Unit - 1

Algorithm

algorithm means a finite set of instructions which are useful for solve any problem.

Characteristics of Algorithm

They are 5 characteristics

1) input :- an algorithm accept either 0 or more number of inputs.

for ex: we want to display "welcome to c lang" on monitor. so in this algorithm there is no input our target is only display output

2) output :- An algorithm must produce minimum 1 output (i.e 1 or more) without output we can not write any algorithm. So an algorithm must produce 1 output. (or) more num of outputs

3) Definiteness :- definiteness means each step of an algorithm must be clear and un ambigious (i.e instruction have only 1 meaning)

Not like add 1 or 2 to c

4) Finiteness :- Algorithm must contain finiteness no. of steps. (i.e algorithm must be terminated after a countable step

5) Effectiveness : Each step of an algorithm must be effective (we have to solve that algorithm people with the help of paper and pencil in finite amount of tim

Ex: we want to perform multiplication of 2 matrices, solve this problem using paper & pencil in a finite amount of time is called Effectiveness.

∴ Every Algorithm must satisfy these 5 characteristic (or) properties

issues in the study of Algorithms

in order to study of Algorithm we have to focus mainly 4 issues

1. How to devise (→ means create) Algorithm → In order to create Algo we have several APproads by using those AP we create

what are the APproaches to create an Algorithms

1. Divide & conquer
2. Greedy
3. Dynamic programming
4. Branch & Bound
5. Back tracking

by using
an algor
This is

2. How
valid
a algori

* suppose
Then
to

* if i
no
step

3.

The
Time complexity
* A
Pm
Algorithm

by using any one of these approach we create an algorithm
This is the first issue.

2. How to Validate Algorithms?

validation means the process of checking wheather a algorithm is producing correct results or not.

* suppose algorithms desn't produce correct result Then we have to rectify the problem and according to the we have to modify the algorithm

* if it is producing correct results Then there is no problem. we can continue with the next step. (i.e go to 3 step) otherwise Modify the algorithm

3. How to Analyze Algorithm?

Analysis is the Task of determining The time complexity & space complexity.

Time complexity

* Analysis Task of determing time is How much computing time and & storage space required for that Algorithm.

Algorithm

Algorithm

computing time :- means time complexity the amount of a time an algorithm requing for it's execution.

space complexity :- ~~An algorithm~~ The amount of a space an algorithm requires for it's execution

inorder to calculate the time complexity we have
2 Approaches

1. frequency or step count
2. Asymptotic notations

4. How to Test a program.

Testing involves 2

1. Debugging → it is checking wheather the program producing correct result or not
2. profiling (or) Performance measurement

Debugging

it is checking wheather the prog producing corred result or not. if it is producing correct result there is no problem. if it is producing not correct results then we have modify or rectify results. after modifications over

* once again perform the Debugging
* so we have to Repeat the proccdure until the program produces correct results

profiling

profiling is nothing but performance measurement on the modified program. so once the debbuging is over then we will get a new program. so we have to analyze, we have to measure the Performan

of that algorithm using profiling

* This alg but how to Test a program

By considering these 4 Issues we can study the Algorithm

Performance Analysis

we can Analyze a performance of Algorithm by using two ways

1. Time complexity → amount of time required for execution
2. Space complexity → amount of space required for execution

Time complexity

amount of time required for execution

* There are mainly 3 Types of Time complexities

1. Best case time complexity
2. worst case time complexity
3. average case time complexity

1. Best case time complexity

if an algorithm requires minimum amount of time for it's execution then it is called Best case time complexity.

2. worst case time complexity

if an algorithm requires maximum amount of time for it's execution then it is called worst case time complexity.

3. Average case time complexity

If an algorithm requires average amount of time for it's execution. then it is called Average case Time complexity

Example of best case worst case average case

linear search

searching elements of the list one by one until the Key element is found (or) The array is completely exhausted.

Ex: 10 20 30 40 50

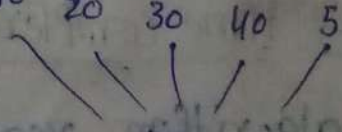
Key element is 40

compare 10 with 40 not equal, 20 with 40 not equal, 30 with 40 not equal, 40 with 40 equal

so we can say that the Key element is found.

remaining are elements of the array/list.

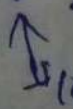
Ex: 10 20 30 40 50



The array is exhausted. we can say Key element Not found

*By searching using linear seach. Element found in 1st position

10 20 30 40 50



* Then it is called as Best case time complexity

Worst case complexity

By searching linear search, if the key element requires maximum amount of time is called worst case, found in last position

Ex: 10 20 30 40 50 →

50 ←

* The key element found in 2nd or 3rd positions or like these called as average time complexity.

Ex: 10 20 30 40 50 → average

50

To calculate Time complexity

we have 2 Approaches

1. Frequency count (or) step count
2. Asymptotic notations

Frequency count (or) step count

it specifics the number of times the statement to be executed

* How many times the corresponding time is executed. so that is ntg but frequency count

4 rules

1. For comments
   declaration
   step count is 0

2. return, Assignment statement:
count is 1

3. Ignore lower order Exponents when higher order Exponents are present
consider it
Ex $3n^4 + 4n^3 + 10n^2 + n + 1$
big order

4. Ignore constant multiplier
order of
Big 'O'
$3n^4 \Longrightarrow O(n^4)$
remove multiplier

Ex

```
int sum(int a[], int n)
{
    s = 0;                          -> for Assignment statement count = 1
    for (i=0; i<n; i++)             -> n+1
        s = s + a[i];               -> n
    return s;                       -> for Assignment/return count is 1
                                    -> return statement
}
```

add all freq = 1 + n+1 + n + 1 = 2n+3
↳ high order
Time complexity = O(n)

n=3
0<3
1<3
2<3
3<3
↳ 4 times checked
4 n+g but n+1

only 3 times Executed
3 n+g but n

29e Dynamic proc
Space complexity
+ amount of

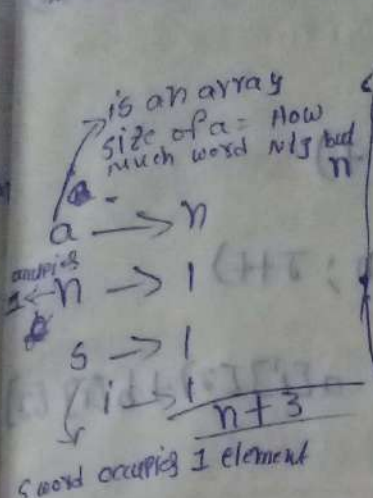
is an array
size of a = How much word n+g
a → n
occupies
n → 1
s → 1
i → 1
n+3
s word occupies 1 elem

Space com

Ex 2:
a, b, c
n=3
3×3
3 rows
3 columns

Space complexity of this algorithm

* amount of space requires for execution of Algorithm

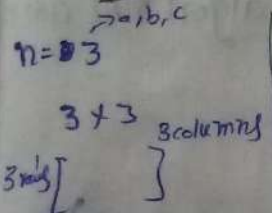



```
Ex:
int sum (int a[], int n)
{
  s=0;
  for(i=0; i<n; i++)
    s = s+a[i];
  return s;
}
```

Space complexity = O(n)

finding [] time complexity & space comp on this algo

Ex 2: matrix addition algorithm




```
void mat_add (int a[][], int b[][])
{
  int c[][];
  for(i=0; i<n; i++)          ---> n+1
  {
    for(j=0; j<n; j++)        ---> n x (n+1)
    {
      c[i][j] = a[i][j]+b[i][j];   ---> n x n
    }
  }
}
```

$= n+1 + n^2+n + n^2$

$= 2n^2+2n+1$

Ignore ↓ constant multiplier; ↓ higher order

Time complexity = $O(n^2)$

Space complexity

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

Variable we have
size = n/n = 3/3 = 9

a → n2
b → n2
c → n2
i → 1
j → 1
n → 1

3n2+3

Space complexity

= O(n²)

EX3 ; find the time complexity for this algorithm in frequency count (or) step count.
Matrix multiplication Algorithm ?

```
Void mat_mat(int a[][], int b[][])
{
    for(i=0, i<n; i++)
    {
        for(j=0, j<n; j++)
        {
            c[i][j] = 0;
            for(k=0; k<n; k++)
            {
                c[i][j] += a[i][k] * b[
            }
        }
    }
}
```

outer loop always executed n+1

Inside of the outer subloops = n

n+1
n²+1
n²
n³+n²
n³

2n³+3n²

Ignor multiplier

Time co

Space

frequen

memory
variables

a →
b →
c →
i →
j →
k →

3n²+

this outer

- They

*(n+1)

[j];

$n+1$

n^2+1

n^2

n^3+n^2

n^3

$2n^3+3n^2+2n+1$

↳ high order

Ignor multiplier

Time complexity = $O(n^3)$

Space complexity

frequency count (or) step count

memory occupied variables		
a	n^2	normal variable
b	n^2	
c	n^2	
i	$n+1$	
j	1	
k	1	
	$3n^2+4$	

↳ high order

Space complexity = $O(n^2)$

```
void fmatt_mat (int a[][],int b[][]
{
for (i=0, i<n; i++)
{
for (j=0; j<n; j++)
{
c[i][j]=0;
for (k=0; k<n;k++)
{
c[i][j]+= a[i][k]* b[k][j];
```

this becomes outer loop so n+1

$n*(n+1)$

} - They are sub loops

$n*n*(n+1)$ → this becomes outer loop so add n+1

$n*n*n$ → sub loop

Aysmptotic Notation

Aysmptotic Notation are mainly useful to calculate time complexity of an algorithm

* By using Aysmptotic Notations we calculate
  1) Best case time complexity
  2) Worst case time complexity
  3) average case time complexity

There are 5 Types of Aysmptotic Notations They are

1. Big oh Notation (O)
2. Big omeg ll (Ω)
3. Theta Notation (Θ)
4. little oh Notation (o)
5. little omega Notation (ω)

* By using These Notation To calculate time complexity.

1. Big-oh Notation (O) :- let f(n), g(n) be 2 Non Negitive functions.

* ~~Upper~~ Big-oh mainly represents upper Bound of Algorithms of Runing time.

* By using Big-oh Notation we can calculate max amount of time That can requires it for execution.

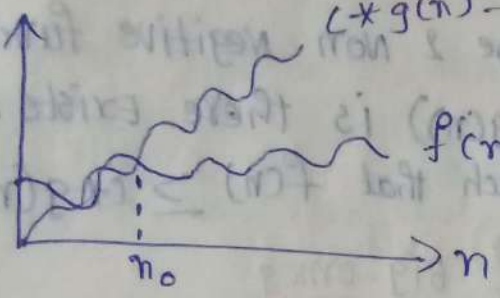
* We can ~~calculate~~ by using Big-oh to calculate worst case T.C

Definition of Big-oh

Let $f(n)$, $g(n)$ be 2 Non Negelive functions
Then $f(n) = O(g(n))$ if there Exists 2 +ve constants c, n_0 such that $f(n) \leq c * g(n)$, $\forall\ n > n_0$

Graphical Representation of Big oh Notation

x -> represents n
y -> t



if n is greater than n_0 $f(n)$ value should be less than or equal to $\leq c * g(n)$ -> This is known as upper Bound.

