Execution	Frequency	total Count
1	Algorithm Simple(A,B,C)	0	-	
2	Begin	0	-	
3	A = B=1	1	1	1
4	C = A + 2	1	1	1
5	End	0	-	
			T(n) = 2	

(Multiply of Steps & frequency)

↑ Constant type Algorithm

(Bcz the run value is constant)

PART-B Q will be like

(6M & 5M)

Find the Stepvalue for the sum of odd numbers

STEP NO	ALGORITHM	Steps per execution	Frequency	total count
1	Algo sum()	0	0	0
2	Begin	0	0	0
3	sum=0	1	1	1
4	For i=1 to n do	1	n+1	n+1
5	Sum = Sum + 1	1	n	n
6	End For	0	-	-
7	Return sum	1	1	1
8	End	0	-	0

$n \times (n+1) \times 1$

$= 1 + n + 1 + n + 1$

$2n + 3$ (ignored)

$= n$

$T(n) = O(n)$

If n=3 i=01

(For) For ($i \leq 3$)

(3+1) (n+1) n=4

1	$\Rightarrow i \leq 3$	✓
2	$\Rightarrow 2 \leq 3$	✓
3	$\Rightarrow 3 \leq 3$	✓
4	$\Rightarrow 4 \leq 3$	✗

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

Step No	ALGORITHM	Steps Per execution	Frequency	Total count
1	Algo SUM( )	0	0	0
2	Begin	0	0	0
3	sum = 0	1	1	1
4	for i=1 to n do	1	n+1	n+1
5	val = 2 * i	1	(*) n	n
6	sum = sum + val	1	n	n
7	End For		0	—
8	return sum	0	1	1
9	End	1 0	0	—

OPERATION COUNT :

- Another method for Performing Algorithm Analysis
  - Here the Basic idea is count the number of operations instead of steps
    - It has two Categories.
      (i) Elementary
      (ii) Non-Elementary

(i) Elementary - It improves primitive operations that are implemented directly eg: Arithmetic & logical operation

(ii) Non-Elementary Operation - these operations are not atomic in nature it involves many primitive operations

Eg: Sorting & finding Maximum & Minimum element from the list

: RULES FOR OPERATION COUNT

Step1 - Count the number of basic operations of a program and express it as a formula.

Step2 - Simplify the formula

Step3 - Represent time complexity as a functions of the operation count

Some additional rules are

(i) Sequence - Let an algorithm in Sequence

```
Begin
  S1
  S2
End
```

] follows Additional Principle i.e ($m+n$) equation

(ii) Selection :-

```
If( __ )
    statement 1;
else
    statements 2;
```

} select 'max' value max(m,n)

(iii) Repetations :-

Eg: Recursive algorithms

follows 'multiplication principle' i.e ($n \times m$)

(13) Consider the algorithm statement and perform Complexity analysis using operation count.

Cost -> C

Step No:	ALGORITHM	Operation Accounted For	Operation Cost	Repetations	Total Count
1	Algo Swap(a, b)	—	—	—	—
2	Begin	—	—	—	—
3	temp = a	Assignment	c_1	1	c_1
4	a = b	Assignment	c_2	1	c_2
5	b = temp	Assignment	c_3	1	c_3
6	End	—	—	—	—

$T(n) = c_1 + c_2 + c_3$

C_n

Constant type Algorithm

Q) $2n^2+2n+1$

Step No.	ALGORITHM	Steps per execution	Frequency	total count
1	~~void~~ Algo add(int a[][], int b[][], int c[][])	0	—	—
2	Begin : int c[][];	0	—	—
3	for int i to n	1	(n+1)	n+1
4				
5	for int j to n	1	n(n+1)	n^2+n
6	c[i][j] = a[i][j] + b[i][j]	1	n^2	n^2
7	END	0	0	0
				$2n^2+2n+1$

STEP COUNT

Q) void mat-Add (int a[][], int b[][], int c[][]).

```
int c[][];
for(i=0; i<n; i++)          (n+1)
{
}
for (j=0; j<n; j++)         n(n+1)
{
c[i][j] = a[i][j] + b[i][j]   n(n)
}
}
}
```

If n=3

0<3	1
1<3	2 (n+1)
2<3	3
3<3	4

//MATRIX ADDITION

Algorithm

Step no:	ALGORITHM	Operation Accounted for	Operation cost	Repetitions	Total count
1	Algo (int a[][], int b[][], int c[][])	—	—	—	
2	int c[i][j];	—	—	—	
3	Begin	—	—	—	
4	for int i to n	Assignment, comparision, Increment	C_1	n+1	$C_1(n+1)$
5	for int j to n	Assignment, comparision, Increment	C_2	n(n+1)	$C_2(n^2+n)$
6	c[i][j] = a[i][j] + b[i][j]	Assignment	C_3	n(n)	$C_3 n^2$
7	END	—	—		
			$C_1n + C_1 + C_2 n^2 + C_2 n +$		$C_3 n^2$

Space Complexity

4/4/24

$S(P) = C + Sp$ → Variable length / Dependent variable

C → Constant / Independent / Fixed type of variable

1 Byte - 8 bits

1 word - 4 bytes

1 time - 1 unit

The space Complexity of the algorithm is the amount of memory required to run any Program, it having two components

(i) Fixed Path Component

(ii) Variable Path Component

(i) Fixed Path Component

It is independent variable, it includes the instruction space, simple variable and constance

(ii) Variable Path Component

It consists of space needed by component variable the size is dependent on particular problem instance Example reference variable and recursion stack space

Algorithm sum

Ex 1
```
{
    a = 5;
    b = 3;
    c = a + b;
}
```

* allocate 1 unit space time = a ⇒ 1 t, b ⇒ 1 t, c ⇒ 1 t

$S(P) = 3$

Reduce Complexity value

$S(P) = n + 3$

$S(P) = n$

By using higher order

Ex 2

```
Algorithm size(a, n)
{
    S = 0.0;
    for i = 1 to n do
        S = S + a[i];
    return S;
}
```

variables ⇒ s, i, n, a[i]

s, i, n → 1 unit of space time; a[i] → n

$S(P) = n + 3$

$S(P) = 3(n+1)$

always have 3 unit of time

Stack (3) space

Ex 3

```
Algorithm RSum(k, n)
{
    If (n ≤ 0) then
        return 0;
    else recursive return RSum function
        (k, n-1) + k[n];
}
```

recursive type

↓ It always use stack data structure

Formal Parameter
return Address
local variable

CONSIDER of GROWTH of an Algorithm

P.T.O

Let consider refotend values for various "n"

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	n!
10	3.3	10	3.3×10	10^2	10^3	2^{10}	3.6×10^6
10^2	6.6	10^2	6.6×10^2	10^4	10^6	3×10^3	9.3×10^{157}
10^3	10	10^3	1.0×10^4	10^6	10^9		
10^4	13	10^4	1.3×10^5	10^8	10^{12}		
10^5	17	10^5	1.7×10^6	10^{10}	10^{15}		
10^6	20	10^6	2.0×10^7	10^{12}	10^{18}		

Algorithm size always depends on input size (n)

Increasing input size will change behaviour of algorithm

Growth rate -> behaviour of algorithm

* Any algorithm is effected to work fast for any input size the change in the behaviour of an algorithm while the input size is increases is called order of Growth of an algorithm.

* To find out the order of Growth of an algorithm always go for higher value of input size n.

* In this table the exponential function and the factorial function both functions grows as fastly that thier values become larger even for small value of n.

* There is a large difference between these two functions its called as exponential growth function.

Efficency of Algorithm

(1) Best Case Efficiency — $C_{best}(n) = 1$

~~Case~~

(2) worst Case Efficency

Case(i) Element at last position

$$C_{worst}(n) = n$$

Case('ii') Elemat Not in last

$$C_{worst}(n) = n$$

(3) Average case

Element not found in first or last position, Elemat may be present in Middle position (i) Let probability of Successful search ⇒ p

$$0 \leq p \leq 1$$

(ii) Let Bobability of Unsuccessful scard

$$1 - p$$

Now find element at ith position from the given list is represented by ~~P/n~~ P/n, therefore the probability of a successful search at ith position can be ⇒ P/n

∴ The element Availability Position in the list,

$$C_{Avg}(n) = \underbrace{\left[(1 \times p/n) + (2 \times p/n) + \ldots\ldots + (n \times p/n)\right]}_{\text{Successful Search}} + \underbrace{n(1-p)}_{\text{Unsuccessful Search}}$$

$$= p/n \left[1 + 2 + \ldots\ldots n\right] + n(1-p)$$

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

Search Algorithm ↓ sub(1)

succes = 1

$$\Rightarrow 1/n \left[\frac{n(n-1)}{2}\right] + n(1-1) \Rightarrow \left[\frac{n(n-1)}{2n}\right] \Rightarrow \text{Success}$$

Unsuccess = 0

$$\Rightarrow 0/n \left[\frac{n(n-1)}{2}\right] + n(1-0) \Rightarrow n \Rightarrow \text{Unsuccess}$$

Let consider three algorithms a, b & c their respective running time complexity is given as follows $T_A = 7n$, $T_B = 7\log_{10} n$, $t_C = n^2$

Compare the rate of work when the input is scaled from n value 100 to 10,000

$7n \Rightarrow 7(100) = 700$

$T_A = 7n \Rightarrow 7(10,000) = 70,000/100$

$T_B = 7\log_{10} n \Rightarrow 7\log_{10}(10,000) \Rightarrow \frac{7\log_{10}(100)}{}$

$T_A = 100$ ← best case

$$T_A = \frac{7n}{7n} \Rightarrow \frac{7(10,000)}{7(100)} = \frac{10,000}{100}$$

$$\boxed{T_A = 100}$$

$$T_B = 7\log_{10}^{n} \Rightarrow \frac{7\log_{10}^{10000}}{7\log_{10}^{100}} \Rightarrow \frac{\log_{10}^{10^4}}{\log_{10}^{10^2}}$$

$$\Rightarrow \frac{\log_{10}^{10^4}}{\log_{10}^{10^2}} \Rightarrow \frac{4\log_{10}^{10}(1)}{2\log_{10}^{10}(1)} \qquad \log_{10}^{10} = 1$$

$$\Rightarrow \frac{4(1)}{2(1)} \Rightarrow \frac{4}{2} \Rightarrow 2$$

$$\boxed{T_B = 2}$$

$$T_C = \frac{n^2}{n^2} = \frac{(10,000)^2}{(100)^2} \Rightarrow \frac{10,000 \times 10,000}{100 \times 100}$$

$$\boxed{T_C = 10,000}$$

T_B is the Best algorithm based on time

Let there be a system with the limit of maximum 5 hours for performing a task. Let there be two algorithms a & b having time Complexity

$$\boxed{\begin{array}{c}\text{Limit of System} = 5\text{hrs} \\ T_A = n^2 \\ T_B = 2n\end{array}}$$ to find n value

How many inputs can be processed by these two algorithms and which is efficient?

(21)

$T_A = n^2 = 18,000$

$n^2 = 18,000$

$n = \sqrt{18,000}$

$T_B = 2n = 18,000$

$n = \frac{18000}{2}$

$n = 9000$

T_B is the Best algorithm Based on number of input.

H.W Let there be a two algorithms $T_A = n^2 + n \quad T_B = n^2$

Assume that each operation takes 1 second to compute a task for $n = 100$. How long it will be take for n value 1000 ?

$n = 100 \;,\; 10000$

$n = 1000 \;,\; ?$

$T_A = \frac{n^2+n}{n^2+n}$

$\Rightarrow \frac{100^2+100}{1000^2+1000}$

$= \frac{1+100}{100+1000} = \frac{101}{1100}$

$\frac{100 \times 100 + 100}{1000 \times 1000 + 1000}$

$100 = 1 \text{ second}$

$T_B = \frac{n^2}{n^2} = \frac{(100)^2}{1000^2}$

$= \frac{100 \times 100}{1000 \times 1000}$

$= \frac{1}{100}$

$1 = 10000$

$100 = ?$

$T_A =$

Asymptotic Notations

8/4/2024

PART B

Prove $2n+3 \in O(n)$ ⁂

$f(n)$, $g(n)$

$f(n)$ ← class function

$g(n)$ ↓ growth function

with the helps this well be able to determine the behaviour of the algorithm.

(i) Big 'Oh' [O] worst

(ii) Big Omega [Ω] best

(iii) Big Theta [Θ] avg

(iv) little oh [o]

(v) little omega [ω]

(vi) Hilde Notation [Ñ]

(v) & (vi) 2 Mark

Asymptotic notation is a way to describe the behaviour of function in the limit or without bound values.

(3) Importance of Notation.