1) $f(n) = 3n+2$ $g(n) = n$ then prove $f(n) = O(g(n))$

$f(n) \leq c * g(n)$ $\forall\ n > n_0$ -> it is satisfy we can say that $f(n) = O(g(n))$

A) $3n+2 \leq c * g(n)$ $\quad g(n) = n$, let $c = 4$

$3n+2 \leq 4 * n$

let $n = n_0 = 1$ ->

$\Rightarrow 5 \leq 4$ X

n_0 value correct in $n_0 \geq 2$

let $n = n_0 = 2$ $\quad n_0 = 2$

$3(2)+2 \leq 4 * 2$

$\Rightarrow 8 \leq 8$ ✓

let $n = n_0 = 3$

$3n+2 \leq c * g(n)$

$3(3)+2 \leq 4 * 3 \Rightarrow 11 \leq 12$ ✓

Big-omega Notation (Ω)

→ Big-omega mainly represents lower Bound of Algorithm's run time.

→ it mainly represents minimum amount of time that can requires it for execution.

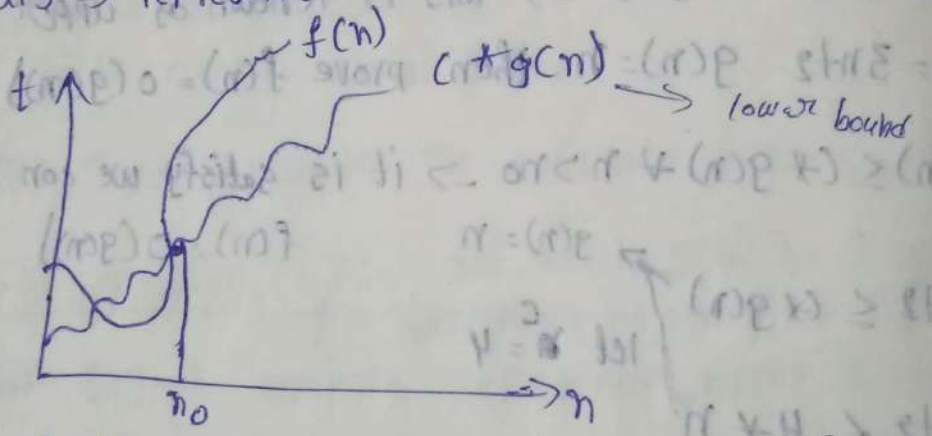
→ minim amount Nlg but Best case of complexity.

Def :- let $f(n)$, $g(n)$ be 2 Non Negitive functions Then $f(n) = \Omega(g(n))$ is there Exists 2 the constants c, n_0 such that $f(n) \geq c * g(n), \forall n > n_0$

graphical representation of Big-omeg

x axis → represents time t

y axis → represents n



Ex → $f(n) = 3n+2$ $g(n) = n$ then prove $f(n) = \Omega(g(n))$

prove that $f(n) \geq c * g(n) \forall n > n_0$

$3n+2 \geq c * n$

let $c = 1$

$3n+2 \geq 1 * n$

let $n_0 = 1 \Rightarrow 5 \geq 1$ ✓ $n_0 > 1$

$n_0 = 2 \rightarrow 8 \geq 2$ $n_0 > 2$ ✓

This is the omega Notation.

Theta Notatio

→ Theta Not
Bound of

→ it mainl
that can

→ Avg an

Def

let f(n

Then f(n) =

c_1 ,

graphical

x axis →

y axis →

t

Ex → f

prove th

f(n)

f

c_1 *

D

Theta Notation (Θ)

-> Theta notation mainly represents Average Bound of Algorithm's run time.

-> it mainly represents ~~minimum~~ Average amount of time that can requires it for execution

-> Avg amount Nothing but avg case of Time complexity

Def

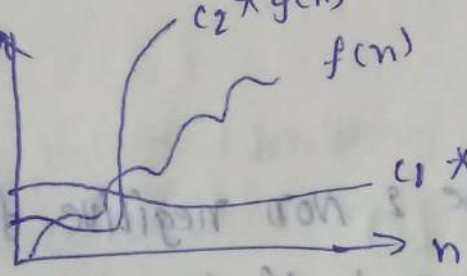
let $f(n), g(n)$ be 2 Non Negetive functions

Then $f(n) = \Theta(g(n))$ if there Exists 2 the constants c_1, c_2, n_0 such that $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$ $\forall n > n_0$

Graphical representation of Theta notation

x axis -> no. of elements

y axis -> represents t



Ex -> $f(n) = 3n+2$ $g(n) = n$ then prove $f(n) = \Theta(g(n))$

prove that $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$

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

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

$c_1 * g(n) \leq f(n) \leq c_2 * g(n)$

$c_1 = 1$ $c_2 = 4$

$1 * n \leq 3n+2 \leq 4 * n$

let $n_0 = 1$

$1 \leq 5 \leq 4$ ✗

$f(n) \leq c * g(n)$ $c = 4$

$f(n) \geq c * g(n)$ $c_2 = 4$

$n_0 = 2$ ✓ $n_0 \geq 2$

$\Rightarrow 2 \leq 8 \leq 8$ ✓

let $n_0 = 3$

$3 \leq 11 \leq 12$

In this way we calculate average Theta Notation of

4. little oh (o):-

Def :- let $f(n)$, $g(n)$ be 2 non negitive functions

Then $f(n) = o(g(n))$ such that

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

5. little - omega (ω):-

Def :- let $f(n)$, $g(n)$ be 2 non negitive functions

Then $f(n) = \omega(g(n))$ such that

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

(or)

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

Space co

* How Mu
Ntg but

To solve
can develop

* let us a
Algorithm/

* Aim is
of space

Formula

space ← SCP
comp

Ex 1 :-

* Here
we are
Algorith

* which
that
based

Ex 1

Space complexity of an Algorithm in DS

* How much space required for it's execution. is Nlg but space complexity.

To solve any problem to develop a program. But we can develop a program in no. of Algorithms.

* let us assume that we want to implement sorting Algorithm / sorting program

✓ Aim is our Algorithm should contain less number of space. So that is our target

Formula for calculating space complexity

Space comp ← $S(P) = C + S_P$ → Instance characteristic (or) dependent (or) variable Part

$S(P)$ → Problem

C → constant Part (or) Independent Part (or) fixed Part

Ex1 :-

* Here we are developing an Algorithm. He we are calculating the space complexity for the Algorithm

* which Algorithm requires less space that Algorithm that Algorithm should be return with help of a program based on that program we develop a program

Ex1 :-

```
Algorithm sum (a,b,c)
{
    a = 10;
    b = 20;
    c = a+b;
}
```

In this way we calculate 2 numbers.

$S(P) = C + SP$ → no instance part here so $=0$ → Now, we have to calculate space complexity by using formula

$= 3 + 0$

$= 3$ time units ↓ constant

a → 1 time unit
b → 1 time unit
c → 1 time unit
3 ↓ constant c

Assuming using a 32 bit compiler → then the Integer is 2 bits

Assuming using a 64 bit compiler → then size of Integer is 4 bits
size

∴ So space complexity = $O(1)$ → 1 specifies some constant

when it is 32 bit using = 2 bytes
when compiler 64
The n = 4 bytes
a = 4
b = 4
c = 4
12

a = 2
b = 2
c = 2
6
size

Ex2 :- calculate some of Array elements.

Algorithm sum(a, n) → Array, → size of Array
{
total = 0;
for (i = 0 to n do → depending part depends on n Ex: n=5
total = total + a[i];
}

$S(P) = C + SP$ → 3, → 5×n

$S(P) = 3 + 5 \times n$ → Ignore con multi, → highorder

=

Space complexity = $O(n)$

The 5+n
n → 1
total → 1
i → 1
3

n = 3
SP = 3×n

SP → Instance ch
Here (a) is
size of (a)

Various types of computing Times
(or)
Typical growth rates

$O(1)$ → constant computing time

$O(n^2)$ → Quadratic computing time

$O(n^3)$ → cubic computing time

$O(2^n)$ → Exponential computing time

$O(\log n)$, $O(n \log n)$ logarithamic computing time

n	n^2	n^3	2^n	log n	n log n
1	1	1	2	0	0
2	4	8	4	1	2
4	16	64	16	2	8
8	64	512	256	3	24

contains better values

one relationship among computing time is

$\log_2^2 = 1 = O(\log(n)) < O(n) < (n \log(n)) < O(n^2) < O(2^n) < O(n^3)$

It contains Best Time complexity

Relationship

It contains worst comp

General Rules (or) Norms for calculating time complexity

Rule 1: Loop

The Runing time of for loop is the runing time of statements in the for loop

running time for loop

```
Ex:- for (i=0; i<n; i++)      ---- n+1
         s = s+i;             ---- n
```

So, time complexity = $O(n)$

Runing statement in for loop

$\frac{n+1}{n}$

$2n+1$

If n = 3
0 < 3
1 < 3
2 < 3

Rule 2 : Nested loop (writing 1 loop inside another loop)

* if there is a nested loop How we calcalate Time complexity we discuss here

- The total runing time of statements inside a group of nested loops is the product of sizes of all loops.

Ex: for (i=0; i<n; i++) ——→ n+1

{
  for (j=0; j<n; j++) → This loop becomes outer loop — n * (n+1)
  {
    c[i][j] = a[i][j] + b[i][j]; → n * n
  }
}

n+1 : => : n+1
n * (n+1) : n² + n
n * n : n²

Time complexity = $2n^2 + n + 1$ → highorder
Ignore constant

$O(n^2)$

Rule 3 :- Consecutive statements

If there are some consecutive statements Then the runing time is maximum 1

* Here runing time the maximum one.

Ex :-
1st statement { for (
for
2nd statement {

-> inside loop

if
Here ru
* if s1
The ti
* suppose
the ti
Ex :-

Ex :

If (or) else
executed
1 time

Ex:-

```
for(i=0; i<n; i++)        ---> n+1
    s=s+i;                ---> n
for(j=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
    }
```

1st statement

2nd statement

$= O(n^2)$

$n+1$
n
$n+1$
n^2+n
n^2
$2n^2+4n+2$ → high order; ignore const mult

Rule 4:- if-else

if

Here running time is max of s_1 and s_2

* if s_1 have maximum time complexity s_1 become the time complexity of entire block/code.

* suppose s_2 has maximum time complexity s_2 become the time complexity of entire code/block.

Ex:- if (cond)
s_1
else
s_2

suppose if statement executed 1 time

Ex:

```
if(n<=0)          ---> 1
    return n;     ---> 1
else              ---> 1
{
}
```

return statem step count is always 1

If (or) else block executed suppose else 1 time

```
for (i=0; i<n; i++)  ——→ n+1
    s=s+i;           —— n
return s;            —— 1
}
```

$2n+5$

↓ Ignore const ↳ high order

$= O(n)$

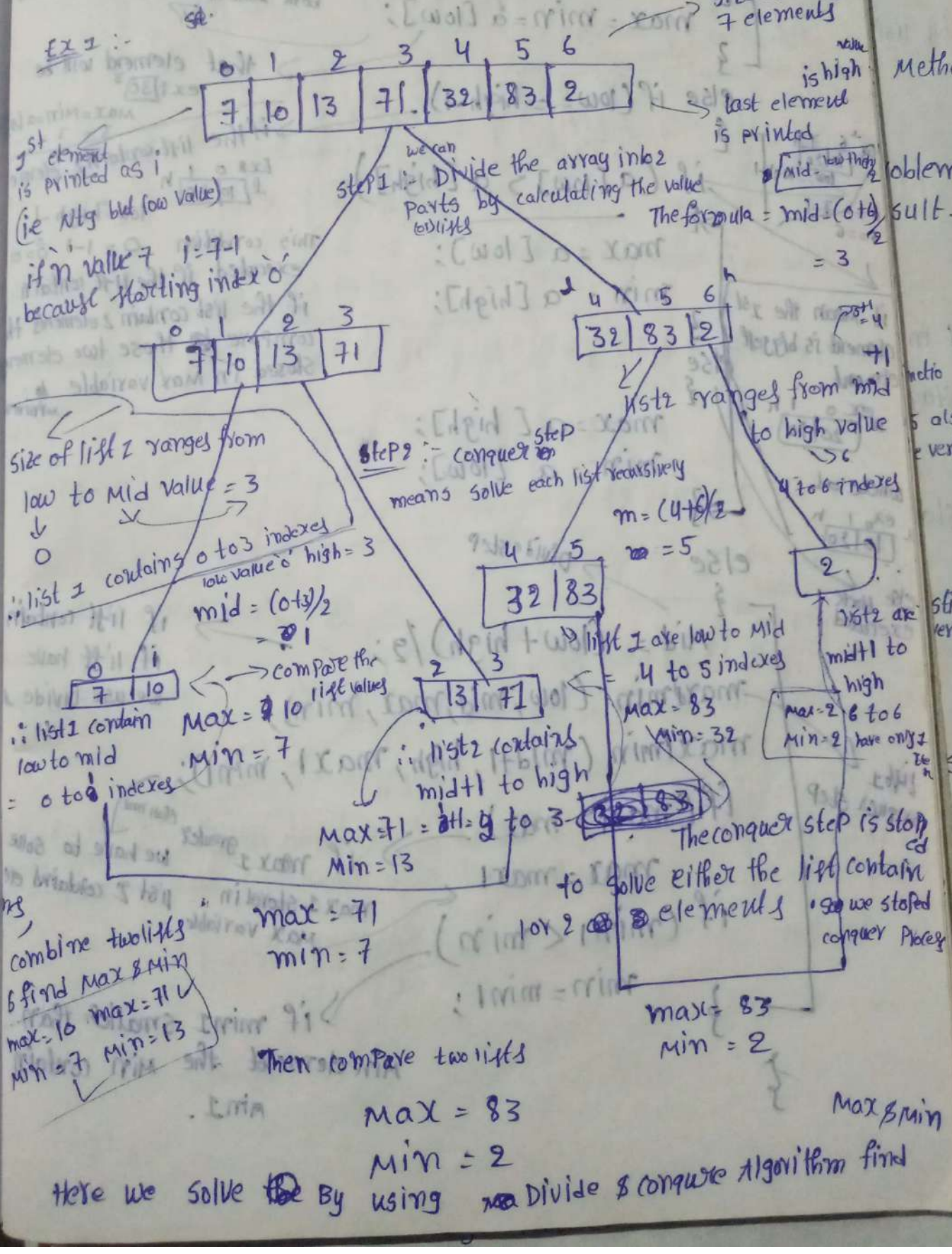
∴ So the time complexity of above code become $= O(n)$

Finding maximum & minimum Element using Divide & conquer Algorithm

* Divide and conquer Algorithm is implemented using 3 steps
  1) Divide
  2) conquer
  3) combine statement
* In Divide ~~and conquer Algorithm~~ the given array is divided into 2 parts
* In conque step the given array is divided into sub problem, sub list or is solved recursively
* In combine step The solution of the sub problems sub lists or merged or combine

Let us discuss How can find out maximum & minimum Element using Divide & conquer Algorithm

* let we have an array that contains n elements
* let us see first example and then see Algorithm, as how to find out time complexity for finding the maximum & minimum element



Algorithm

```
Algorithm maxmin (low, high, max, min)
{
  if (low == high)
  {
    max = min = a[low];
  }
  else if (low == high-1)
  {
    if (a[low] > a[high])
    {
      max = a[low];
      min = a[high];
    }
    else
    {
      max = a[high];
      min = a[low];
    }
  }
  else
  {
    Mid = (low + high)/2;
    maxmin (low, mid, max, min);
    maxmin (mid+1, high, max1, min1);
    if (max < max1)
      max = max1;
    if (min1 < min)
      min = min1;
  }
}
```

if the list contain only 1 element then what we happend that element both max element & min element

that element will be stored

ex: 30 → max = min = 30

if the list contain 2 elements

ex: 0 1 h
10 20

low = high-1

This condition 0 = 1-1 executed if the list contain 2 elements

if the list contain 2 elements then we compare those two elements stored in max variable & min variable

ex: 6 7 h
10 20
low == high+1
6 == 7-1
6 == 6

it mean the 1st element is biggest element

ex: 0 1 h
20 10

where as

ex: 0 1 h
10 20

else block executed

Divide step

if list contain more than 2 elements or if list have more than 2 elements we use Divide & conquer Approach

left conquer step

if max1 greater than max we have to store max1 stored in max variable

list 2 contains

if min1 smaller than min element the min contains min1.

if left & recursive method which

How to find time complexity

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \\ 2T(\frac{n}{2}) + 2 & \text{if } n > 2 \end{cases}$$

- if list contain only 1 element → if not the 0 comparision because that element become both max & min
- → list contain 2 elem then we require 1 comparision. Ex: compar 10 to 20 is 1 comp
- 2 partition there; if list contain more than 2 elements:

ex:	10 30	40 50
	T(n/2)	T(n/2)
	$2T(n/2) + 2$	

→ 1 for divide 2 part, 1 for combining, = 2. Then divide list into 2 part.

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

↓ Here we substitute $T(n/2)$ formula

$$= 2\left[2T(n/4) + 2\right] + 2$$

if mean in place of 'n' we substitute n/2

$$\Rightarrow 4T\left(\frac{n}{4}\right) + 4 + 2 \rightarrow 2^2 T(n/2^2) + 2^2 + 2^1$$

$$= 4\left[2T(n/8) + 2\right] + 4 + 2$$ This is the formula for $T(n/4)$

$$= 8T(n/8) + 8 + 4 + 2$$

$$= 2^3 T\left(n/2^3\right) + 2^3 + 2^2 + 2^1$$

we can write last term as T(n) = last term denoted as k-1

$$T(n) = 2^{k-1} \cdot T\left(\frac{n}{2^{k-1}}\right) + \sum_{i=1}^{k-1} 2^i \quad \therefore \; \sum_{i=1}^{3} 2^i$$

$$2^3 \cdot T\left(\frac{n}{2^3}\right) +$$

if k = 4 ⇒ we get last term

if k = 3 we get last of 2nd term

(Left margin notes:)

- more than 2 elements
- or than 2 elements Then
- conquer Approach of 3 steps
- ... recursively,
- ... which ranges midtl to high)
- high value
- ... element will be stored in min var

last term:

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

a^n means $1 \rightarrow a^m/a^n$

$$= \frac{2^k}{2} \cdot T\left(\frac{2n}{2^k}\right) + \sum_{i=1}^{k-1} 2^i$$

geometrical progression $\rightarrow \frac{a(r^n-1)}{r-1}$

let $2^k = n$

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

$$\Rightarrow \frac{n}{2} \cdot T\left(\frac{2n}{n}\right) + \left(\frac{2 \cdot 2^k}{2} - 2\right)$$

$$\Rightarrow \frac{n}{2} \cdot T\left(\frac{2n}{n}\right) + (n-2)$$

assuming let $2^k = n$

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

$T(2)$ means one comparision

$$\Rightarrow \frac{n}{2} \cdot 1 + n - 2$$

$$\frac{n}{2} + n - 2$$

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

constant value → highorder

$$= \frac{3n}{2} - 2$$

$$= O(n)$$

Here is the time complexity of Best case T complexity avg case Time complexity, worst case time complexity. In order to find out Maximum & Minimum Elements

Binary

we

the array

array may

* if Array is all pos

. so, it is

* let

multiple of 2, 4, 8 values = 2

1st

Here

The we ca

Key eleme

any middle ele

if key ele

only left

we

middle

* here

we ha

low is

Binary search

we can perform binary search only when the array elements are in sorted order (that means array may be either Asending or desending order)

* if Array elements are not in proper order then it is all possible to apply binary search
. so, it is first condition of binary search

* Let n=7 & element of array are 10 15 18 30 50 70 90
we can perform these searching operation only when low value is less than or equal to i, Then we calculate mid

multiplier of 2, 4, 8 values = 2

formula for mid = $(l+h)/2$

$= (0+7)/2$

$= 7/2$

$= 3$

1st condition

The key of Index 3 is element $= 30$

Here the key element is equal to arrays Middle element The we can say that our searching process is sucessfull.

for ex: key = 10

Key element 10 array middle element 30. but key element less than middle element

if key element less than array middele element we have search only left part

0	1	2	3	4	5	6	7
10	15	18	30	50	70	90	100

No need to search right part

* we have to check wheather key element is less than the middle element or greater than key element

* here the key element less than array of middle element we have to search only the left part (ie 0 to 2) already low is at 0 position. so we have to updat high value that is 2 mid = 3 mid-1 = 2 -> This way to get 3 from 2.

high = mid - 1
= 2

Now the key element available blw 0 to 2. This is only first part.

Next condition

	low							high
	0	1	2	3	4	5	6	7
Ex:	10	15	18	30	50	70	90	100

let key = 100

* check first condition it means 100 == equal to 30 it is false
* check second condition key is less than array middle element (100 lessthan 30) it is also false.
* check third condition key is greater than array Middle element (100 greatter than 30) so, now we to search only the right part

Here no, need to check left part

* we have to compare from 4 to 7. already high is at 7 then we have to change low at 4 position
* How place low at 4 position, we know Mid value 3 then low = mid + 1

This is nothing but ~~linear~~ search tree

Expiramation

First we have to calculate Array's middle elements

1. compare key element with Array's middle element

if Key == a[mid] then Key element is found

Ex:-

let key = 300

	l					h
	0	1	(2)	3	4	5
	100	200	300	350	400	500

m = (l+h)/2
= (0+5/2)
= 2

if 300 == a[300] → array middle mein condition is true we can say that key element is found.