* Used to determine rough estimation of running time of different algorithm.
* It describes the efficiency & the performance of algorithm. It allows the comparision of the performance of different algorithms.
* It always considering large value of inputs

Let $f(x)$ and $g(x)$ be the functions from the set of natural numbers to the set of real numbers

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = c \quad [\text{constant}]$$

Asymptotic

Big 'oh' (O) notation / Upper Bound Notation

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

* where the class function $f(n)$ that grows not faster then $g(n)$
* This notation is used to give <u>upperbound</u> on a positive run time function $f(n)$ where n is the input size.
* This notation is also called asymptotic Upperbound notation

* Let f & g two function defined from the set of natural numbers to the set of non-negative real numbers

$$f, g : N \to R \geq 0$$

$n \geq R = 0$
$R \geq 0$

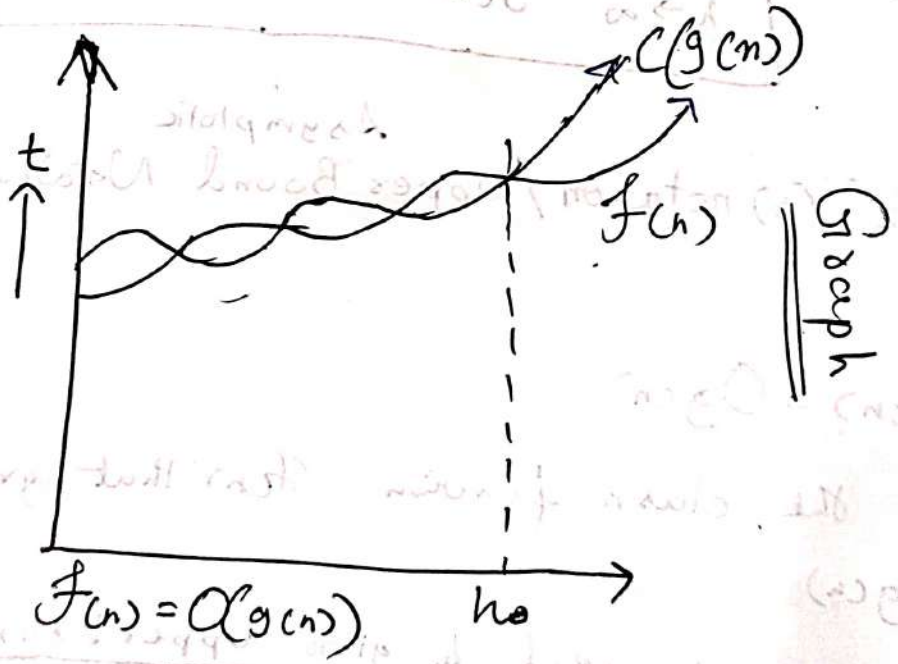
* It is said that $f(n) = O(g(n))$ there exist two positive constants (c, n_0) where

$c \in R$ and $n_0 \in N$

$$\therefore f(n) \leq c(g(n)), \forall\ n \geq n_0$$

class
$f(n)$ will always lesser than growth for $g(n)$

Visualization of $O(g(n))$



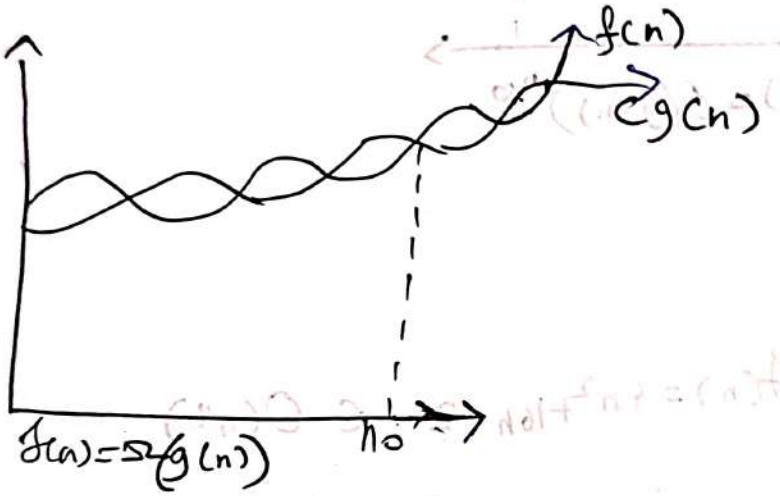
(ii) Big (Ω)mega (Ω) / Asymptotic Lower Bound

"Ω g(n)" is used to give lowerbound on a positive run time function f(n) where (n) is the input size. it is also called Asymptotic Lower bound

Let f & g $f(n) = \Omega\, g(n)$

$$\therefore f(n) \geq c(g(n)) \;\forall\; n \geq n_0$$

Visualization of Ω g(n)



(iii) Big Theta / Asymptotic type Count

Θ g(n) is used to give type count on a positive runtime function f(n) where (n) is the input size, also called as Asymptotic type Count

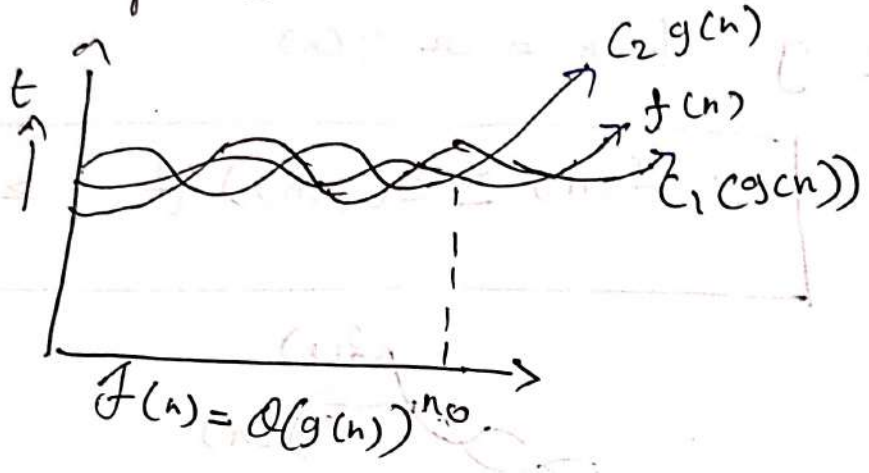
Let f, g two function defined from the set of natural numbers to the set of non-true constants:

c_1, c_2, n_0

$c_1, c_2 \in R$ and $n_0 \in N$

$$c_1(g(n)) \leq f(n) \leq c_2(g(n)), \forall \; n \geq n_0$$

Visualization of $\Theta(g(n))$



Eg:

$f(n) = 2n+7$

$n^3 \leq 2n+7 \leq \log_2$

$1 \leq \log n + n^2 \leq n^3$

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

Eg:

Prove that, $f(n) = 4n^2+16n+2 \in O(n^4)$

① Mark → $f(n) \leq c(g(n))$

$c \in R$

$n_0 \in N$

$f, g : N \to R \geq 0$

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

$n_0 = ?$

$c = 1$

$f(n) = 4n^2+16n+2$

$= 4+16+2$

$g(n) = n^4$

	$f(n)$	$g(n)$	
$n=1$	22	1	×
$n=2$	50	16	×
$n=3$	86	81	×
$n=4$	130	256	✓

$130 \geq 256$

$f(4)$ is lesser than $g(4)$

Hence it satisfy the above property

2) If the relation $f(n) = 6n^2 + 7n + 8$ Prove that

$f(n)$ is not in $\Omega(n^3)$

$$f(n) \geq C(g(n^3))$$

Any

$$1 < \log n < n < n^2 < n^3 < \dots < 2^n < n!$$

given $g(n)$ is higher order Hence it is not possible to be in Ω.

$2n^2 + 4n - 3$

(3) Prove that $2n^2 + 4n - 3 \in \Theta(n^2)$

C_1? C_2?

$$C_1(g(n^2)) \leq f(n) \leq C_2(g(n))$$

h.w

$C_1(g(n^2)) \leq 2n^2 + 4n - 3 \leq C_2(g(n^2))$

$C_1 \cdot 1 \leq 3 \leq C_2 \cdot 1$ ×

$C_1 \cdot n^2 \leq 2n^2 + 4n - 3$

$C_1 \leq 2 + \frac{4}{n} - \frac{3}{n^2}$

$C_1 \cdot \Theta \leq 2 + \frac{4}{n} + \frac{3}{n^2}$

$C_1 - 2 \leq \frac{4}{n} - \frac{3}{n^2}$

$C_1 - 2 \leq 1$

18/4/2024

Notations

Little 'oh' Notation – 'o' $o(g(n))$

The little 'o' Notation (order of $g(n)$) $o(g(n))$ is used to give an upper bound that is not asymptotically typed on a positive run time function $f(n)$ where n is the input size represented as

$$0 \geq f(n) < cg(n), \forall\ n \geq n_0$$

Little ω Notation (Omega) $\omega(g(n))$

The little omega (ω) Notation

$\omega(g(n))$ is used to give lower bound that is not asymptotically typed on a positive runtime function $f(n)$ where n is the input size

$$0 \geq f(n) \geq cg(n) \quad \forall\ n \geq n_0$$

Tilde Notation: (~)

$t(n), g(n)$

It is useful when the functions $t(n)$ & $g(n)$ that grows in a same rate. If the functions $t(n)$ & $g(n)$ have the same growth then we can write it as

$$t(n) \sim g(n)$$

(Part-B) Compulsory – Mergesort – whether it belongs to iterative / recursive

Analysis of recurrance Equation.

- linear
- non linear (sorting, searching)

If algorithm is a recurrence alg.
- this analysis used to solve it

- → Substitution Method (Mathematically you have to prove)
  - Backward substitution
  - Forward substitution
- → Recursive Tree Method (Dynamic diagram)
- → Master Method (certain cases, time complexity will be given compare it with master formula)

recusive
- Merge alg
- Tree alg
- binary alg

(Identify pattern)

1 3 6 9

```
n=1  Print(n)
n1=n1+2  Print(n1)
for(int i=0; i<=size)
{
  n2=n1+3
  Print(n2)
}
```

Iterative Algorithm

Solving recurrance relation.

Let consider merge sort algorithm

(1) Substitution Method

* Let T(n) → Computing Time for the merge sort algorithm for the input size (n)

* It has

Let T(n)

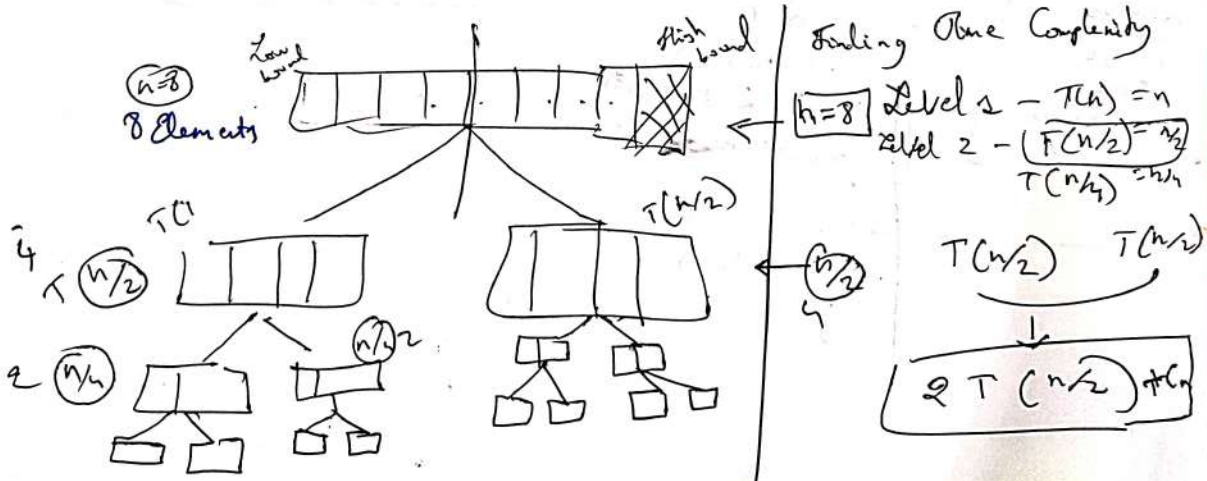
* Divide : Computing the middle task, $\Theta(1)$
* Conquer : Solving 2 Subproblems, task as $2T(n/2)$
* Combine : Mergin 'n' Elements, tasks $\Theta(n)$

Then the recursive relation,

$$T(n) = \begin{cases} a & , n=1 \\ 2T(n/2) + Cn & , n>1 \end{cases}$$

T - Time
C - Cost

BACKWARK SUBSTUTION METHOD



$T(n) = 2T(n/2) + C_n$ — ① $\quad n/2$

$= 2[2T(n/2^2) + C_{n/2}] + C_n \quad n/4$

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

$= 2^2[2T(n/2^3) + C_{n/4}] + 2C_n \quad n/8$ — ②

$= 2^3 T(n/2^3) + 3C_n$ — ③

$$T(n) = 2^i T(n/2^i) + iC_n$$

* NOTE THAT:
The Base Condition

$T(1) = a$

$2^i = n$

$i = \log n$

Time Complexity for Merge & Quick sort

$T(n) = n \log n$

~~Ex-2~~ $T(n) = 2^i T(n/2^i) + iC_n$

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

$= na + \log_n C_n$

∴ Linear with logarithmic so Linear Neglected

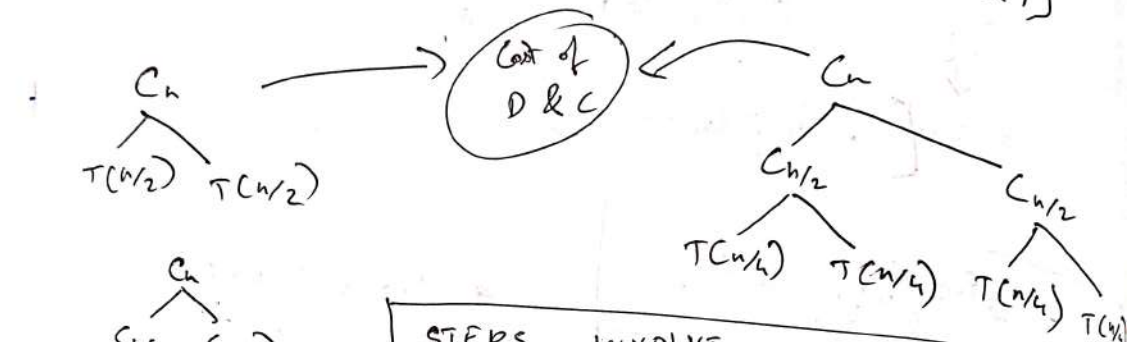
$$T(n) = \Theta(C_n \log n)$$

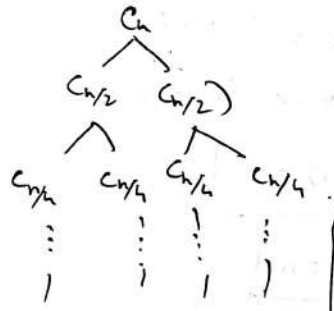
Time Complexity for Merge Sort

(1.W) ① Fibonacci Series

② Tower of Hanoi

DAA RECURSION TREE METHOD [EG: MERGESORT]





STEPS INVOLVED IN RECURSION TREE CONSTRUCTION

- Write down the recurrence as a tree with recursive calls as the children
- Expand the children value.
- Add each level & perform / generate time complexity computation

LEVEL	No. of Problem	Problem Size	Work done
0	1	n	1 x n = n
1	2	n/2	n/2 x 2 x cn
⋮	⋮	⋮	⋮
K	K	n/2^K	n
log(n)	2^log(n)	1	N

To find overall cost of recursion

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

① Solve $t_n = t_{n-1} + 3$, $t_1 = 4$ [By Backward Substitution Method]

$t_n = t_{n-1} + 3, t_1 = 4$ (2) — 1×3

$[t_{n-2} + 3] + 3$ — 2×3

$[t_{n-3} + 3] + 3 + 3$ — 3×3

$[t_{n-4} + 3] + 3 + 3 + 3$

$t_{n-1} + (1)(3)$

$t_{n-2} + (2)(3)$ $\quad t_{n-i} + i3$

i

$t_n = t_{n-i} + i3$ → (1)