(2) If key < a[mid] the search only left part. So update high value. high = mid - 1

Key = let 100

l	0	1	(2)	3	4	5	h
	100	200	300	350	400	500	

l	0	1	(2)	h
	100	200	300	

So, high = m - 1
= 2 - 1
= 1

Sure key element left than array middle element so have to search only left part

(3) if key > a[mid] then search only right. So update low value

low = mid + 1
let k = 1000

l	0	1	(2)	3	4	5	h
	100	200	300	350	400	500	

1000 is greater than 300 so condition is true. So now we have to search only right part. because right part contains element greater than key element

l	3	4	5	h
	350	400	500	

~~high < m + 1~~

Place low at 3 position low = mid + 1

∴ This is the explanation of Binary search

No we write Algorithm for Binary search

Binary Search Algorithm → 1st we write array size

Input :- Array size (n), Array elements, key element

output :- Key element is found or not found

we get one or 2 msgs

* Now write Algorithm first step

1. flag = 0 → then we say key element not found
   flag = 1 then say key element found
2. low = 0
3. high = n - 1 → write loop of largest low < high
4. while low <= high do

4.1) mid = (low + high)/2

4.2) if key == a[mid] then
   a. flag = 1
   b. break → break executed the control where ever key element found

4.3) if key < a[mid] then
   a. high = mid - 1

4.4) if key > a[mid] then
   a. low = mid + 1

if key greater mid. we have to search only right part

5. if flag == 1 then
   a. print "key element found"

6. else
   b. print "key element not found"

So, this is the Algorithm for binary search

Now we see for program of binary search

C program to implement binary search

```
#include<stdio.h>
main()
{
   int n, a[10], key, low, mid, high, flag;
   printf("enter array size");
   scanf("%d", &n);
   printf("enter array elements");
   for(i=0; i<n; i++)
      scanf("%d", &a[i])
Printf("enter key element");
```

→ some array

scanf("
flag =
low =
while hi
no need to search

flag == 1

Tim

Time
for it

```
scanf ("%d", &key);
flag = 0;
low = 0;
high = n-1;
while (low <= high).
{
    mid = (low+high)/2;
    if (key == a[mid]).
    {
        flag = 1;
        break;
    }
    else if (key<a[mid])
        high = mid-1;
    else
        low = mid+1;
}
if (flag == 1)
    printf ("Key element found");
else
    printf ("Key element not found");
}
```

while no need to search

60 == 40 false

60 < 40 false

60 > 40, low = mid+1

low ≤ high = 4 ≤ 6 ✓

mid or 2 = 5

flag == 1

Time complexity of binary search

Time complexity means the amount of time required for it's execution.

Here 3 types of times complexity

1) Best case time complexity
2) worst case time complexity
3) Avg case time complexity.

- Best case time complexity

* if an algorithm requires minimum amount of time for it's execution then it is called as best-case time complexity.

* In Binary search the best-case will occur. if key element is exactly the array's middle element then only 1 comparision is here

$T(n) = O(1)$ → that require only 1 comp

2. worst case time complexity → maximum amount of time require for it's

In Binary search the worst case will occur if key element is either 1st element or last element of the array then the binary search Algorithm requires maximum no. of comparisions for it's execution.

Reqgurance relation $T(n) = T\left(\frac{n}{2}\right) + 1$ → Dividing the array in 2 parts → for comparision

time complex for array of n elements

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

$= T(n/8)$ $1+1+1$ = $T\left(\frac{n}{2^2}\right) + 1$

$= T\left(\frac{n}{2^3}\right) + 1$

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

let $2^k = n$

Apply log on b.s

$\log 2^k = \log n$

$k \log 2 = \log n$

$k \cdot \frac{2}{1} = \log n$

$k = 2 \log n$

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

$= T(1) + \log n$

$= 1 + \log n$

$= \log n$

worst case time complexity = $O(\log n)$

3. Average - time complexity

It means it requires Average amount of time required for it's execution

The avg - time complexity is also = $O(\log n)$

Merge sort

* Merge sort work based on the principle of Divide and conquer Algorithm
* Merge sort is divided into 3 steps
  1. Divide
  2. conquer
  3. combine
* In the divide step the given array is divided into 2 parts
* In conquer step each sub array sorted recursively
* In combine step to fectch/combine the two solutions of array.

Algorithm
void merge_sort (int low, int high)
{
int mid
if (low < high)
{
mid = (low + high)/2;
merge_sort (low, mid);
merge_sort (mid+1, high);
merge (low, mid, high)
}
}

Algorithm
void Merge (int low, int mid, int high)
{
int i, j, k, b[20];
i = low;
j = mid+1;
k = low
while (i <= mid && j <= high)
{
if (a[i] < a[j])
{
b[k] = a[i];
i++;
}
else
{
b[k] = a[j];
i++;
}
k++;
}

starting value
size-of-array - 1
n-1
0 < 9 condition true
0 1 2 3 4
310 285 179 652 351
1st subarray
2nd subarray
1. Divide step
0 1 2 3 4 | 5 6
310 285 179 700 351 | 423 861
conquer step
1st subarray sorted
0 1 2 3 4 | 5 6
179 285 310 351 651 | 423 861
combine / Merge step
suppose resultant array
b[20] to store resultant array
1st array 1st element
compare element of 1st subary & 2nd array 1st element
i will be at 1st position
i = 0 1 2 3 4
j = 5 k = 0 1 2 3 4
* The smaller of these two will be moved to resultant array & we have to increment the appropriate counter value.
179 423 The smaller is 179 it is moved to resultant array.
285, 423 The smaller is 285 moved to resultant array
310, 423 310 smaller
351, 423 351 smaller
700, 423 423 smaller moved to resultant array. we have to increment j value as well
700, 658 658 smaller
700, 750 700 smaller
i will be at 5th position
condition (while i <= mid)
5 <= 4 condition false
it mean one of the sub array exhausted then we have to check wether 2nd sub array is completed or not, suppose there are some elements in 2nd sub array then we copy remaining elements to resultant array not completed array elements are moved to resultant

resulsat array
0 1 2 3 4 5 6 7 8 9
179 285 310 351 423 658 700 750 861 900
k value

if (i <= mid)
{
for (r = i; r <
{
b[k] = a[
k++;
}
}
remaining elements
6 7 8 9
861 750 900 658
8 9
900 658
7 8 9
750 861 900
array 2nd element
move to
ment the
array.
we have
as k value.
increment i value
false

if (i <= mid) → 1st sub array not completed

{ → remaining elements

for (r = i; r <= mid, r++;

{

b[k] = a[r]; r = i value ex 3 3 4

k++;

} condition true so copy a[r] value into b[k]

}

else

{

for (r = j; j <= high; r++)

{

b[k] = a[r];

k++;

}

}

Here we have to compare 1st element with 1st array to 1st element of 2nd sub array

The smaller of those two moved to resultant array and increment appropriate counter variables such as i, j, k

2nd step is if the array is not completed (while condition false) then we have to copy remaining 1st sub array elements into resultant array.

like wise 2nd sub array not completed then we have to copy remaining sub array elements into resultant array

if 2nd array not completed to copy those remaining elements to resultant array for r = j

Ex: Merge sort

	0	1	2	3	4	5	6
	52	34	21	8	62	82	4

first main function (we have to call merge sort function

void merge_sort (int low, int high)

int mid

if (low < high)

{

mid = (low + high)/2

merge_sort (low, mid);

merg_sort (mid+1, high);

merge (low, mid, high);

}

MS(0,6)

mid = (0+6)/2 = 3

MS(0,3) mid=1

MS(mid+1, high) MS(4,6) Mid = 5

MS(4,5) mid=4

MS(5,6) mid=5

MS(0,1) mid=0

MS(2,3) Mid=2

MS(2,2) false

MS(mid+1, high) MS(3,3) false

MS(0,0) condition(false)

MS(1,1)

MS(0,0,1)

merge (0,1) element are sorted

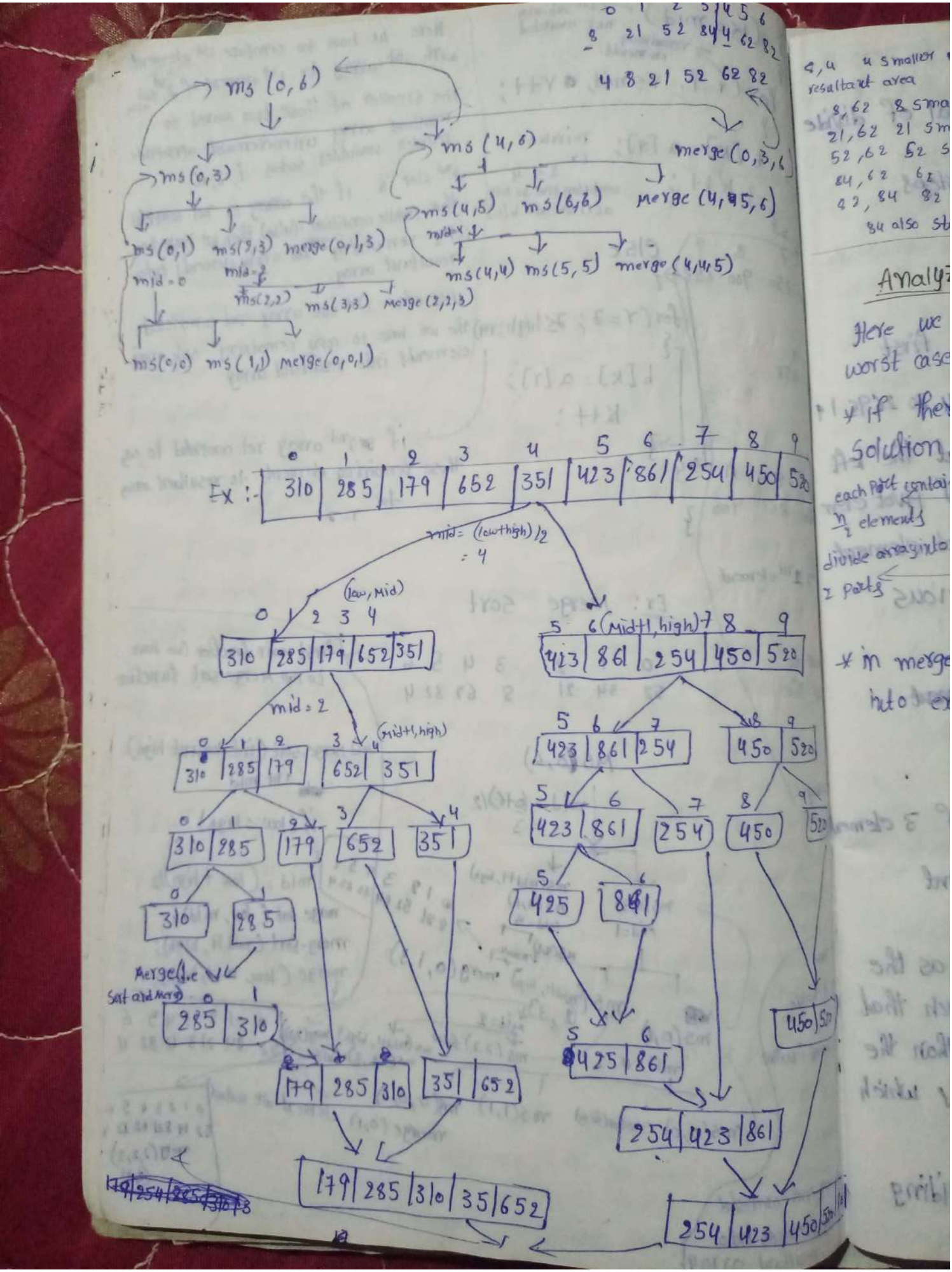
merg(0,1,3)

merg(2,3)

merg(2,2,3) (2,3)

	0	1	2	3	4	5	6
	52	84	21	8	62	82	4

array not completed array elements into resultant array


ms(0,6)
ms(0,3)
ms(4,6)
merge(0,3,6)
ms(0,1)
ms(2,3)
merge(0,1,3)
ms(4,5)
ms(6,6)
merge(4,5,6)
mid=0
mid=2
mid=4
ms(2,2)
ms(3,3)
merge(2,2,3)
ms(4,4)
ms(5,5)
merge(4,4,5)
ms(0,0)
ms(1,1)
merge(0,0,1)
Ex :-
310 285 179 652 351 423 861 254 450 520
mid = (low+high)/2
= 4
(low, Mid)
(mid+1, high)
Merge i.e
Sort and Merge
285 310
179 285 310 351 652
425 861
254 423 861
450 520
254 423 450
179 285 310 351 652
254 423 450

4, 4 4 smaller 4 stored in resultant area

8, 62 8 smaller

21, 62 21 smaller

52, 62 52 smaller

54, 62 62 smaller

62, 84 62 smaller

84 also stored in resultant array

179	254	285	310	350	423	450	5201

Analyze of merge sort

Here we Analyze of merge sort Algorithm of Best case, worst case, avg case

* if there is in only one element then we get the solution directly

each part contains $\frac{n}{2}$ elements

$T(n) = 1$ → There is no need to solve

$T(n) = 2T(\frac{n}{2}) + n$ → More than 1 we have to divide 2 parts

divide array into 2 parts

ex:- 40 30 60 | 10 40 50

require $\min(n)$ comparisions

* in merge sort the Best case will occur if we divide the given array into exactly two parts.

$T(n) = 2T(\frac{n}{2}) + n$ — $2 \times n/2$

$2^1 \cdot T(\frac{n}{2^1}) + 1 \cdot n = 2\left[2T(\frac{n/2}{2}) + n/2\right] + n$

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

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

$2^3 \cdot T(n/8) + 3n = 8T(n/8) + 3n$

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

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

Let $2^K = n$

$\log_2 2^K = \log n$

$K \log 2 = \log n$

$K \cdot 1 = \log n$

$K = \log n$

substitute K value

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

$= n \cdot T(1) + n \log n$

$= n \cdot 1 + n \log n$

$= n + n \log n$ → high

$= O(n \log n)$

Quick Sort

* Quick sort works based on the principal of divide & conquer Algorithm
* Any Divide & conquer Algorithm using three steps.
  1) Divide
  2) conquer
  3) combine

* let see what is divide step here. we have select first element of array is called pivot element

1st Elem 60 39 46 56 84 30 26 96 14

* Divide the given array into two parts such that the left part contains the elements which of left hand of pivot element and right part of elements are greater than pivot element
* in order to select the pivot element we have various stratagies like
  1. we always select first element as pivot element
  2. Select the Last element as pivot element
  3. Select a random element as pivot element
  4. Take 3 elements and calculate the median of 3 elements

By using These 4 stratagies we select a pivot element

* Now Here in our example selecting 1st element as the pivot element, Divide the array into 2 parts such that the left part contains the elements which are less than the pivot element and right part contains the elements which are greater than the pivot element.
* This process is also called as partitioning. Dividing process is also known as 'partitioning process'

2. conq
In co
the 2nd
as a so
3. combin
* There
step nee

void
{

2. conquer step

In conquer step we have to sort first part as well as the 2nd part recursively. So that we will get the entire array as a sorted output

3. combine step

* There is no need combine step in Quick sort. combine step needed only in merg sort.

```
                                  → accept 2 argument
                                    1) low
                                    2) high
void quick_sort (int low, int high)
{                  → pivot element
  int P;
  if (low < high)
  {
    P = partition (low, high);
    quick_sort (low, P-1);
    quick_sort (p+1, high);
  }
if partition (int low, int high)
{
int i, j, pivot, temp;
pivot = a(low);
  i = low;
  j = high;
while (i<j)
{
  while (a[i]<=pivot)
      i++;
  while (a[j] >pivot)
      j--;
  if (i<j)
  { temp=a[i];
    a[i] = a[j];
    a[j] = temp;
  }
}
```

→ in order to calculate Pivot element we using a function called partition

Ex:

	i							J
	0	1	2	3	4	5	6	7	8
	60	39	46	56	84	30	26	96	19

select 1st as Pivot element

60 < 60 if a[i] < pivot the inc Algo

39 ≤ 60 condition true inc

46 ≤ 60 condition true

56 ≤ 60 condition true

84 ≤ 60 condition false

check if i is < j or not

4 < 8 condition true. swap

0	1	2	3	4	5	6	7	8
60	39	46	56	19	30	26	96	84

check if i<j or not

19 ≤ 60 condition true Inc

30 ≤ 60 true Inc

26 ≤ 60 true Inc

96 ≤ 60 false a[j]=84 84 > 60 true

swap[a[i], a[j]] no need to change 84. we have in 'i' value decrement 'j' value.

```
temp = a[low];
a[low] = a[j];
a[j] = temp;
return
}
```

swap(a[low], a[j])

The i & j are at 7 & 8 (96, 84)

96 > 60 There is no need to change 96 decrement j value

26 (i=6), 96 (i=7)

26 > 60 false came out from while loop

7 < 6 false comes out from while loop

This j element become Pivot

0	1	2	3	4	5	6 (j)	7 (i)	8
(60)	39	46	56	19	30	(26)	96	84

This Pivot element will Moved

26 → left part

60 → right part

→ Pivot element

Here 96 84 are greater than Pivot element

Ex:

0	1	2	3	4	5	6	7	8
34	26	1	(45)	18	(78)	(12)	89	(27)
			27		(12)	78		45
					34			

j → This become Pivot element

q (0, 8)

pivot = a[low]

Pivote element = 34

i = 0

i = 0; j = 8

i = 1; 26 ≤ 34 True

i = 2

1 < 34 True inc i

i = 3

45 < 34 false so we have to swap it a[j] to a[i]

27 > 34 false comes out from if (i < 8)

0	1	2	3	4	5	6	7	8
12	26	1	27	18	34	78	89	45

left than Pivot element

right part greater than Pivot. a[j]

if i < j

3 < 8 True swap a[i] & a[j]

0	1	2	3	4	5	6	7	8
34	26	1	(45)	18	78	12	89	(27)
			27					45

i < j

3 < 8 True body will executed

i < 34 True

18 < 34 true increment a[i]

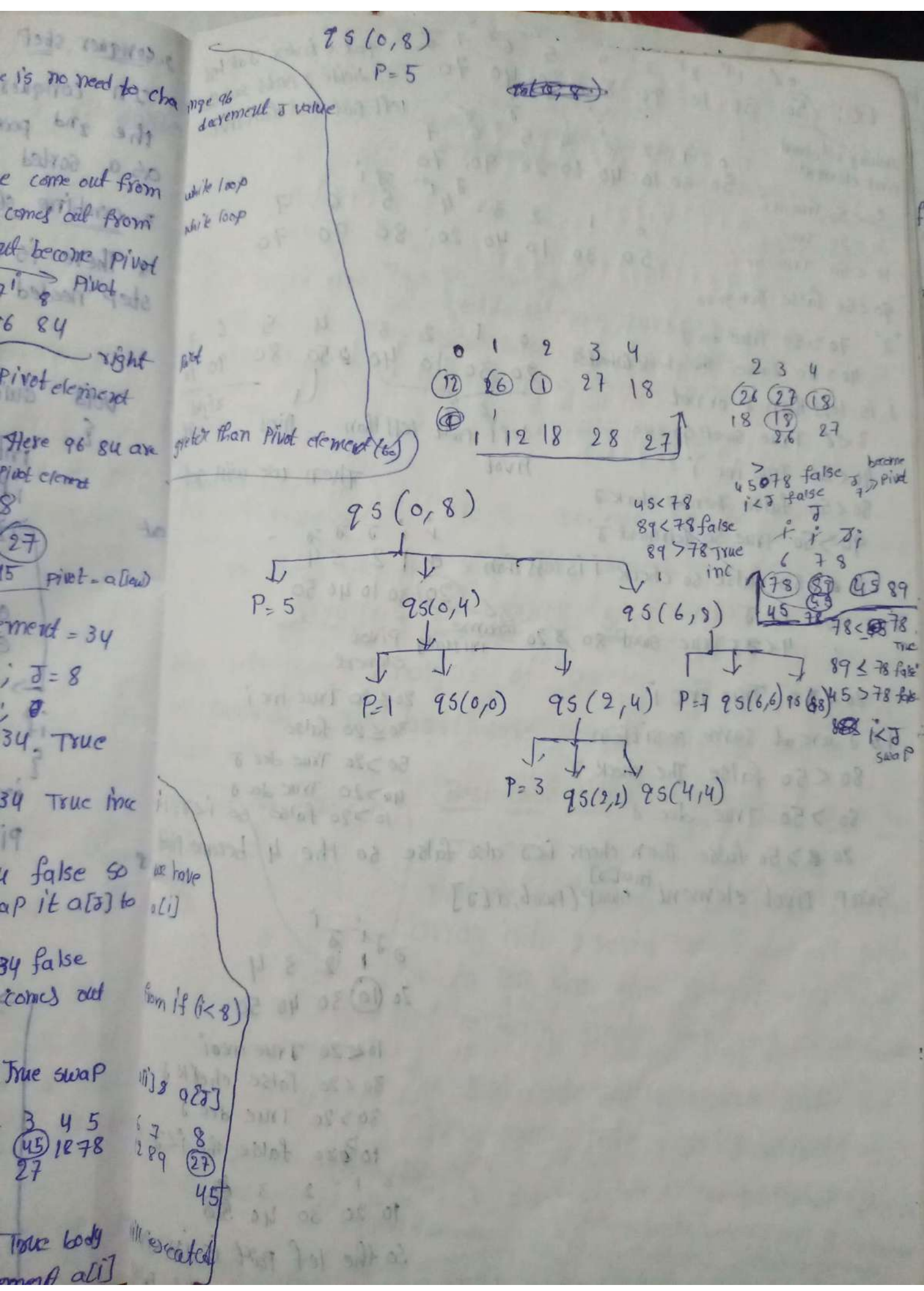
45 > 34 false

89 > 34

12 > 34

q (0, 5)

P = 5



qs(0,8)

P=5

~~qs(0,8)~~

change 96 decremeant j value

e come out from while loop

comes out from while loop

ot become Pivot

Pivot

6 84

right part

Pivot element

Here 96 84 are greater than Pivot element (6)

pivot elemnt

8

27

15 pivot = a[low]

ement = 34

; j = 8

; 0

34. True

34 True inc

i9

u false so we have

ap it a[j] to a[i]