Now assume $i = n-1$

$t_n = t_{n-(n-1)} + (n-1)(3)$

$= t_1 + 3n - 3$

$= 4 + 3n - 3$ $\quad t_1 = 4$

$t_n = 3n + 1$

② Solve

$t_n = n t_{n-1}$, $\forall n > 1$ and $t_0 = 1$

$t_n = n t_{n-1}$

i_1 — $= n[(n-1) t_{n-2}]$

i_2 — $= n(n-1)[(n-2) t_{n-3}]$

i_3 — $= n(n-1)(n-2)[(n-3) \cdots t_{n-n}]$

$\vdots$

ith level $\quad i = n-1$

$1 = n - i$

$= n(n-1)(n-2)\ldots(n-i)$

$= n(n-1)(n-2)\ldots 1$

$t(n) = \Omega(n!)$

(3) $t_n = t_{n-1} + 3,\ t_0 = 4$

By forward Substitution (2)

(i) Sub $n=1$

(ii) Apply $t_0 = 4$ to get general for

$t_n = t_{n-1} + 3$

$n = 1$

$t_1 = t_0 + 3$

$= 4 + 3$

$t_1 = 7$

$t_1 = 4 + 3$

$t_2 = t_{2-1} + 3$

$= t_1 = 3$

$t_2 = 10$ $t_2 = 4+3+3$

$t_3 = t_{3-1} + 3$

$= t_2 + 3$

$= 10 + 3$

$t_3 = 13$

$t_3 = 4+3+3+3$

$t_4 = t_{4-1} + 3$

$t_4 = 16$

$t_4 = 4+3+3+3+3$

$= 4, 7, 10, 13, 16, \ldots n$

$t_n = 4 + (i \times 3)$

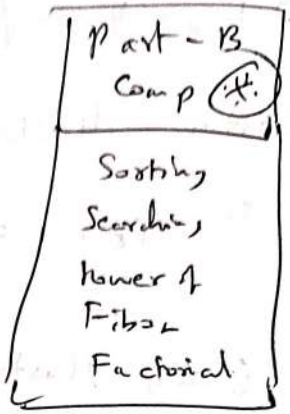
$t_n = 4 + i\,3$

Master theorem (3)

- Solving recursion equation

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

where, $f(n) = \theta(n^k \log^p n)$



* where $a \geq 1$ and $b > 1$ and $f(n)$ is asymptotically positive function there are three cases

Case 1 (i) If $\log_b a > k$ then $\theta(n \log_b^a)$

Case ii

If $\log_b a = k$

(a) $p > -1$, $\theta(n^k \log^{p+1} n)$

(ii) $p = -1$, $\theta(n^k \log \log n)$

(iii) $p < -1$, $\theta(n^k)$

Case (iii) If $\log_b^a < k$ then,

(i) If $p \geq 0$, $\theta(n^k \log^p n)$

(ii) If $p < 0$, $\theta(n^n)$

Solve the following recurrence relation using Master Method:

① $T(n) = 8\,T(n/2) + n\log n$ | $a = 8$, $b = 2$

Using Master Method

$T_n = a\,T(n/b) + f(n)$ — formula

$f(n) = n^1 \log^1$

$n^k \log^p n$ | $k = 1$, $p = 1$

$\log_b^a = \log_2^8$

$= 3\log_2^2 \Rightarrow 3$

$\therefore 3 > 1$

case(ii) $a > k$

$\log_b^a > k$ the $\Theta(n \log_b^a)$

$\Rightarrow \Theta(n \log_2^8)$

$\Rightarrow \Theta(n^3)$ //

H.W

② $T(n) = 8T(n/2) + n$

③ $T(n) = 2T(n/2) + n$

④ $T(n) = 4T(n/2) + n^2 \log n$

⑤ $T(n) = 2T(n/2) + n/\log n$

⑥ $T(n) = 2T(n/2) + n^2 A \log^2 n$

(2) $T(n) = 8T(n/2) + n$

$a = 8 \quad b = 2$

$f(n) = n$

$k = 1, \quad p = 0$

$\Rightarrow \log_b a = \log_2 8 = 2^3$

$= 3 \log_2 2$ $\boxed{\log_2^2 = 1}$

$\therefore 3 > 1$

Belongs to case (i)

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

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

$\therefore T(n) = \Theta(n^3)$

(iii) $T(n) = 2T(n/2) + n \log n$

$a = 4, \quad b = 2$

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

$n = 2 \quad p = 1$

$\log_b a = \log_2 4$

$= 2 \log_2 2 = 2$

$2 > 1$

Belongs to case (i)

$\log_b a > k, \quad \Theta(n^{\log_b a})$

$\Rightarrow \Theta(n^2)$

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

$a = 2, \quad b = 2$

$f(n) = n$

$k = 1, \quad p = 0$

$\log_b a = \log_2 2 = 1$

$\Rightarrow \log_b a = k$

Belongs to case (ii), (1)

$p > -1$ then $\Theta(n^k \log^{p+1} n)$

$\Rightarrow \Theta(n \log n) \rightarrow$ Merge sort

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

$a = 2 \quad b = 2$

$k = 1 \quad p = -1$

$\log_b a = \log_2^2 = 1$

$\log_b a = k$

Belongs to case (ii)

$p = -1 \quad \Theta(n^k \log \log n)$

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

(vi) $T(n) = 2T(n/2) + n^2/\log^2 n$

$a = 2, \; b = 2$

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

$k = 2, \; p = -2$

$\log_b a = \log_2 2 = 1$

$\log_b a < k$

Belongs to case (iii)

$p < 0, \; \Theta(n^k)$

$\Rightarrow \Theta(n^2)$

(vii) $T(n) = 2T(n/2) + n^2 \log^2 n$

$a = 2 \quad b = 2$

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

$k = 2, \; p = 2$

$\log_b a = \log_2 2 = 1$

$\log_b a < k$

Belongs to case (iii)

$p > 0, \; \Theta(n^k \log^p n)$

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

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

$a = 1 \quad b = 2$

$f(n) = n^2$

$k = 2, \; p = 0$

$\log_b a = \log_2 1 = 0$

$\log_b a < k$

Belong to case iii

$p = 0, \; \Theta(n^k \log^p n)$

$\Rightarrow \Theta(n^2)$

Classification of Recursive Equation 25/4/2024

① Homogeneous ② Inhomogeneous

Let $t(n)$ is the time complexity of the algorithm for the size of input n. Let Assume that $t(n)$ is recursively defined as

Homogeneous

Let $T(n) = b_0 T(n) + b_1 T(n-1) + b_2 T(n-2) + \ldots + b_k T(n-k)$

The Constant 'b_i' is connected to 'a_i'

$$a_0 T(n) + a_1 T(n-1) + \ldots + a_k T(n-k) = 0$$

Let us denote $T(i)$ as x^i, then

$$a_0 x^n + a_1 x^{n-1} + \ldots + a_k x^{n-k} = 0$$

This is the Homogeneous equation

Method for Solving recurrance Equations

(i) Write down the characteristic equation
(ii) Solve that Equation
(iii) Write down the general form of the solution
(iv) Determine the polynomial Co-efficient using the initial Condition

STEP 1

- write down the Corresponding characteristic eq.

$$a_0 x^n + a_1 x^{n-1} + \ldots a_k x^{n-k} = 0$$

Remove the Common x terms from the equation

$$x^{n-k}\left[a_0x^k + a_1x^{k-1} + \dots + a_k\right] = 0$$

Characteristic Equation

$$\boxed{a_0x^k + a_1x^{k-1} + \dots a_k = 0}$$

STEP 2: SOLVE The Characteristic equation as a polynomial equation.

Let the roots of $r_1, r_2 \dots r_k$ there are k solution for k order polynomial equation the roots may or may not be same.

STEP 3: The General Solution for the recurrence equation is

Case(i) All the roots are different, then the general solution will be

$$\boxed{T(n) = \sum_{i=1}^{k} c_i r_i^n \quad \text{— (1)}}$$

c_i - Constants

Case(ii) Suppose some of the 'p' roots are equal & remaining is different, the general equation is

k -> total roots

p -> No. of similar roots

$$\boxed{T(n) = \sum_{i=r}^{p} c_i n^{i-1} r_i^n + \sum_{i=p+1}^{k} c_i r_i^n \quad \text{— (2)}}$$

Ex:1 Solve $T(n) - 5T(n-1) + 6T(n-2) = 0$, $n \geq 2$

$t_0 = 0, t_1 = 1$

(-x-)
recurrence eqn with the help of homogeneous with characteristic eqn.

$T(n) = x^n$

$T(n-1) = x^{n-1}$

$\vdots$

$T(n-5) = x^{n-5}$

$\rightarrow x(n) - 5x(n-1) + 6x(n-2) = 0$

removing common x terms

$x^n - 5x^{n-1} + 6x^{n-2} = 0$

$x^{n-2}[x^2 - 5x + 6] = 0$

$x^2 - 5x + 6 = 0$

$x = 3, 2$

$+6$

$-3 \quad -2$

-5

$T(n) = \sum_{i=1}^{k} c_i r_i^n$

Case(i) The roots are different

$T(n) = C_1 3^n + C_2 2^n$

$t_0 = 0$

$= C_1 3^0 + C_2 2^0$

$= C_1 + C_2$

$C_1 + C_2 = 0 \rightarrow (1)$

$t_1 = 1$

$= C_1 3^1 + C_2 2^1$

$= 3C_1 + 2C_2 - (2)$

$T_0 \rightarrow 3C_1 + 3C_2 = 0$

$T_1 \rightarrow 3C_1 + 2C_2 = 1$

$1C_2 = -1$

$C_2 = -1$

$C_1 = 1$

$3C_1 + 3C_2 = 0$

(-1)

$3C_1 + 3 = 0$

$C_1 = +3$

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

$T(n) = 3^n - 2^n$

(9) $(x-2)^3 (x-3) = 0$

$x = 3$

$(x-2)(x-2)(x-2)$

$x = 2$

$x = 2, 2, 2, 3$

Case (ii)

Number of similar roots

$p =$ ~~total~~ roots

$p = 3$

$k =$ total roots

$k = 4$

$T(n) = C_1 2^n + C_2 n 2^n + C_3 n^2 2^n + C_4 3^n$

$T(n) = \sum_{i=1}^{p} C_i n^{i-1} \gamma_i^n + \sum_{i=p+1}^{k} C_i \gamma_i^n$

$i = 1, 2, 3, \; p = 3$

$i = 4$

$T(n) = C_1 n^0 x_1^n + C_2 n^1 2^n + C_3 n^2 2^n + C_4 3^n$

$T(n) = C_1 2^n + C_2 n 2^n + C_3 n^2 2^n + C_4 3^n$ — (*)

In Homogeneous

General Equation

Step 1 $a_0 t_n + a_1 t_{n-1} + \dots a_k t_{n-k} = b_1^n P(n) + b_2^n P_2(n) + \dots$

where, a_i, b_i → Constants

$P_i(n)$ → Polynomial in 'n' of order of 'd_i'

Step 2 :

Corresponding Characteristic Equation

$$(a_0 x^k + a_1 x^{k-1} + \dots a_k)(x-b_1)^{d_1+1}(x-b_2)^{d_2+1} \dots = 0$$

$$a_0 x^k + a_1 x^{k-1} + \dots + a_k = 0$$

$$(x-b_1)^{d_1+1} = 0$$

$$(x-b_2)^{d_2+1} = 0$$

$$\vdots$$

STEP 3 :

Solve the Characteristic Equation as a Polynomial Equation.

* Let the roots are $r_1, r_2 \dots r_k$, there are K solutions for K order polynomial equation.

The roots are may be same or may not be same.

STEP 4 :

The General solution for the Inhomogeneous recurrance equation is

$$P(n) = S_1 + S_2 \dots$$

STEPS : Using initial Condition Solve for the Co-efficient in above equation in order to find a particular Solution

QUESTION

③ $t_n - 5t_{n-1} + 8t_{n-2} - 4t_{n-3} = 0 \quad \forall \quad n > 0$

$[t_0 = 0, t_1 = 1, t_2 = 2]$

④ $t_n - 11t_{n-1} + 30t_{n-2} = 0$ $[t_0 = 0, t_1 = 1]$.

~~⑤ $t_n - 11t_{n-1} + 30t_{n-2}$~~

$x_n - 5x_{n-1} + 8x_{n-2} - 4x_{n-3} = 0$

$\boxed{x^n - 5x^{n-1} + 8x^{n-2} - 4x^{n-3} = 0}$

removing Common x^k terms

$x^{n-3}\left[x^3 - 5x^2 + 8x - 4\right] = 0$

$x^3 - 5x^2 + 8x - 4 = 0$

Using synthetic Division

$-1, 2, 2$

	1	−5	8	−4
1	0	1	−4	4
	1	−4	4	0

$(x-1)(x^2 - 4x + 4) = 0$

$(x-1)(x(x-2) - 2(x-2)) = 0$

$(x-1)(x-2)(x-2) = 0$

$\boxed{x = 1, 2, 2}$ roots

Some roots are equal case (ii)

$T(n) = \sum_{i=1}^{p} C_i n^{i-1} \gamma_i^n + \sum_{i=p+1}^{k} C_i \gamma_i^n$

$p=2 \quad k=3 \quad \gamma_1 \gamma_2 \gamma_3$

$T(n) = n^{0} \gamma_2^n C_1 + C_2 n^{2-1} \gamma_2^n + C_3 1^n$

$T(0) = C_1 + 0 + C_3 = 0 \rightarrow (1)$

$T(1) = 2C_1 + 2C_2 + C_3 = 1 \rightarrow (2)$

$T(2) = 4C_1 + 3C_2 + C_3 = 2 \rightarrow (3)$

$C_1 \neq -C_3$ (from (1))

(2) $\Rightarrow 2C_1 + 2C_2 - C_1 = 1$

$C_1 + 2C_2 = 1$

(3) $4C_1 + 3C_2 - C_1 = 2$

$3C_1 + 8C_2 = 2$

$3C_1 + 6C_2 = 3$

$2C_2 = -1$ $\quad C_2 = -1/2$

$3C_1 + 8(-1/2) = 2$

$3C_1 - 4 = 2$

$3C_1 = 6$

$C_1 = 2 \quad C_2 = -1/2 \quad C_3 = -2$

7) $t_n - 11 t_{n-1} + 30 t_{n-2} = 0 \quad [t_0 = 0, t_1 = 1]$

$x^n - 11x^{n-1} + 30x^{n-2} = 0$

$x^{n-2}[x^2 - 11x + 30] = 0$

30 : $-5, -6$: -11

$x = 5, 6 \quad : \quad \gamma = 5, 6$

$T(n) = C_1 6^n + C_2 5^n$

$T(0) = C_1 6^0 + C_2 5^0$

$0 = C_1 + C_2 \quad -(1)$

$6C_1 + 5C_2 = 1$

$C_1 + C_2 = 0$

$6 \times$

$C_2 = 1$

$t(1) = C_1 6^1 + C_2 5^1$

$1 = 6C_1 + 5C_2 \quad -(2)$

$6C_1 + 5C_2 = 1$

$6C_1 + 6C_2 = 0$

$-1C_2 = 1$

$C_2 = -1$

$C_1 = 1 \quad C_2 = -1$

UNIT - II Quick Sort : [A[0...n-1])

Algorithm:

Problem Description:

This algorithm perform sorting of the elements given in array (A[0...n-1]

I/P: I In array (A[0...n-1]) which element are given. The pointer low indicates left most element & right vice-versa

O/P. Create a Sub-array which is started in ascending order.