34 false

comes out from if (i<8)

True swap i] & a[j]

3 4 5 6 7 8

45 1878 289 27

27 45

True body created

ment a[i]

Scanned with OKEN Scanner

Ex: 50 30 10 90 80 20 40 70

Initially 1st element pivot element

i at 0 index j at last

divide 2 parts such that left part less than Pivot right part >

0	1	2	3	4	5	6	7
50	30	10	40	80	20	90	70

50 < 50 True inc i

30 < 50 True

10 < 50 True inc i

90 < 50 false Then go to 'j' 70 > 50 True dec j

40 > 50 false The check i is less than j or not

3 < 6 True swap 90 & 40

40 < 50 True inc 'i'

80 < 50 false Then we check j

90 > 50 True so decrement 'j'

20 > 50 false so check i is less than j or not i < j

4 < 5 True swap 80 & 20

20 < 50 True inc i

i & j are at same position

80 < 50 false The check j

80 > 50 True dec j

20 > 50 false Then check i < j also false so the 4 become Pivot

Swap pivot element in a[j] swap(Pwot, a[j]

0	1	2	3	4	5	6	7
20	30	10	40	50	80	90	70

left part less than Pivot — Pivot — right part > Pivot

Then we will get

0	1	2	3	4
20	30	10	40	50

assume initially Pivot element

20 <= 20 True inc i

30 <= 20 false

50 > 20 True dec j

40 > 20 True dec j

10 > 20 false so i < j : 1 < 2 True

0	1	2	3	4
20	10	30	40	50

10 <= 20 True inc i

30 < 20 false check j

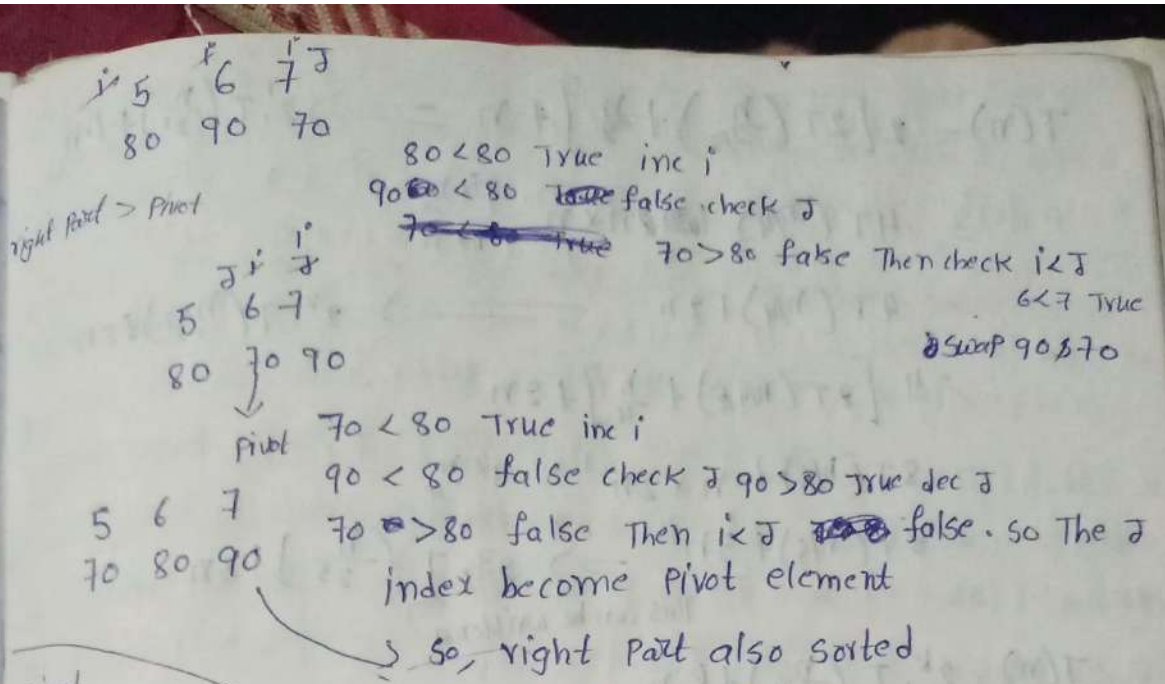
30 > 20 True dec j

10 > 20 false the i < j : 2 < 1 false

0	1	2	3
10	20	30	40

So the left part is in proper order

Now we Quic



10, 20, 30, 40, 50, 70, 80, 90,

In this way we sort the elements in quick sort.

Analysis of Quick sort

Now lets see Analysis of Quick sort. in this way we calculate the best case, worst case, avg case of the time comp Quick sort

Best case time complexity of Quick sort

* In quick sort first we select the pivot element, next we divide the given array into 2 parts such that left part contain less than pivot element wher right part contain greator than pivot element value
* The Best case will occur if we divide the given array into exactly 2 parts

Time comp by an array of elements $T(n) = 1$ → if there is 1 we get the soln directly → Then we no need to sort the element

→ if $n > 1$ → require Min (n) comp

Species divide $T(n) = 2T\left(\frac{n}{2}\right) + n$ → species each sub array contains $n/2$ elements

$\frac{1}{2}$ species divide into 2 parts

$T(n) = 2\left[2T\left(\frac{n}{2/2}\right) + \frac{n}{2}\right] + n \Longrightarrow 2^1 \cdot T\left(\frac{n}{2^1}\right) + 1 \cdot n$

$= 4T(n/4) + 2n \times n$ ($2 \times \frac{n}{2}$, $n \times n$)

$= 4T(n/4) + 2n \Longrightarrow 2^2 \cdot T(n/2^2) + 2n$

$= 4\left[2T(n/4_2) + \frac{n}{4}\right] + 2n$

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

$= 8T(n/8) + 3n \Longrightarrow 2^3 \cdot T\left(\frac{n}{2^3}\right) + 3n$

This can be written as

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

$= \text{let } 2^k = n$

$\log 2^k = \log n$

$K \log 2$

$K = \log n$

Substitute 'k' element

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

$= n \cdot T(1) + n \log n$

$= n \cdot 1 + n \log n$

$= n \log n$

Best Time complexity = $O(n \log n)$

Avg case

Here the avg case also divide the array into 2 parts such that the left part contains less than pivot element & right part contains the greater than the pivot element

* So In Quick Sort There is no difference b/w best case and average case

The best case & avg case Time complexity of Quick sort is = $O(n \log n)$

* In avg Th
elements whe
like that

* where In
elements.

* where In
worst case

* In Quick
in ascendi
are in sor

let we have

Then what i
Here no ele

The

* In avg There we have 10 elements The left part have 3 elements wher right part contains 7 elements or left 4 right 6 like that

* where In best case The left & right part have same no of elements.

* where In avg case some difference b/w them.

worst case Time complexity of Quick sort

* In Quick sort the worst case will occur if the elements in ascending order (or) descending order that means the elements are in sorted order when it is the worst case

let we have 5 elements 0 1 2 3 4 → having ascending order

(10) 20 30 40 50 → The logic of QS is to divide the two parts left part < pivot, right part > Pivot

Pivot element

Then what is the left part Here.

Here no elements which are < Pivot elem

so, if there are n elements

left part contain 0 elements ← 0

1 → right part n-1 elements

(20) → Pivot

0 n-2

0 n-3 → n-3 comp

n comps

n-2 comps

$1+2+3+4+5 \Rightarrow \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$

sum of n value nums

Ignor coefficient ↳ high oroder

$= O(n^2)$

The worst case time complexity of Quick sort $= O(n^2)$

Algorithm

Master's Theorem

* Master's Theorem is mainly used for finding the time complexity of given reccurance relation.
* It is most imp theorem for finding the time complexity of given reccurance relation.
* substitution method is also used for finding the time complexity of given recourance relation. but substitution method takes more time to finding the time complexity
* To overcome that disadvantage in substitution method we have to use the Master's theorem for finding the time complexity of given reccurance relation within a less time.

* Master's Theorem is a popular method for solving the recurance relations The recurance relation is in the form of

$$T(n) = a \cdot T(n/b) + \theta(n^K \log^P n)$$

(constants; suppose given recurrance is in the form. Then how to find out time complexity is discussed; 2nd we find a, b, P values)

where $a > 1$, $b > 1$, $K \geq 0$ and P is a real number

* compulsary given recuran relation is in this form then we have to find out time complexity using master's Theorem.

* Mater's Theorem, we compare 'a' with 'b^K', then we follow the following cases.

case 1:

if $a > b^K$, Then $T(n) = \theta(n^{\log_b^a})$

case 2:

if $a = b^K$ then

(i) if $P < 1$ then $T(n) = \theta(n^{\log_b^a})$

(ii) if P=

(iii) if P>

case-3:

if a

(i) if P<

(ii) if P

after comp

or not

* Now check

have to fi

* if $a < b^K$

* if $a = b^K$

if a>

* if find gi

is in form (

we comp

T(n) = a

a =

* check

Now

Clea

clear

(ii) if $P = -1$ then $T(n) = \Theta(n \log_b^a \cdot \log^{-1} n)$

(iii) if $P > -1$, then $T(n) = \Theta(n \log_b^a \cdot \log^{P+1} n)$

case-3 :

if $a < b^K$ then

(i) if $P < 0$, Then $T(n) = \Theta(n^K)$

(ii) if $P >= 0$, Then $T(n) = \Theta(n^K \log^P n)$

after comparing given recurance relation is in the form or not

* Now check what are the values of a, b, P & K. Now we have to find the relation a & b^K

* if $a < b^K$ → case 3

* if $a = b^K$ → case 2

if $a > b^K$ → case 1

1st find given relation is in form (or) not

Example-1 : solve the following Reccurance relation using master's theoroem.

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

we compare the given reccurance relation with $T(n) = a \cdot T(n/b) + \Theta(n^K \log^P n)$, we have

$a = 3, b = 2, K = 2$ and $P = 0$

* check the relation a and b^K

Now $a = 3$, $b^K = 2^2 = 4$

clearly $a < b^K$ because $3 < 2^2 \Rightarrow 3 < 4$

clearly we follow case 3.

since $P=0$, so we have use $P >= 0$ because $P=0$ and here

$$T(n) = \theta(n^K \log^P n)$$
$$= \theta(n^2 \log^0 n)$$
$$= \theta(n^2 \cdot 1)$$
$$= \theta(n^2)$$

$\therefore T(n) = \theta(n^2)$ → This is the time complexity

In this way we have to calculate time complexity by using masters theorem.