If (low < high) then

m ← partition (A[low...high])

Quick (A[low...m-1])

Quick (A[mid+1....high]

Quick (A[low...high])

ALGORITHM

Problem Description:

This algorithm partition the subarray using the first element as pivot element.

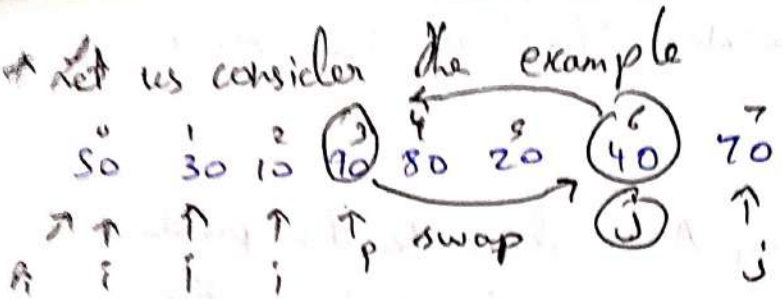
INPUT: A subarray A with pointer 'low' as left most index of the array.

OUTPUT: The partition of array A is done & the pivot occupies its proper position and produce the shorted list

① Algorithm Partition(A[low...high])

```
{
  Pivot ← A[Low]
  i ← low
  j ← high
  while (i≤j) do
  {
    while (A[i]<=pivot) do
      i ← i+1
    while (A[j]>=pivot) do
      j ← j-1
  }
if (i≤j) then
    {
      swap(A[i], A[j])
}
 Swap(A[j], pivot)
return j
 }
```

* Let us consider the example



(i) $30 \leq 50$ ✓ (i) → $i+1$

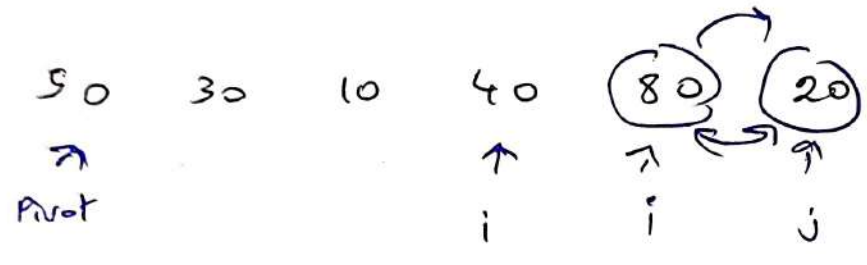
(ii) $10 \leq 50$ ✓ → $i+1$

(iii) $90 \leq 50$ ✗ (x)

(iv) $70 \geq 50$ ✓ (J) → $j-1$

(iv) $40 \geq 50$ ✗

$A[3] \leftrightarrow A[6]$



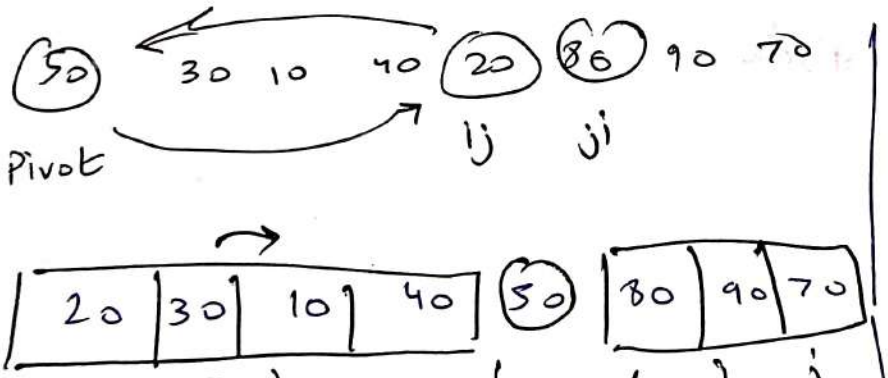
(i) $40 \leq 50$ ✓ → $i+1$

(ii) $80 \leq 50$ ✗

(iii) $90 \geq 50$ ✓ → $j-1$

$20 \geq 50$ ✗

$A[4] \leftrightarrow A[5]$



(i) $20 \leq 50$ ✓ → $i+$

$80 \leq 50$ ✗

(ii) $80 \geq 50$ → $j-$

(iii) $10 \geq 50$ ✗

$20 = 20$ ✓ $i+1$

$30 \leq 20$ ✗

$40 \geq 20$ ✓ $j-$

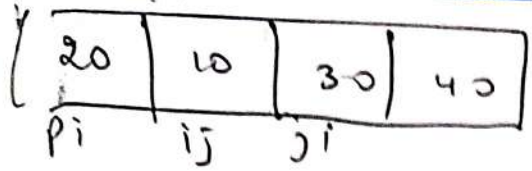
$10 \geq 20$ ✗

$80 = 80$ ✓

$90 < 80$ ✗

$70 > 80$ ✗

$A[i] \leftrightarrow A[j]$



10 < 20 ✓ i+

30 < 20 ✗

30 > 20 ✓ j--

A[i] ⟷ pivot

70 < 80 ✓ i+

90 < 80 ✗

90 > 80 ✓ j-

A[j] ⟷ pivot

70 80 9

10 20 30 40 50

10	20	30	40	50	60	70	80	90

Merge SORT

mergeSort(int A[0...n-1] low, high)

Problem description.

This algorithm is for sorting the element using Merge sort.

INPUT:

Array A of unsorted elements.

Pointer Low & pointer Highest indicates the position of low & highest index position

OUTPUT

Sorted Array A[0...n-1]

① Sorted Array A[0... h-1]

```
If (low < high) then
{
    mid ← (low + high)/2
    Mergesort (A, low, mid)
    Mergesort (A, mid+1, high)
    combine (A, low, mid, high)
}
```

② Algorithm Com (A[0... h-1], low, mid, high) {

```
    k ← low;   // index of temp array
    i ← low;   // left sublist
    j ← mid+1; // right sublist
while (i ≤ mid and j ≤ high) do
{
  if (A[i] < A[j]) then
  {
     temp[k] ← A[i];
      i ← i+1;
      k ← k+1;
  }
  else {
     temp[k] ← A[j];
      j ← j+1;
      k ← k+1;
}}
```

```
while (i ≤ mid) do      // copy remaining elements of sublist to temp
{
   temp[k] ← A[i];
   i ← i+1;
   k ← k+1;
}
while (j ≤ high) do {   // copy remaining elements of right sublist to temp
   temp[k] ← A[j];
   j ← j+1;
   k ← k+1;
}
```

16/5/24 LONGEST COMMON SUBSEQUENCE (LCS) → DP

Algorithm:

I/P: Strings x, y where, $x = x_1, x_2 \cdots x_n$

$y = y_1, y_2 \cdots y_n$

O/P: LCS of strings x, y

```
Begin
    m = length(x)
    n = length(y)
    for j = 0 to length(y) do
        LCS(0,j) = 0
    end for
    for i = 1 to length(x)
    for j = 1 to length(y) do
    if [x[i] ≠ y[i] then
    else
        LCS(i,j) = max{ LCS(i-1,j)
                        LCS(i,j-1) }
        LCS(i,j) = 1 + LCS(i-1,j-1)
    end if
    end for
    end for
    return LCS(m,n)
End.
```

Ex:

① Str1 : largest
Str2 : stone

② S1 : abcd
S2 : ad

① Time Complexity to find LCS using DP: $O(mn)$

$2^k - B - F$

- using recursion in LCS → Dis advantage

DP advantage → using table

MEMORIZATION → store recursive call

Introduction:

- To identify spelling errors in document.
- A subsequence is a subset of characters of the original string which appears in the same orders as those in the original string. And can be obtained by deleting zero/more characters from the sequence & keeping the remaining characters in its original order.
- The common subsequence b/w two strings A & B, is a sequence that is common to both strings.
- LCS is the longest possible subsequence b/w strings A & B.
- Solving LCS using BRUTE Force Method

STEP1: Generate Subsequence of A & B.

STEP2: Find the common subsequence & the longest subsequence & its string.

NOTE The Brute Force Method may be exponential hence partially not used.

Let K is the sum of Length of the two strings then the complexity of this algorithm is order of: $O(2^K)$

- By using Dynamic Programming approach the time complexity is $O(mn)$
- Recursive Formula.

$$LCS(i,j) = \begin{cases} 0, & \text{if } i=0 \mid j=0 \\ \max\{LCS(i,j-1), LCS(i-1,j)\}; & \text{if } i>0, j>0 \ \&\ x_i \neq y_j \\ 1+LCS(i-1,j-1); & \text{if } i>0,\ j>0 \text{ and } x_i=y_i \end{cases}$$

Dynamic Programming

- Construct a table it stores the cost of LCS(i,j) for i=0 to m and j=0 to n.
- Here m,n are length of the strings x,y
- Computation of LCS(i,j) involve two computations:
  (i) Max {LCS(i,j-1), LCS(i-1,j)
  (ii) 1+LCS(i-1,j-1)

⊙ ALGORITHMS

ALGORITHM

A, AL, ALG.....

RIHM

↓

subsequence ✓

Delete in b/w ✓

RITHMA

↓

subseq?

no

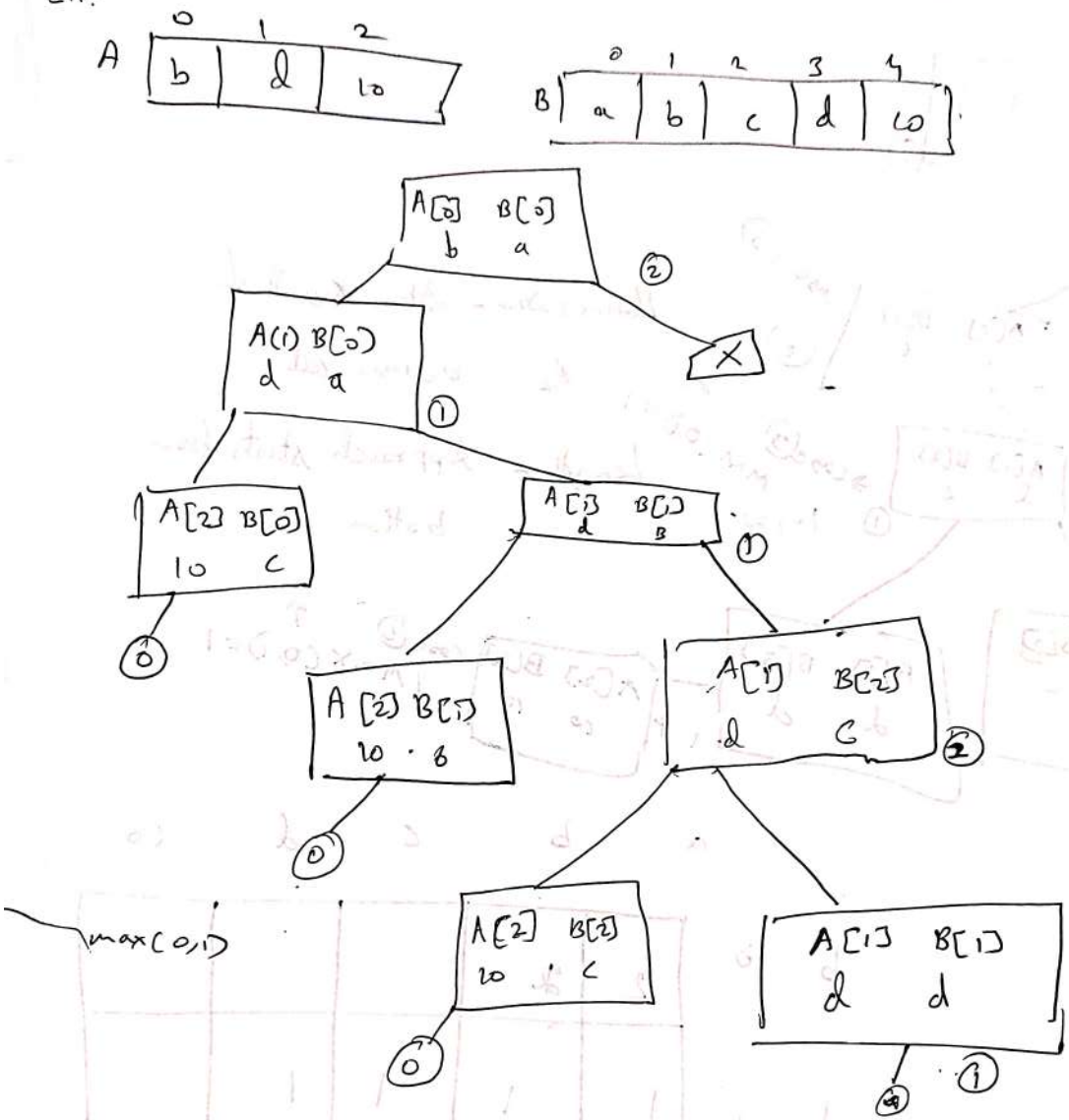
Backtracking ✗

in LCS

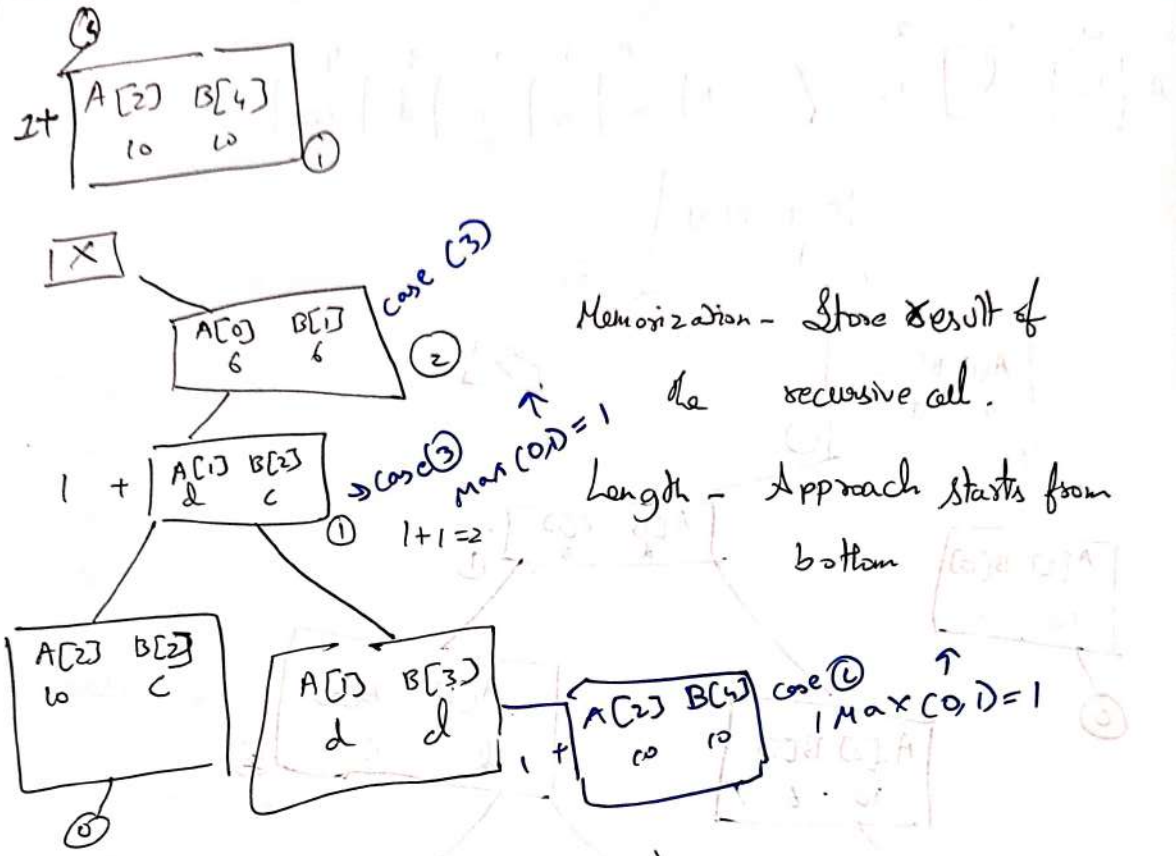
LCS using Memorization.

```
int LCS(i,j)
{
if(A[i]=='\0' || B[j]=='\0')
    return 0;
 else if (A[i]==B[j])
    return 1+LCS(i+1,j+1);
 else
    return max(LCS(i+1,j), LCS(i,j+1));
 }
```

EX:



∴ Length of the Longest Subsequence = 2



Memorization - Store result of the recursive call.

Length - Approach starts from bottom

	a	b	c	d	b
b 0	2	2			
d 1	1	1	1	1	
b 2	0	0	0		0

6/5/2024

Dynamic Programming

Components of DP
① Stages ② Decision
③ State ④ Policy

The given Problem Divided into no. of sub Problems they are always interrelated and overlaping sub-problems.

*To avoid recomputation of multiple overlapping sub problems repeatedly a table was created. whenever a subproblem is solved its solution is stored in a table.

*The Solution of subproblems are combined in a bottom of manner to obtain the final solution

(i) Stages

Number of subproblems are called stages. Its division of problem may take polynomial time it is called as polynomial breakup

(ii) Decision

In every stage decision is taken to acheive optimal solution is called stage decision

(iii) State - It indicates the subproblem for which decision needs to be taken. A variable used to take decision at every stage is called state variable.

(iv) Policy - To find the decision at each stage

Matrix-(CHAIN)-Multiplication (X) PART-B

Lets consides two matrices

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}_{2\times 3} \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{32} \end{bmatrix}_{3\times 2}$$

$A_1 \quad A_2 \quad A_3 \quad A_4$

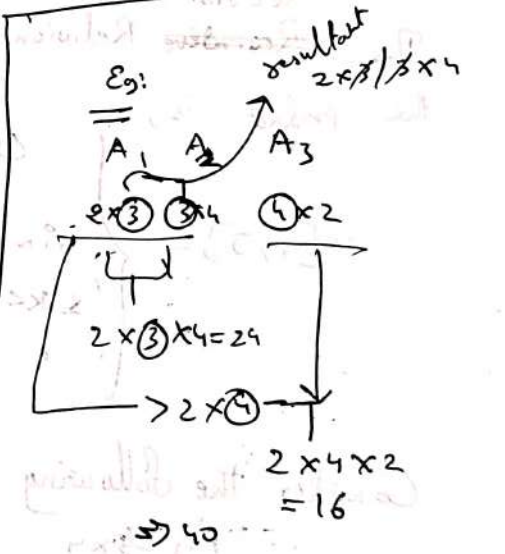
Introduction

* M [i,j] → It is defined only for "$i \le j$", only the portion of table "m", strictly above the main diagonal.

* m [1:n] → It is Required Solution

* s[i,j] → is a value of "k" at which we can divide

Eg:



$(A_1 A_2) A_3 \Rightarrow$ 40 steps

6/5/2024

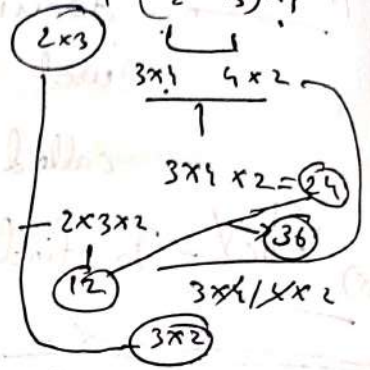
The product → $A_i, A_{i+1} \ldots A_j$

* If $s[i,j]$ equals to value 'k', such that

$$m[i,j] = m[i,k] + m[k+1,j] + p_{i-1} \cdot p_k \cdot p_j$$

Eg:

$A_1 (A_2 A_3) = ?$



36 steps

$m[i,j]$ table: j →

i \ j	1	2	3	4
1				
2				
3				
4				

$s[i,j]$ table:-

i \ j	1	2	3	4
1				
2				
3				
4				

Main purpose → minimum no. of multiplication

Recursive

The ~~Recursive~~ Relation for the minimum cost of parentesizing the product is,

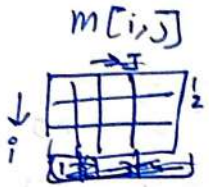
$$m[i,j] = \begin{cases} 0, & \text{if } i=j \\ \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} \cdot p_k \cdot p_j \}, & \text{if } i<j \end{cases}$$

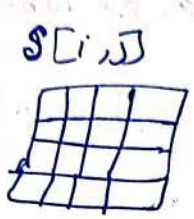
P → Dimension for the Given Matrix

Consider the following matrixes

$A_1 = 5 \times 4$ $A_2 = 4 \times 6$ $A_3 = 6 \times 2$ $A_4 = 2 \times 7$

Point out the optimal order sequence and optimal cost for multiplying these matrices using dynamic programming.



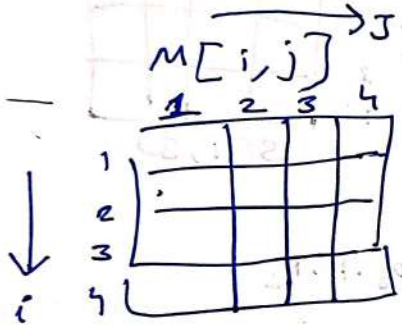


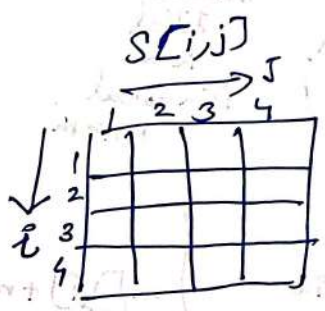
$A_1 = 5 \times 4$ $A_3 = 6 \times 2$

$A_2 = 4 \times 6$ $A_4 = 2 \times 7$

10/5/24

* Let the Matrix 'A_i' as Dimension $P_{i-1} * P_i$





STEP 1

* $A_1 = 5 \times 4 \Rightarrow P_{i-1} * P_i \Rightarrow P_0 * P_1$

$P_0 = 5$ $P_1 = 4$

* $A_2 = 4 \times 6 \Rightarrow P_1 * P_2$

$P_1 = 4$ $P_2 = 6$

* $A_3 = 6 \times 2 \Rightarrow P_2 * P_3$

$P_2 = 6$ $P_3 = 2$

* $A_4 = 2 \times 7 \Rightarrow P_3 * P_4$

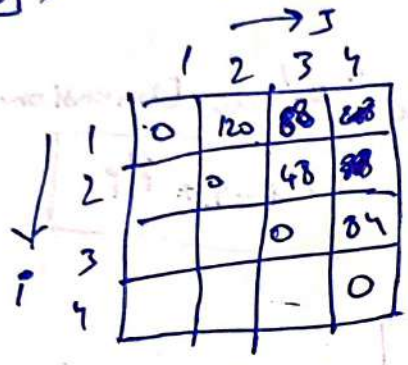
$P_3 = 2$ $P_4 = 7$

Matrix Chain Multiplica

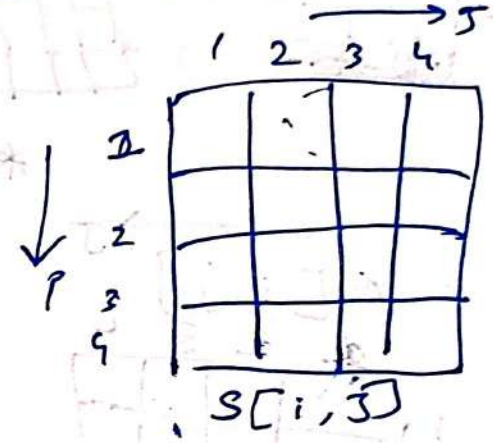
Step 2:

Construct the values for M & S table.

$i=j \Rightarrow m_1[1,1] = m_2[2,2] = m_3[3,3] = m[4,4] \Rightarrow 0$



values should filled in a Diagonal Pattern



$$m[1,2] = \min_{1 \le k < 2} \{ m[1,1] + m[2,2] + P_0 \cdot P_1 \cdot P_2 \}$$

$k = 1$

$\Rightarrow \min\{0 + 0 + 5*4*6\} \Rightarrow 120$

$m[1,2] = 120$

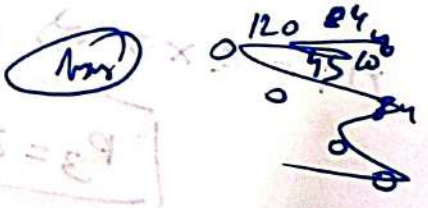
Minimum Cost

$i \le k < j$

$m[2,3] \Rightarrow \min\{m[2,2] + m[3,3] + P_1 \cdot P_2 \cdot P_3\}$

$= 0 + 0 + 4*6*2$

$m[2,3] = 48$



$m[3,4] \Rightarrow \min\{m[3,3] + m[4,4] + p_{3-1} * p_3 * p_4\}$

$\Rightarrow 0 + 0 + p_2 * p_3 * p_4$

$\Rightarrow 6 * 2 * 7$

$\Rightarrow 84$ $\quad \boxed{m[3,4] = 84}$

$m[1,3] \Rightarrow \min\{m[1,1] + m[3,3] + p_0 * p_1 * p_3\}$
$\quad i\ j \qquad [1,k] \quad [k+1,j] \quad i \le k < j$
$\quad 1 \quad (1,2) \quad 3$

$= \min\{0 + 48 + 5 * 4 * 2\}$

$= \min\{48 + 40\}$

$\boxed{m[1,3] = 88}$ $\quad \boxed{k = 1,2}$

$\boxed{k=1}$

$m[2,4] \Rightarrow \min\{m[2,2] + m[3,4] + 4 * 6 * 7\}$

$\Rightarrow \{0 + 84 + 4*6*7\}$

$= 84 + 24 + 7$

$= 84 + 168$

$= 252$

$m[1,4] = 258$

0	120	88	1258
	0	48	104
		0	84
			0

$m(i,j)$

	1	1	3
		2	3
			3

$s(i,j)$

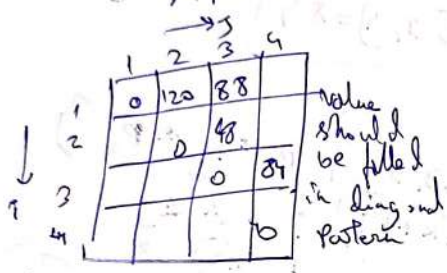
Matrix Multiplication Operation

$P_0 = 5$ $P_1 = 4$ $P_2 = 6$ $P_3 = 2$ $P_4 = 7$

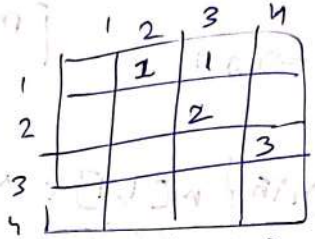
Steps: Construct table for M & S table

$i = j \Rightarrow m[1,1] = m[2,2] = m[3,3] = m[4,4] \Rightarrow 0$

$A_1 = 5 \times 4$ $A_2 = 4 \times 6$ $A_3 = 6 \times 2$ $A_4 = 2 \times 7$



m(i,j)



S(i,j)

(k values will be entered)

$m[1,2] = m[1,k] + [k+1,2] + P_{i-1} * P_k * P_j \Rightarrow 0 + 0 + 5*4*6$

$k = 1$ (1) (2) (0) (1) (2)

$120 = m[1,2]$

$m[2,3] = [m[2,2] + [3,2] + P_1 * P_2 * P_3 \Rightarrow 4*6*2$

$k = 2$

$m[2,3] = 48$

$m[3,4] = m[3,3] + [4,4] + P_2 * P_3 * P_4 \Rightarrow 6 \times 2 \times 7$

$k = 3$

$m[3,4] = 84$

$m[1,3] \Rightarrow m[2,2] + m[2,3] + P_0 * P_1 * P_3 \Rightarrow 5 \times 4 \times 2$ ($20 < 2$)

$k = 1$

$\Rightarrow 48 + 40$

$m[1,3] = 88$ Min Cost

$m[1,3] \Rightarrow m[2,2] + m[3,3] + P_0 \times P_2 \times P_3 \Rightarrow 120 + 48$