Example 2 : solve the following Reccurence Relation using Master Theorem (merge sort)

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

we compare the given reccurance relation with $aT(n) = a \cdot T(n/b) + \theta(n^K \log^P n)$, we have

$$a=2 \quad b=2 \quad K=1 \quad P=0$$

Now $a=2$, $b^K = 2^1 = 2$

clearly $a = b^K = 2 = 2^1 \Rightarrow 2 = 2$

clearly we follow case-2

since $P > -1$, so we have

$$T(n) = \theta(n \log_{2b}^{a} \cdot \log^{P+1} n)$$
$$= \theta(n \log_2^2 \cdot \log^{0+1} \cdot n)$$
$$= \theta(n^1 \cdot \log^1 \cdot n)$$

$= O(n \log n)$

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

closest pair of points

```
closest Pair (X, Y){
  n = X.length
  if (n==2) : return dist (X[1], X[2]);
  if (n==3) : return min ( dist (X[1], X[2]), dist (X[2], X[3]),
                           dist (X[1], X[3]));
  mid = X [n/2];
  d1 : Closest Pair (X[1...mid], Y);
  dr = Closest Pair (X [mid+1...n], Y];
  d = min (d1, dr)
  S = Points X whose x-coordinates are in the range of
  [mid.X-d, mid.X+d]
  for i=1 to s length
  for j=1 to j=7
    d = min (d, dist [S[i] S(i+j));
```

Unit - 2

Greedy Method

Knapsack problem using Greedy method

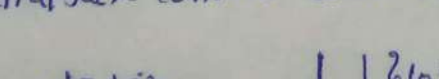
Let us assume that there are ('n') objects. Here each having a profit (p_i). each object which have weight also represented as (w_i)

Here we have knapsack here knapsack is ntg but bag & knapsack (or) bag whose weight is represented as 'm'

* ex m = 20 the size of knapsack (or) bag is 20

- The main objective of knapsack problem is to place the corresponding (or) all objects in the bag with the maximum profit
- Here optimal soln (or) best solution is represented with the help of knapsack vector is represented as x_i
- The value of x_i ranges from 0-1 $0 \leq x_i \leq 1$ min=0 max=1 blw some fractional values also

* if an object placed in the knapsack then we say $x_i = 1$
* if an object not placed in knapsack then we can say $x_i = 0$
* Kanapsack contain some in that space to place the object



* we can't place 30kg object on 10kg of the bag (or) knapsack
* Then x_i can be calculated as remaining size of knapsack by actual weight of object

$$x_i' = \frac{\text{Remain size of knapsack}}{\text{actual weight of object}}$$

Algorithm

1. calculate $\frac{p_i}{w_i}$ fo
2. Arrange all $\frac{p_i}{w_i}$
3. Place objec

problem

we have 5

5 objects
m = 100

profits (P1,

(w

1. Calculate

$\frac{P_1}{w_1}$ =

$\frac{P_4}{w_4} = \frac{40}{40}$ =

2. Arrange $\frac{p_i}{w_i}$

$\frac{P_3}{w_3}$

3. Placed

Algorithm

1. calculate $\frac{P_i}{w_i}$ for every object
2. Arrange all the objects in decreasing order based on $\frac{P_i}{w_i}$
3. Place objects in knapsack based on previous data.

Problem

we have 5 objects size of knapsack $m = 100$

5 objects
$m = 100$

profits $(P_1, P_2, P_3, P_4, P_5) = (20, 30, 66, 40, 60)$

$(w_1, w_2, w_3, w_4, w_5) = (10, 20, 30, 40, 50)$

1. Calculate $\frac{P_i}{w_i}$ for each object

$\frac{P_1}{w_1} = \frac{20}{10} = 2$, $\frac{P_2}{w_2} = \frac{30}{20} = 1.5$, $\frac{P_3}{w_3} = \frac{66}{30} = 2.2$,

$\frac{P_4}{w_4} = \frac{40}{40} = 1$, $\frac{P_5}{w_5} = \frac{60}{50} = 1.2$.

2. Arrange all the objects in decreasing order based on $\frac{P_i}{w_i}$

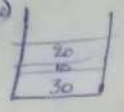
$$\frac{P_3}{w_3} > \frac{P_1}{w_1} > \frac{P_2}{w_2} > \frac{P_5}{w_5} > \frac{P_4}{w_4}$$

3. Placed objects in knapsack

$m = 100$

(P_3, w_3) $x_3 = 1$

(P_1, w_1) $x_1 = 1$

(P_2, W_2) $(x_2, 20)$ = 

$60 - 20 = 40$ x_2 size = 40, $x_2 = 1$

$(P_5, W_5) = (60, 50)$ we can not place 50 on 40 size bag, so

$x_5 = \frac{40}{50}$

$(P_4, W_4) =$ remaining size = 0 m - 0 bag is empty

$x_4 = 0$

Nxt our target is Profit.

Profit = $\sum P_i x_i$ => $20*1 + 30*1 + 66*1 + 40*0 + 60*4/5$ (12)

=> $20 + 30 + 66 + 48$

=> 164

So we can say that 164 is profit for this problem.

Job sequencing with deadlines

Assumes n' Jobs. Here each Job having a profit (P_i) as well as each Job has a deadline (D_i)

* we can earn profit if and only if we have completed the Job by its deadline (D_i)
* Inordere to complete the Job execution we have two constraints

1. Onemachine → only one machine inorder to execute all the Jobs
2. 1 unit of time → machine require 1 unit of time inorder to complet the exeation of Job

→ Inorder to complet 1st Job The machine require 1 unit of time

→ inorder to complete the 2nd Job execution machine machine requires 1 unit of time like that

* target is complete all the Jobs with maximum profit. by its Deadline

Algorithm
1. Arrange
profit
2. Select
slot

Ex:-

Algorithm

1. Arrange all the Jobs in decreasing order based on profit
2. select the 1st Job & execute it by assigning a slot

slot

, our target is need to find optimal solution, best solution.

Ex:- 5 Jobs

$(P_1, P_2, P_3, P_4, P_5) = (100, 19, 36, 27, 52)$

$(d_1, d_2, d_3, d_4, d_5) = (2, 1, 2, 1, 3)$

[2-2, 2-1] = [0, 1]

[3-1] [2-1, 2] = [1, 2]

decreasing order

$(P_1, P_5, P_3, P_4, P_2) = (100, 52, 38, 27, 19)$

$(d_1, d_5, d_3, d_4, d_2) = (2, 3, 2, 1, 1)$

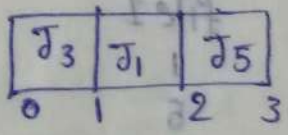
highest so 3 slots

Assigning all the Jobs to the correspond machine. machine assign a slot to them and executes 1 by 1

[3-1, 3]

[2, 3]

[r-1, r] = [2-1, 2] = [1, 2]



Assigned slot	Job selected	Action	profit
none	J1	[1,2]	100
[1,2]	J5	[2,3]	152
[1,2], [2,3]	J3	[0,1]	190
[0,1], [1,2] [2,3]	J4	rejected	190
[0,1], [1,2], [2,3]	J2	reject it	190

The Job sequence = $\{J_1, J_5, J_3\}$
= 100 + 52 + 38
= 190

minimum cost spanning tree

1. 'n' vertices → spanning tree, minimum spanning, Prims Alg, Kruskal's Alg } - on rough book
2. 'n-1' edges
3. no cycle

optimal merge patterns using Greedy method

The main Aim of this method is

* let there 'n' files
* each may file contain some no. of elements

→ Here our target combine all these files into 1 file with minimum num of comparisions

→ combining ntg but Merging

* let us assume we have 2 files are there

file 1 → contain 4 elements
5
10
15
↓ sorted order

file 2 → contain 5 elements
3
4
13
20
30
↳ sorted order

* Then only we Perform to Merging.
* otherwise 2st we need to Sorting the elements and the we need to Merging

* Here our target merging two files into another file (file3) for that the logic is simple.

* If
of 1st
* initial
out of
to the
* cost
* The
Next

pointer
5 & 4
pointer
5 & 10
pointer
10 & 13
The poin
check
0

Here
comPar
Thecomp
* In

In

=)

initially the pointer to be points to the 1st element of 1st file as well as 1st element of 2nd file

initially the pointer at 3rd file. so 1 & 3 are compared out of 1 & 3 the smaller element is 1. The 1 will be Moved to the resultant file

pointer →	file 1	file 2	file 3
	→ 1	→ 3	→ 1

The operation on 1 is over, so Move the pointer to the next location 5 in file also place pointer to next

	file 1	file 2	file 3
5 & 3 → 3 Moved	1	~~3~~	1
pointer comes to 4	~~5~~	~~4~~	~~3~~
5 & 4 → 4 Moved			~~4~~
pointer comes to 10	→ 10	~~13~~	~~4~~
5 & 10 → 5 Moved	→ 15	20	→ 5
pointer comes to			
10 & 13 → 10 Moved opn 10 over		10 → 20	
The pointer comes to 15			
check 15 & 13 → 13 small Moved			

file 1	f2	file 3
~~1~~	→ 3	~~1~~
~~5~~	~~4~~	~~3~~
	10 → 13	→ 4
15	20	5
↓	30	10
1st file over		13
	directly moved to 3	15
		20
		30

Here file 1 size 4 file 2 size (5) in worst case we require 4+5-1 comparisions

m n

The comp w.c = $O(m+n)$

* In this way we perform merging on two files

Ex:

A	B	C
10	30	50

In First way first we Merge AB and Next Merge BC

A	B	C
10	30	50

40 comp

90 comparisions

=) 130 comparisions

Ex: In second way

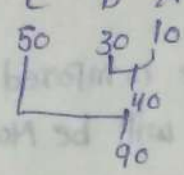
A	B	C
10	30	50

80

90

=) 170 → comparisions

In third way (or) some other order



=) 130 comp

=> Here we have 3 files. like wise we find all these feasible solutions and from that we can set 1 solution as the best solution There is No problem.