$k = 2$

$m[1,3] = 168$

$m[2,4] \Rightarrow m[2,2] + m[3,4] + P_1 \times P_2 \times P_4$

$k = 2$

$\Rightarrow 0 + 84 + 4 \times 6 \times 7$

$24 \quad 168$

168 + 84 = 252

148 + 84 = 232

224

$25^2 11$

$m[2,4]_{k=3} \Rightarrow m[2,3] + [m[4,4]] + p_2 \times p_3 \times p_4$

$\Rightarrow 48 + 0 + 6 \times 2 \times 7$

$\Rightarrow 1$

$m[1,4]$ $k=3$ $= m[1,3] + m[4,4] + p_2 \times p_3 \times p_4$

$\Rightarrow 88 + 0 + 84$

Ex: $A_1 = 5 \times 4$ $A_2 = 4 \times 6$ $A_3 = 6$

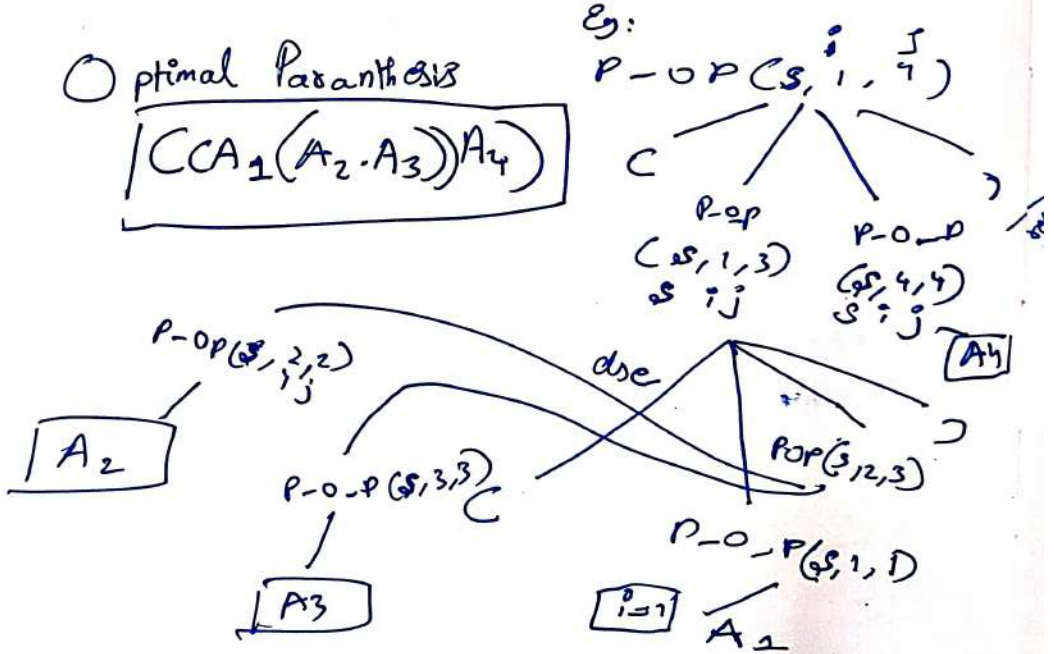
Initial Call to print-Optimal-Parenthesis Algorithm is (s, i, n)

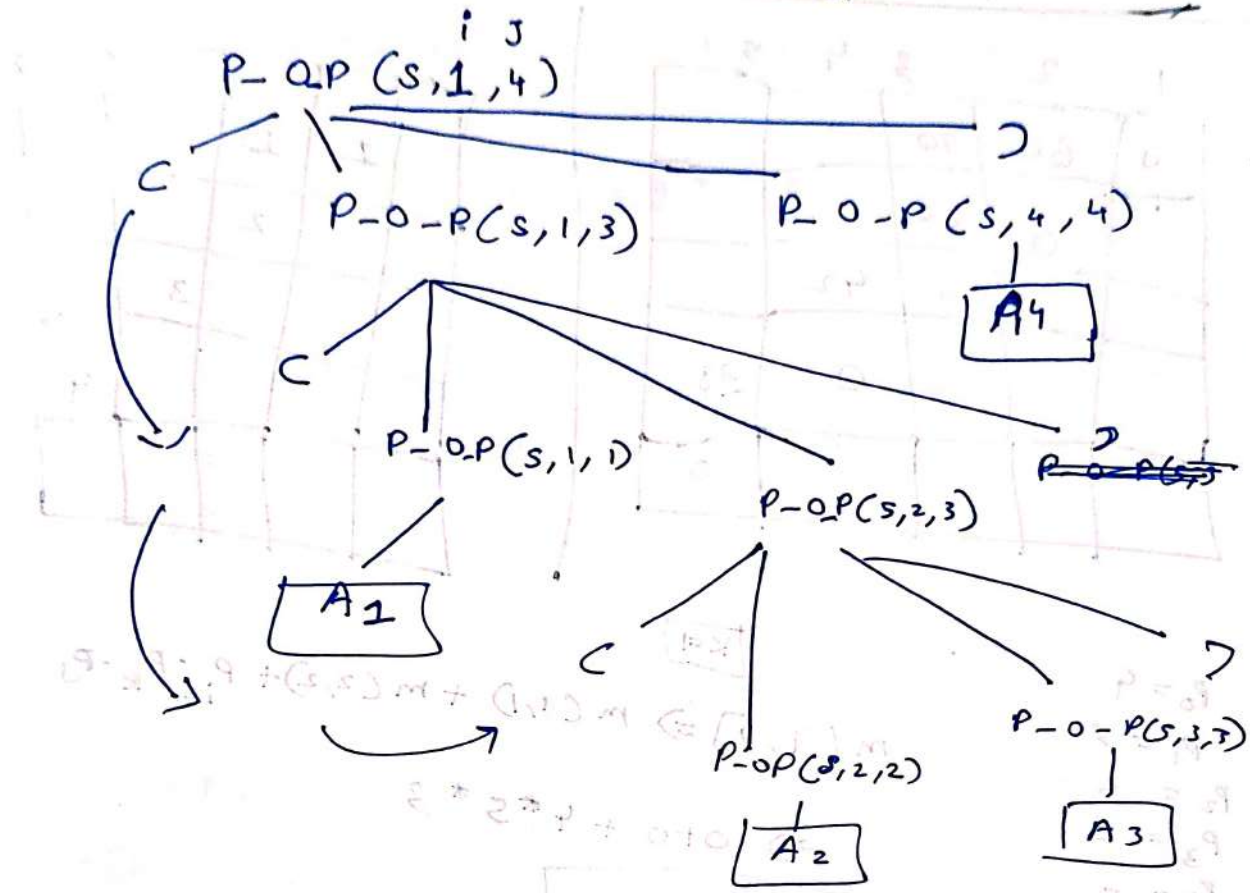
```
Print_optimal_parenthesis (s, i, j)
{
  If i = j then part "Ai"
  else
  print "(";
  Print_Optimal_parenthesis (s, i, s[i, j]);
  Print_Optimal_parenthesis (s, s[i, j]+1, j);
  print ")";
}
```

i - start value
j - end value
n - size

Optimal Parenthesis

$((A_1(A_2 \cdot A_3))A_4)$





Eg A(4×5) B(5×3) C(3×2) D(2×7) E(7×2)

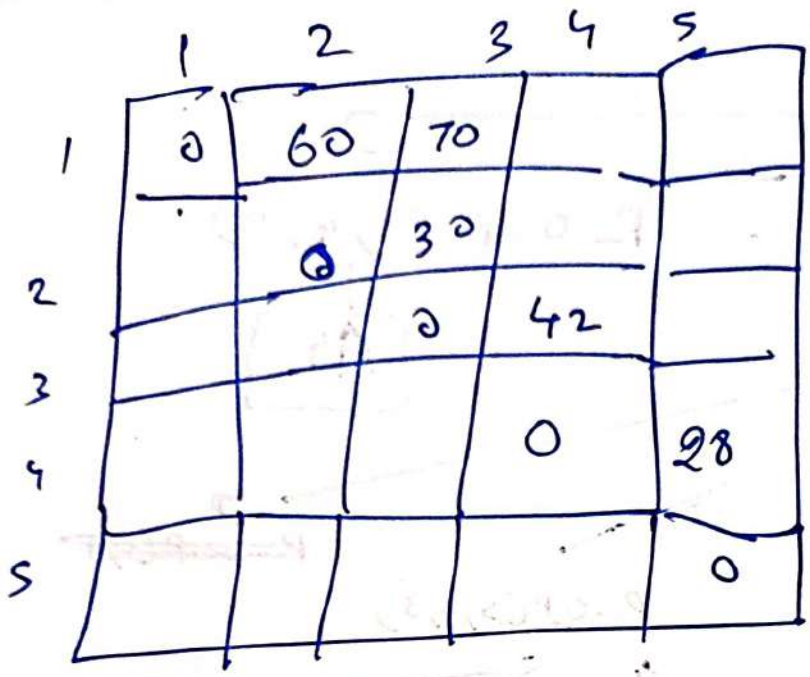
~~$A = P_1 = P_i$~~

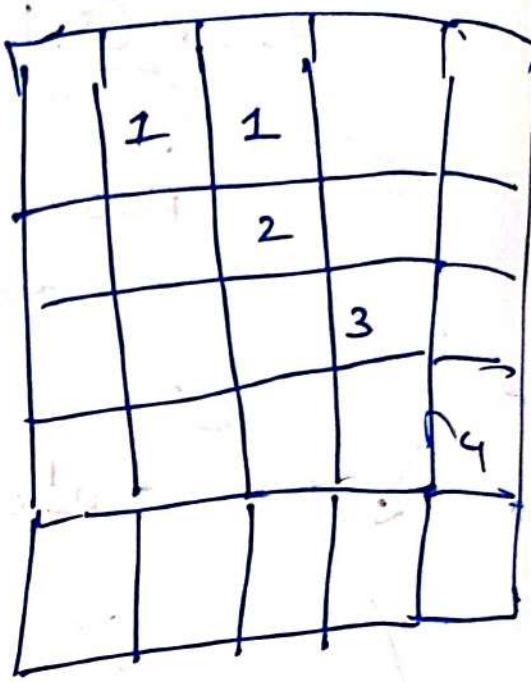
$\Rightarrow P \quad A_1 = P_{i-1} * P_i$

$= P_0 * P_1$

$P_0 \Rightarrow 4$ $P_1 = 5$ $P_2 = 3$ $P_3 = 2$

$P_4 = 7$ $P_5 = 2$





$P_0 = 4$
$P_1 = 5$
$P_2 = 3$
$P_3 = 2$
$P_4 = 7$
$P_5 = 2$

K=1

$m[1,2] \Rightarrow m(1,1) + m(2,2) + P_{i-1} \cdot P_k \cdot P_j$

$\Rightarrow 0+0+4*5*3$

$m[1,2] \Rightarrow 60$

$m[2,3] = m[2,2) + m(3,3) + P_1 \cdot P_2 \cdot P_3$ K=2

$= 0+0+5*3*2$

$m[2,3] = 30$

K=3

$m[3,4] \Rightarrow m(3,3) + m(4,4) + P_2 * P_3 * P_4$

$\Rightarrow 0+0+3*2*7$

$m[3,4] \Rightarrow 42$

$m[4,5] \Rightarrow m[4,4] + m[5,5] + p_3 * p_4 * p_5$

$k=4$

$\Rightarrow 0 + 0 + 2 * 7 * 2$

$m[4,5] \Rightarrow 28$

$m[1,3] \Rightarrow m[1,1] + m[2,3] + p_0 * p_1 * p_3$

$k=1,2$ $k=1$

$\Rightarrow 0 + 30 + 4 * 5 * 2$

$m[1,3] = 70$ min. cost

$k=2$

$\Rightarrow m[1,2] + m[3,3] + p_0 * p_2 * p_3$

$m[1,3]$ $k=2$

$\Rightarrow 60 + 0 + 4 * 3 * 2$

$= 24 + 60$

$m[1,3] = 84$

$k=2$

$m[2,4] \Rightarrow m[2,2] + m[3,4] + p_{2-1} * p_2 * p_4$

$k=2,3$

Algo - dynamic programing - Chain mul (P, n)

I/p // P is order, and 'n' is the chain of matrices

O/P // Min Cost of Matrix chain mul

```
Begin
for i=1 to n do
    m[i,j] = 0.
end for
    for diagonal = 1 to n-1
        j = i + diagonal
            m[i,j] = ∞
    for k=1 to j-1 do
        if m[i,j] > m[i,k] + m[k+1,j] + P_{i-1} * P_k * P_j
        then m[i,j] = m[i,k] + m[k+1,j] + P_{i-1} * P_k * P_j
            R[i,j] = k
        else
            m[i,j] = m[i,j]
            R[i,j] = k
        end if
```

Time Comp = $\Theta(n^3)$

end for
end for
end for
return c[1, n]
End

23/5/24

LCS Using Dynamic Programming

Recursive Relation

$$LCS(i,j) = \begin{cases} 0, & \text{if } i=0, j=0 \\ \max\{LCS(i,j-1), LCS(i-1,j)\}, & x_i \neq y_i \\ 1+LCS(i-1,j-1), & x_i = y_i \end{cases}$$

Solve the find LCS for the given String

Str1 = SAVANNAH STR2 = HAVANA

ε - Epsilon (nearest too)

	⊗	0	1	2	3	4	5		6	7
		ε	S	A	V	A	N	N	(A)	H
0	ε	0	0	0	0	0	0	0	0	0
1	H	0	0	0	0	0	0	0	0	1
2	A	0	0	1	1	1	1	1	1	1
3	V	0	0	1	2	2	2	2	2	2
4	A	0	0	1	2	3	3	3	3	3
5	N	0	0	1	2	3	4	4	4	4
6	A	0	0	1	2	3	4	4	5	(5)

A V A N A

$LCS(0,j)=0$ / if $(x[i] \neq y[j])$

$LCS(i,0)=0$ / if $H \neq E$ | $LCS(1,0), LCS(0,1)$

$\max\{LCS(i,j-1), LCS(i-1,j)\}$

$\max\{0,0\}$ $C(1,1)=0$

(1,2) max{LCS(1,0), LCS(0,1)}

max{0,0}

maximum value = 0

(i) (1,2)

S = A

max{LCS(1,1), LCS(0,2)}

↓ 0 ↓ 0

(iii) (1,3) (S≠V)

(iv) (1,4)

(i) (2,2) i≠j

(i,j) (2,1) (1,2)

Eg: 2 str1: LONGEST str2: Stone

		0 Ø	1 L	2 O	3 N	4 G	5 E	6 S	7 T
0	Ø	0	0	0	0	0	0	0	0
1	S	0	0	0	0	0	0	1	01
2	T	0	0	0	0	0	0	1	2
3	O	0	0	1	1	1	1	1	2
4	N	0	0	1	2	2	2	2	2
5	E	0	0	1	2	2	3	3	3

(1,1) = (0,1) (1,0)

(2,1) = (1,1) (2,0)

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

(2,6) = (1,6) (2,5)

(2,5) = (1,5) (2,4)

(3,7) ⇒ (3,0) (2,7)

(1,7) (0,7) (1,6)

↓ 0

(3,1) ⇒ (2,1) / (3,0)

(3,2) (2,1)

(3,3) → (3,2) (2,3)

(3,4) → (3,3) (2,4)

(3,5) → (3,4) (2,5)

(3,6) → (3,5) (2,6)

GREEDY technique

Greedy Method Selects and import at each stage which derives a feasible solution then it is added to the optimal solution until the problem terminates with a condition.

If the greedy method fails, to produce optimal solution then they are considered under the NP class (Non Deterministic Polynomial Factor Problem)

Determine → find solution for problem

non-Determine → may or may not produce Result

Feasible Solution:

Any Subset that satisfies some constraints is called feasible solution.

Optimal Solution

We need to find the final feasible solution that either maximize or minimize the given objective function.

The feasible solution can produce this objective function is called as optimal solution.

Every Prob has its own objective function

Unit-3

Knapsack Problem

Algorithm Assuming the objects already sorted into non-increasing order of p_i/w_i

```
Void GreedyKnapSack(float m, int n)
// p[1:n] and w[1:n] - profit & weight
// p[i]/w[i] >= p[i+1]/w[i+1], 'm' is the knapsack
// size and x[1:n] is the solution vector
{
  for (int i=1; i<=n; i++)
      x[i] = 0.0;
      float U = m;
  for (i=1; i<=n; i++) {
      if (w[i] > U)
        break;
      x[i] = 1.0;
    U = w[i]
  }
  if (i<=n)
    x[i] = U/w[i];
}
```

Knapsack Problem.

Maximize $\sum_{1 \le i \le n} p_i x_i$ — ①

P - Profit
W - Weight

Subject to $\sum_{1 \le i \le n} w_i x_i$ — ② and $0 \le x_i \le 1, 1 \le i \le n$

Greedy Strategy:- ① It use total Profit as Optimization

(ii) Weight as optimization.

(iii) Ratio of Profit to weight (p_i / w_i) as Optimization function.

Consider the following Instance of the knapsack Problem:

$n = 3, m = 20, (P_1, P_2, P_3) = (25, 24, 15)$

$(w_1, w_2, w_3) = (18, 15, 10)$

The i/p values $(x_1, x_2, x_3) = \{(1/2, 1/3, 1/4), (1, 2/15, 0), (0, 2/3, 1), (0, 1, 1/2)\}$

Sol

Considering First input Value

$$\sum_{1 \le i \le n} (p_i x_i) = \left\{ \frac{25}{2} + \frac{8}{24} + \frac{15}{4} \right\}$$

$$= \left\{ \frac{50 + 32 + 15}{4} \right\} = \frac{97}{4}$$

$$= 24.25$$

$$\sum (w_i x_i) = \frac{18}{2}^{9} + \frac{15}{3}^{5} + \frac{10}{4}^{2.5}$$

$1 \le i \le n$

$$= 16.5 //$$

$$\sum (p_i x_i) = 25 + 24 \times 2/5 + 0 \times 15) \quad (2)$$

$$= 28.69$$

$$\sum (w_i x_i) = 18 + 15 \times 2/5 + \frac{1}{4} \times 15$$

$$= 20.00$$

$$\sum (p_i x_i) = 25 \times 0 + 24 \times 2/3 + 15 \times 1 = 31 \quad (3)$$

$$\sum (w_i x_i) = 18 \times 0 + 15 \times 2/3 + 10 \times 1 = 20$$

$$\sum (p_i x_i) = 25 \times 0 + 24 \times 1 + 15 \times 1/2 = 31.5 \quad (4)$$

$$\sum (w_i x_i) = 18 \times 0 + 15 \times 1 + 10 \times 1/2 = 20$$

Feasible solution

S.No	i/p	$\sum p_i x_i$	$\sum w_i x_i$
1	$(1/2, 1/3, 1/4)$	24.33	16.5
2	$(1, 2/15, 0)$	28.2	20
3	$(0, 2/3, 1)$	31	20
4	$(0, 1, 1/2)$	31.5	20

∴ Fourth is the optimal Solution

Naive String Matching Algorithm

2nd comes Under Unit

```
Algo(T,P)
n= T.length (total length of text value)
m= p.length (Total Number of Pattern)
for S=0 to n-m
  if (P[1...m] = T[S+1... S+m])
      Print "Pattern Found with shift", S
```

* It is a simplest method for Pattern Matching, it performs checking Operation or check all the position in the given text whether an occurance of the pattern available or not.
* After each iteration of an algorithm shift the pattern by exactly one position to righthand side.
* Let P is a subset of 'T' text

'P' -> subset | 'T' - Text

* If P is found in 'T', pattern 'P' occurs with shift S in capital "T"

Ex1: Let consider, Text (T) = a c a a b c,
Pattern (P) = a a b

Sol: $n=6$ $m=3$ $n-m=3$

T	a	c	a	a	b	c	No. of Comparisons	
shift 0	a ✓	a ✗	b				2	No result
shift 1		a ✗	a	b			1	No result
shift 2			a ✓	a ✓	b ✓		3	Shift 2
shift 3				a ✓	a ✗	b ✗	2	No result

Remark

Total Number of Comparison = 8

Ex:2 T = 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1

P = 0 0 0 1 $n = 15$ $m = 4$ Shift $n-m = 11$

	0	0	0	0	1	0	0	0	1	0	1	0	0	0	1	No. of Comparisons	Result
t	0 ✓	0 ✓	0 ✓	1 ✗												4	No result
		0	0	0	1											4	shift 1
t1			0 ✓	0 ✓	0 ✗	1										3	No result
t2				0 ✓	0 ✗	0	1									2	No result
t3					0 ✗	0	0	1								1	No result
t4						0	0	0	1							4	shift 5
5							0 ✓	0 ✓	0 ✗	1						3	No result
6								0 ✓	0 ✗	0	1					2	No result
7									0 ✗	0	0	1				1	No result
8										0 ✓	0 ✗	0	1			2	No result
9										a	0	0	0	1		1	No result
10											0	0	0	1		4	shift 10
11																	

No. of Comparisons = 31

Ex.3 $n=10$ $m=3$ $n-m=7$

T = A B C A B A B C D D

P = B A B

	A	B	C	A	B	A	B	C	D	D	No. of Comparison	Result
shift 0	B	A	B								1	No result
shift 1		B	A	B							2	No result
shift 2			B	A	B						1	No result
shift 3				B	A	B					1	No result
shift 4					B	A	B				3	shift 4
shift 5						B	A	B			1	No result
shift 6							B	A	B		2	No result
shift 7								B	A	B	01	No result

No. of Comparisons 12.

Knuth – Morris – Pratt Algorithm (KMP Algorithm)

Step 1 ① Take two variables i & j

i = string (T(i), j = P[0])

② Compare T(i) with P(j+i).

(i) If match is found

(Move both 'i' and 'j' to R.H.S)

(ii) If mismatch (Move 'j' to the location as per the "TT table" index value)

(iii) if j = 0, then move 'i' to R.H.S

finding π Table / Longest Proper Prefix (LPP)

	0	1	2	3	Index
P1:	a	b	a	b	Prefix

P2: a b c d a b c y

P1

0	1	2	3	Prefix
a	b	a	b	
0	0	1	2	

P2

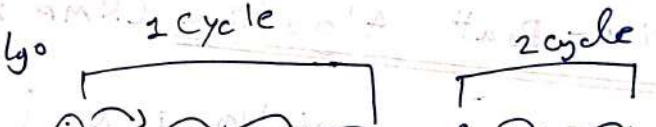
0	1	2	3	4	5	6	7
a	b	c	d	a	b	c	y
0	0	0	0	1	2	3	0

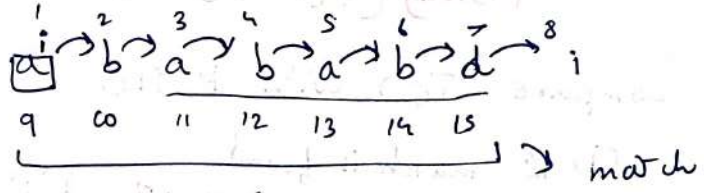
P3:

1	2	3	4	5	6	7	8	9	10	11
a	b	c	d	e	a	b	f	a	b	c
0	0	0	0	0	1	2	0	1	2	3

11/5/24 KMP Algo

Ex1 Text: a b a b c a b c





Pattern a b a b d

π table:

	a	b	a	b	d
j=0	0	0	1	2	0

Compare T(i) & P(j+1)

i, j → → →

⇒ Match – move i, j to R.H.S

Cycle 1:

$T(i) = P(j+1)$

$T(1) = P(1)$

$T(2) = P(2)$

$T(3) = P(3)$

$T(4) = P(4)$

$T(5) \neq P(5)$

mismatch – move i as per π table

$T(5) \neq P(3)$

j = 0, move i to R.H.S

Cycle 2:

$T(6) = P(1)$

$T(7) = P(2)$

$T(8) \neq P(3)$

j = 0 → move i to R.H.S (As per π table)

Cycle 3:

$T(9) = P(1)$

$T(10) = P(2)$

$T(11) = P(3)$

$T(12) = P(4)$

$T(13) \neq P(5)$

j = 2 (As per π table) (j+1)

$T(13) = P(3)$

$T(14) = P(4)$ $T(15) = P(5)$

Eg: 2 | T = EXAMPLE (i: 1 2 3 4 5 6 7), P = AMP

$T(1) \neq P(1)$

j = 0 (π)

$T(2) \neq P(1)$

j = 0 (π)

$T(3) = P(1)$

$T(4) = P(2)$

$T(5) = P(3)$

∴ There is no further j+1

π table, j = 0

	A	M	P
	1	2	3
	0	0	0

TRANSFORM & CONQUER

objective:

→ It is an algorithm design technique that Solve the given problem from transforming problem into another problem

→ It have three technique:

i) Instance simplification
ii) Representation change
iii) Problem reduction

(i) INSTANCE SIMPLIFICATION

The given problem is transformed into same problem of simpler or convenient instance

(ii) REPRESENTATION CHANGE:

It involves the transformation of instance to a different representation without affecting the instance

(iii) PROBLEM REDUCTION:

It involves the transformation of a problem to another type of problem for which an algorithm already exist.

LU Decomposition / LUP / method of Triangulization: [lower Δ^L, upper Δ^U]

Eg: Lets consider the system of Equation & perform LU Decomposition:

$x+2y+3z=14$

$2x+3y+4z=20$

$3x+4y+z=14$

Sol: The equ can be written in the form of $AX=B$

$$\begin{bmatrix}1&2&3\\2&3&4\\3&4&1\end{bmatrix}\begin{bmatrix}x\\y\\z\end{bmatrix}=\begin{bmatrix}14\\20\\14\end{bmatrix}$$

Let consider $LU=A$

$$\begin{bmatrix}1&0&0\\l_{21}&1&0\\l_{31}&l_{32}&1\end{bmatrix}\begin{bmatrix}u_{11}&u_{12}&u_{13}\\0&u_{22}&u_{23}\\0&0&u_{33}\end{bmatrix}=\begin{bmatrix}1&2&3\\2&3&4\\3&4&1\end{bmatrix}$$

$$LU=\begin{bmatrix}u_{11}+0+0 & u_{12}+0+0 & u_{13}+0+0\\ l_{21}u_{11}+0+0 & l_{21}u_{12}+u_{22} & l_{21}u_{13}+u_{23}\\ l_{31}u_{11}+0 & l_{31}u_{12}+l_{32}u_{22} & l_{31}u_{13}+l_{32}u_{23}+u_{33}\end{bmatrix}$$

$\therefore LU=A$

$u_{11}=1$, $u_{12}=2$, $u_{13}=3$

* $l_{21}u_{11}=2$

$l_{21}=2/u_{11}$

* $l_{21}u_{12}+u_{22}=3$

$2(2)+u_{22}=3$

$u_{22}=-1$

* $l_{21}u_{13}+u_{23}=4$

$2(3)+u_{23}=4$

$u_{23}=-2$

* $l_{31}u_{11}=3$

$l_{31}=3$

* $l_{31}u_{12}+l_{32}u_{22}=4$

$3(2)+l_{32}(-1)=4$

$l_{32}=2$

* $l_{31}u_{13}+l_{31}u_{23}+u_{33}=1$

$3(3)+(2)-2+u_{33}=1$

$u_{33}=-4$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{pmatrix} \qquad U = \begin{pmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ 0 & 0 & -4 \end{pmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ 0 & 0 & -4 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 1 \end{bmatrix}$$

① $\Rightarrow LUX = B$

Consider $UX = Y$ Thus, $LY = B$

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 14 \\ 20 \\ 14 \end{bmatrix}$$

$$\begin{bmatrix} y_1 + 0 + 0 \\ 2y_1 + y_2 + 0 \\ 3y_1 + 2y_2 + y_3 \end{bmatrix} = \begin{bmatrix} 14 \\ 20 \\ 14 \end{bmatrix}$$

$\boxed{y_1 = 14}$

$* 2y_1 + y_2 = 20$

$\boxed{y_2 = -8}$

$* 3y_1 + 2y_2 + y_3 = 14$

$42 + (-16) + y_3 = 14$

$\boxed{y_3 = -12}$

$UX = Y$

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ 0 & 0 & -4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 14 \\ -8 \\ -12 \end{bmatrix}$$

$* x + 2y + 3z = 14$

$-y - 2z = -8$

$y + 2z = 8$ $\boxed{z = 3}$

$* y + 2z = 8$ $\boxed{y = 2}$

$* x + 2(2) + 3(3) = 14$

$\boxed{x = 1}$

Verify:

$x + 2y + 3z = 14$

$1 + (2) + (3)(3) = 14$

$14 = 14$ //

13/6/24 Gaussian Elimination Method

1) $x+2y+3z=6$

$2x+4y+z=7$

$3x+2y+9z=14$

$$\left[\begin{array}{ccc|c} 1 & 2 & 3 & 6 \\ 2 & 4 & 1 & 7 \\ 3 & 2 & 9 & 14 \end{array}\right] \begin{matrix} R_1 \\ R_2 \\ R_3 \end{matrix}$$

Updated Matrix

$$\left[\begin{array}{ccc|c} 1 & 2 & 3 & 6 \\ 0 & 0 & -5 & -5 \\ 3 & 2 & 9 & 14 \end{array}\right] \begin{matrix} R_1 \\ R_2 \\ R_3 \end{matrix}$$

$2R_1 \rightarrow$ 2 4 6 12

$R_2 \rightarrow R_2 - 2R_1 \rightarrow$ 2 4 1 7

2 4 6 12

0 0 −5 −5

$3R_1 \rightarrow$ 3 6 9 18

$-R_3$ −3 −2 −9 −14

0 4 0 4

$R_3 \rightarrow 3R_1 - R_3$

$R_3 \leftrightarrow R_2$

Updated Matrix

$$\left[\begin{array}{ccc|c} 1 & 2 & 3 & 6 \\ 0 & 4 & 0 & 4 \\ 0 & 0 & -5 & -5 \end{array}\right]$$

$-5z = -5$

$\boxed{z=1}$

$4y = 4$

$\boxed{y=1}$

$x+2y+3z=6$

$x+2+3=6$

$\boxed{x=1}$

(9) $2x+2y+z=12$

$3x+2y+2z=8$

$5x+10y-8z=10$

$$(A,b)=\left[\begin{array}{ccc|c} 2 & 2 & 1 & 12 \\ 3 & 2 & 2 & 8 \\ 5 & 10 & -8 & 10 \end{array}\right]$$

$R_2 \to 2R_2 - 3R_1$

$6 \quad 4 \quad 4 \quad 16$

$-6 \quad -6 \quad -3 \quad -36$

$0 \quad -2 \quad 1 \quad -20$

$$\left[\begin{array}{ccc|c} 2 & 2 & 1 & 12 \\ 0 & -2 & 1 & -20 \\ 5 & 10 & -8 & 10 \end{array}\right]$$

$R_3 \to 2R_3 - 5R_1$

$10 \quad 20 \quad -16 \quad 20$

$-10 \quad -10 \quad -5 \quad -60$

$0 \quad 10 \quad -21 \quad -40$

$$\left[\begin{array}{ccc|c} 2 & 2 & 1 & 12 \\ 0 & -2 & 1 & -20 \\ 0 & 10 & -21 & -40 \end{array}\right]$$

$R_2 \to R_3 + 5R_2$

$0 \quad 10 \quad -21 \quad -40$

$0 \quad -10 \quad 5 \quad -100$

$0 \quad 0 \quad -16 \quad -140$

$$\left[\begin{array}{ccc|c} 0 & 2 & 1 & 12 \\ 0 & -2 & 1 & -20 \\ 0 & 0 & -16 & -140 \end{array}\right]$$

$-16z = -140$

$\boxed{z = 8.75}$

$-2y + z = -20$

$2y = 20 + z$

$\boxed{y = 14.375}$

$2x + 2y + z = 12$

$2x + 2(14.375) + 8.75 = 12$

$\boxed{x = 12.75}$

Back tracking technique

It is a most general techique to search set of solution, which satisfies some constraints.

Solution Space It includes set of all possible feasible solution for a given problem.

STATE Space Tree In backtracking technique while solving a given Problem where tree is constructed based on choices.

Such a tree will contain all possible solutions, it is called space state Tree.

Promising & Non-Promising Node

(i) A Node in a state space tree is set to be promising if it corresponds to a partially constructed Solution from which a complete solution can be obtained.

(ii) Nodes which are not promising for a solution in a state space tree is called non-promising node.

Back Tracking technique

Here, solution is exposed as n-tuple $(x_1, x_2 \ldots x_n)$ where x_i is choosen from set 'S'

Eg: N-Queens Problem.

↳ can be divided into

(i) Decision Problem

(ii) Optimization Problem

(iii) Enumeration Problem

N-Queens Problem The N-QUEEN Problem is a Generalization of 4-QUEEN's (or) 8-QUEEN's Problem. Here The NQUEEN's are placed on "n x n" chessboard so that no two queens are attacking eachother that is

(i) No 2 QUEEN in a same row
(ii) No 2 Queens in a same Column
(iii) No 2 Queens in the same diagonal **CONSTRAINT**

- The chess board is considered as a two dimensional array
- * 2D araraya[1:n, 1:n] * place(k,i) returns boolean value i.e True if the queen can be placed in a column i., it is Two conditions. (i) whether 'i' is distinct from all previous values x[1], ... x[k-1]

(ii) No other Queen on Same Diagonal.

Soln: The Nqueens Problem using backtracking to bring all possible placement of N queens on a n×n chessboard without attacking eachother.

```
{for i=1 to n do
 { if place(k,i) then
  {x[k]=i;
    if (k=n) then
     write (x[1:n]);
    else
      NQueens(k+1,n);
  }}
NQueens(k-1,n);
 i=x[k-1]+1;
}
```

Algorithm Place (k,i);

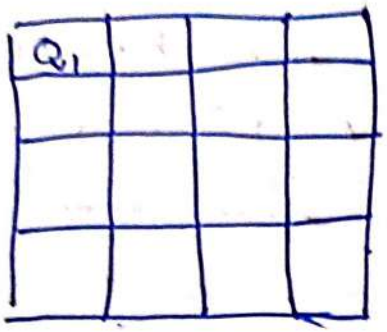
* To place the new queen in a chess board it returns true if the queen can be placed in k row and i column otherwise it returns false

* Array x[] is a global array it takes value upto k-1

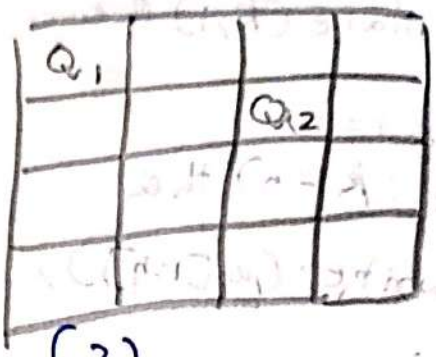
* Abs(r)
  Absolute of r → it returns the absolute value

```
{ for j=1 to k-1 do
    if ((x[j]=i)
        or (Abs[x[j]-i] = Abs(j-k))
    then return false;
  return true;
}
```

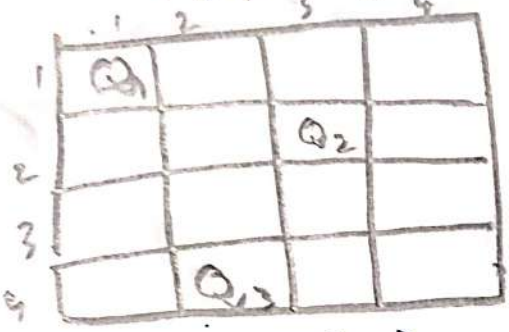
N - Queens Problem :- N = 4



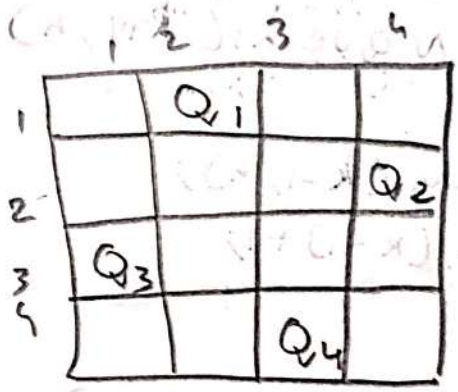
4 × 4 (1)



(2)



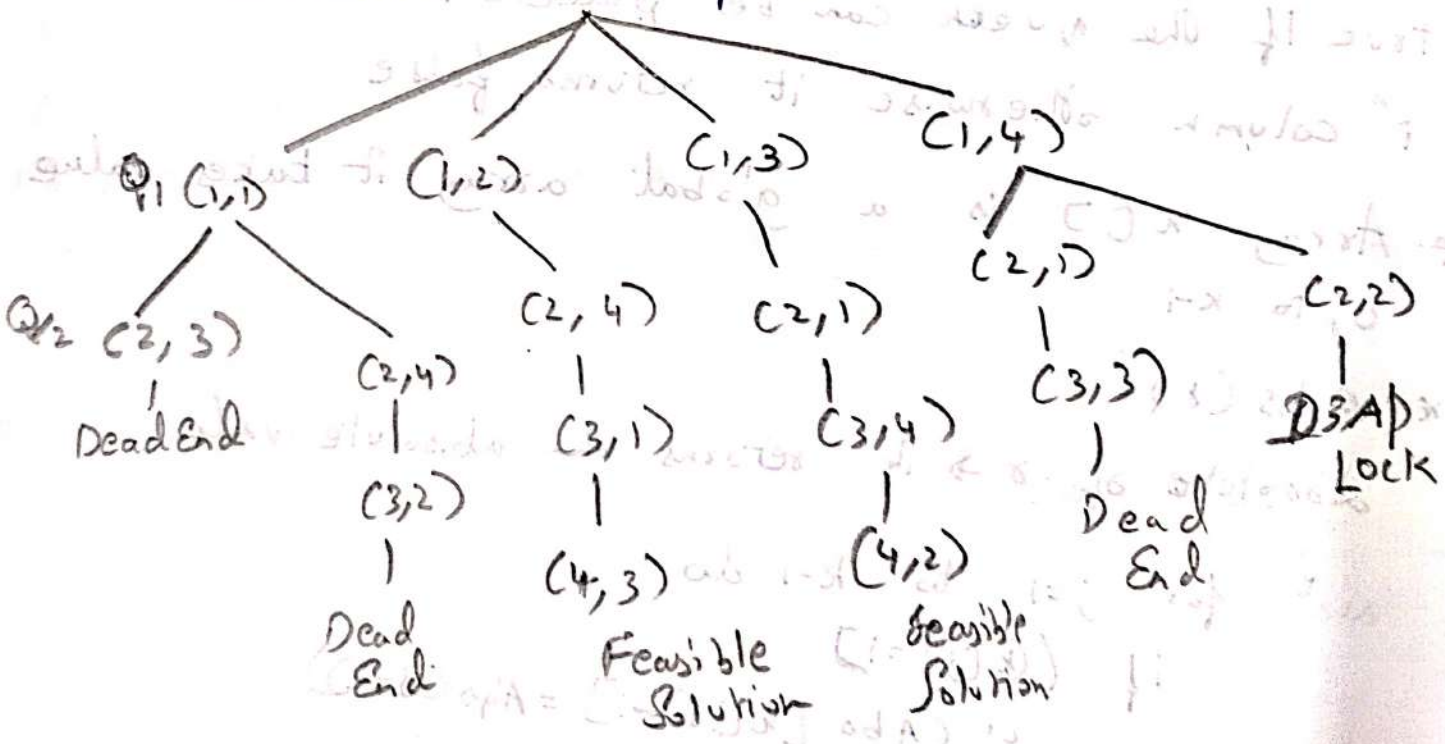
(3) (1,1)

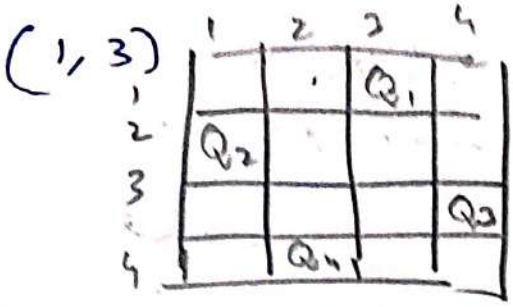


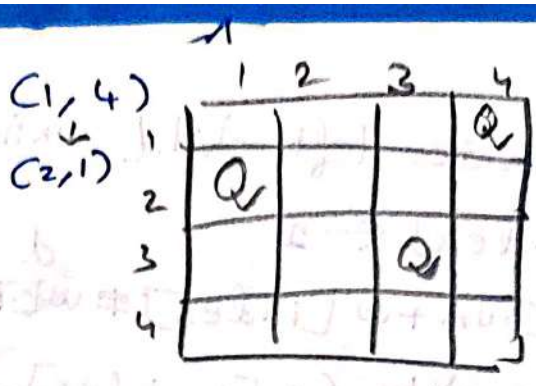
(1,2)

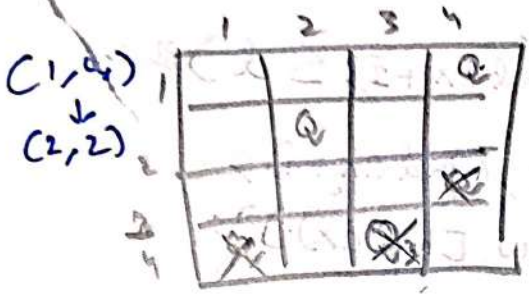
(4)

Solution State Space Tree









Sum of Subset

Algorithm Sum Subset (sum, index, R-sum)

//sum, //sum is a variable that stores sum of all the selected elements

//index it denotes index position of the chosen element

// R-sum, remaining sum is the initially considered the sum of all the elements.

// w[1...n] set of n elements

// a[j] is a subset of lies between 1 to index value

$$// sum = \sum_{j=1}^{index-1} w[j] * a[j]$$

$$// R\text{-}Sum = \sum_{j=index}^{n} w[j]$$

// w[j] → Stores the feasible solution in non decreasing order

$$\sum_{i=1}^{n} w[i] \geq d$$

```
{ // Generates left child until sum + w[index] ≤ d
  a[index] ← 1
  if (sum + w[index] = d) then
      write (a[1...index])
  elseif (sum + w[index] + w[index+1] ≤ d) then
      sum - subset((sum + w[index]), (index+1),
                   (R - sum - w[index]));

// Generates Right child
  if ((sum + R - sum - w[index] ≥ d) AND (sum + w[index+1] ≤ d))
                                                  then
    a[index] ← 0;
    sum - subset(sum, (index+1), R - sum - w[index]);
}
```

Sum of Subset:

* Consider a set S = {5, 10, 12, 13, 15, 18} and d = 30

{5, 10, 15} = d

Subset values	Sum value	Solution Type
{ }	0	—
{5}	5 < 30	———
{5, 10}	15 < 30	Not a feasible Solution
{5, 10, 12}	27 < 30	Not a feasible Solution
{5, 10, 12, 13}	40 > 30	Not a feasible solution
{5, 10, 12, 15}	42 > 30	Back tracking
{5, 10, 12, 18}	45 > 30	Back tracking
		Back tracking

Subset Values	Sum value	Solution Type.
{9,10,13}	28 < 30	Not a feasible Solution
{5,10,15}	30 = 30	feasible solution
{5,10,13,18}	46 > 30	Backtracking
{5,12}	17 < 30	Not feasible solution
{5,12,13}	30 = 30	Feasible solution