* instead of that we have 10 files we have 15 files we find all the feasible solutions is extremly difficult.
* How to find best & optimal solution we have No. of files there

ex: $(x_1, x_2, x_3, x_4, x_5, x_6)$

(15, 7, 8, 12, 13, 4)

Step 1. we have to arrange all this elements in ascending order

2 step : Merge two element we will get 1 more element. after get these 2 elements we will get some resultant that element should be in the list only (i.e 5+7 = 11 -> placed in same list)

3 step : Repeat step 2 as long as we will get only one element

1. 4, 7, 8, 12, 13, 15

11

8 11 12 13 15

19

12 13 15 19

25

15 19 25

34

25 34

= 59

this problem

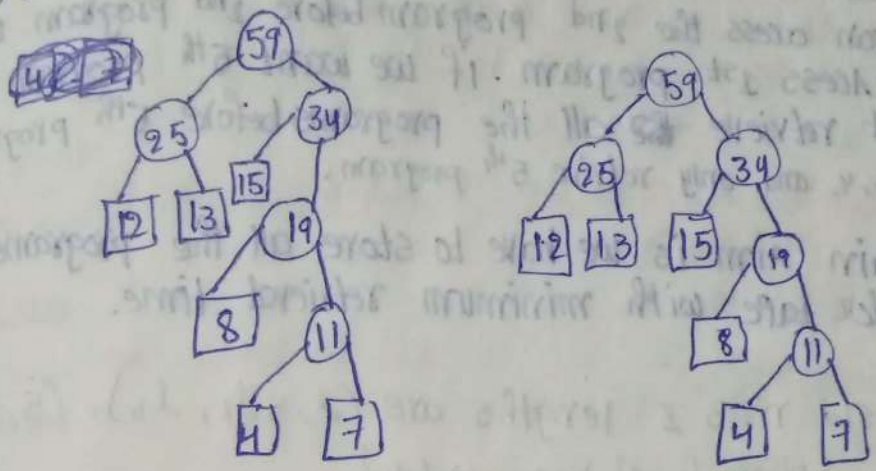
all the elements are leaf node

if there is a leaf node ▭

if there is a parent node should be place in ○

Here we binary search tree conventions

1) left part contain less than root value
2) Right sub tree contain greater than root value
3) Each node can have max of 2 childrens. (ie 0, 1, 2 childrens)



Total no. of merges

we have 2 formulas 1) sum of internal nodes

11 + 19 + 25 + 34 + 59

= 148 → Mergings (or) comparisions . inorder to Merge all these file into 1 file

Based on the leaf nodes also we can calculate total no. of merges that are needed.

formual - $\sum d_i, q_i$ → that leaf node [7]

↓ distance from root node (59) to leaf node like [7]

↓ no. of edges

$4*4 + 4*7 + 3*8 + 2*12 + 2*13 + 2*15$

$= 16 + 28 + 24 + 24 + 26 + 30$

$= 148$

So, we can calculate Merges Based upon internal nodes and leaf nodes also. This is about optimal Merge Pattern.

optimal storge on tapes using Greedy Method

assume n programs are there & we need to store all those programs on computer tape

* let l_i = length of each program is denoted by l_i

-> tap is Mtg Magnetic tape it provide sequential Acess. So that means we retrieve the program we can acess the programs one by one

-> we can acess the 1st program directly

-> we want can acess the 2nd program before 2nd program we must Acess 1st program. if we want 5th Program then we must retrieve all the programs before 5th program i.e 1, 2, 3, 4, and only retrive 5th program.

* our Main Aim is we have to store all the programes on a computer tape with minimum retrieval time.

Ex: consid n=3 & lengths are $(l_1, l_2, l_3) = (5, 10, 3)$ find the optimal storage order!

$l_1 \to 5$
$l_2 \to 10$
$l_3 \to 3$

3! = 6 Possibilies

order	Retrieval Time	Total Rational time
1, 2, 3	5 + (5+10) + (5+10+3)	38
1, 3, 2	5 + (5+3) + (5+3+10)	31
2, 1, 3	10 + (10+5) + (10+5+3)	43
2, 3, 1	10 + (10+3) + (10+3+5)	41
3, 1, 2	3 + (3+5) + (3+5+10)	289 → minimum value
Best solution 3, 2, 1	3 + (3+10) + (3+10+5)	34

* instead possibilities
* so if we difficult
* then the opti
* in orde greedy
n=10

first are
In

Tape

Ta

To

In
sto

instead of 3 programs there are 10 programs then How many possibilities will occur 10! the value is very larger so if we have more no. of programs then it is extremly difficult inorder to find out all the feasible solns then what we have to do is directly we have to find the optimal (or) best soln

in order to find out optimal (or) best solution we using greedy method

$n = 10, l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8, l_9, l_{10}$ → 3 tapes let us assume

10, 20, 45, 70, 1, 3, 7, 54, 23, 67

first arrange

In asscending order :- 1, 3, 7, 10, 20, 23, 45, 54, 67, 70

				Retrival time
Tape 0	1	10	45	70	1+(1+10)+(1+10+45)+(1+10+45+70) = 194
Tape 1	3	20	54		3+(3+20)+(3+20+54) = 103
Tape 2	7	23	67		7+(7+23)+(7+23+67) = 134
					431

Total Retrival time => 194+103+134

mean Retrival time = $\frac{431}{3} = 143$.

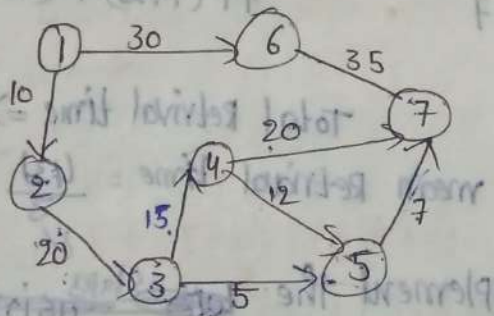
In this way we implement the ~~tapes using~~ 3 tapes optimal storage on tapes using greedy Method.

Single Source Shortest Path Algorithm

We going to discuss about single source shortest Path Algorithm

* In order to solve this problem we are using an Algorithm called Dijkstra's Algorithm
* The Aim of Dijkastra's Algorithm. let us assume that we have a graph called $G = (V, E)$
* let the graph is a directed ~~graph~~ weighted graph
* directed graph means There should be some directions from source to destination
* The Main Aim is we have to select one vertex as source vertex
* we have to assume or select one vertex as source and we need to find out the shortest path to source ~~and~~ vertex to remaining all the vertices of the Graph

Ex:



sorted Vertex	visited set	d[2]	d[3]	d[4]	d[5]	d[6]	d[7]
1	{1}	(10)	∞	∞	∞	30	∞
2	{1,2}	(10)	(30)	∞	∞	30	∞
3	{1,2,3}	(10)	(30)	45	35	(30)	∞
6	{1,2,3,6}	(10)	(30)	45	(35)	(30)	65
5	{1,2,3,6,5}	(10)	(30)	45	(35)	(30)	(42)
7	{1,2,3,6,5,7}	(10)	(30)	No edge 45	(35)	(30)	(42)
4	{1,2,3,6,5,7,4}	(10)	(30)	(45)	(35)	(30)	(42)

sorted Vertex	cost
2	10
3	30
4	4
5	3
6	
7	

selected Vertex

1
3
5
2
4

Sorted Ver

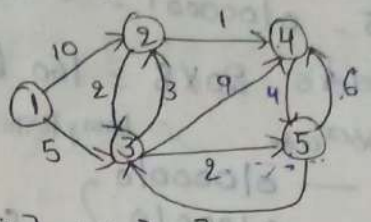
Traffma

* it is
* The
* 2nd
a fil
then

sorted vertex	cost
2	10
3	30
4	45
5	35
6	30
7	42

EX2 : assume that $G = (V, E)$ a directed (set of edges, set of vertices) weighted graph

* one vertex as source vertex and we need to findout the shortest path to source vertex to remaining all the vertices.



selected vertex	visited set	d[2]	d[3]	d[4]	d[5]
1	{1}	10	5	∞	∞
3	{1,3}	8	5	14	7
5	{1,3,5}	8	5	13	7
2	{1,3,5,2}	8	5	9	7
4	{1,3,5,2,4}	8	5	9	7

sorted vertex	cost
2	8
3	5
4	9
5	7

Huffman Coding

* it is a data compression technique
* The size of the data (or) msg are to be reduced
* 2nd we transfer something on network. if we send a file to sender to reciever that file (or) msg compressed then obiviously it take less cost, (or) bytes

* charcter (A,B,C) - 01000001 → binary from → size 8 bits

* for single character transfer to use 8 bits

Ex:- ABBCDBCCDAABBEEEBEAB = 20 character

single char A = 65 - 01000001 = 8 bits → it is a fixed length coding

20 characters 20X8 = 160 bits

ASCII code → binary form

B = 66 — 01000010
C = 67 — 01000010
D = 68 — 01000010
E = 69 — 01000010

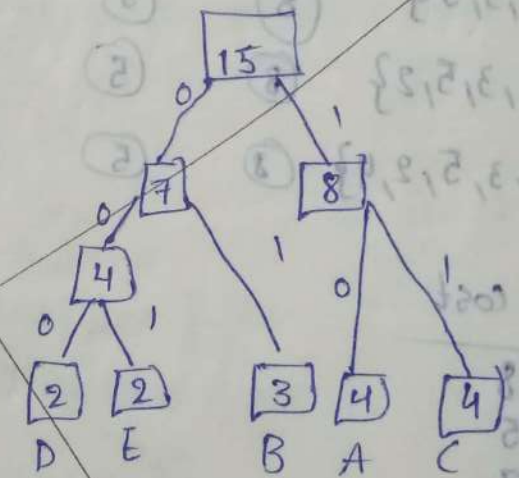
=> Each charact having fixed length coding = 8 bits.

Suppose
00
01
10
11
=> 4 Poss bits (2²)

3bit $2^3 = 8$ Poss bits

8bits $2^8 = 256$ Poss

Char	
A	4 → 10
B	3 → 01
C	4 → 11
D	2 → 000
E	2 → 001



4X2 = 8
3X2 = 6
4X2 = 8
2X3 = 6
2X3 = 8

8+6+8+6+6 = 34
ms-g - 34 bits
code - 12 bits
char 5X8 = 40

34
12
40
86 bits

A C B A
10 01 01 10
A C B A

000 - A
001 - B
010 - C
011 - D
100 - E
1001
110
111

Char
A
B
C
D
E

* The
* The order
2
D
* NOW
(or)

3 bits for each char → in huffman coding fixed length i.e 3 bits but variable length coding A are 3 Bare 2 like that.

000 - A
001 - B
010 - C
011 - D
100 - E
101 } left
110 }
111

20 x 3
= 60 bits

Suppose 5 char
5 x 3
= 15 bits

A B B C D B C C D A A B B E E E B E A B ?

char	frequency/count
A	4
B	7
C	3
D	2 → least
E	4

huffman

* The first step is we have to design a tree using huffman Algo
* The range all the characters according to their increasing order of their count.

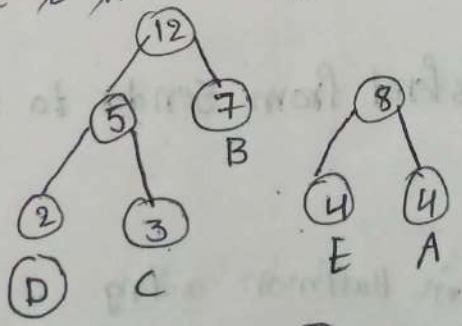
D C E A B
2 3 4 4 7

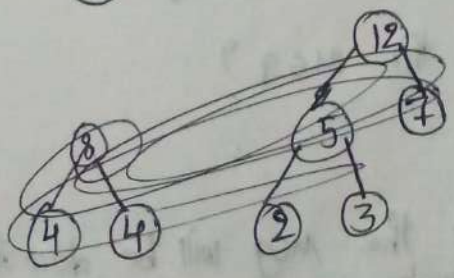
* Now taking two characters which having least ~~characters~~ frequency (or) count.

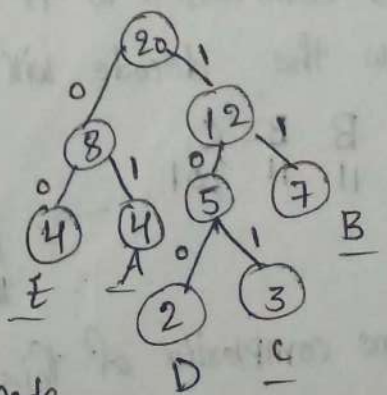
D C E A B
2 3 4 4 7

8 less than 12, it must be in left side

* left side min frequency
* right side max frequency

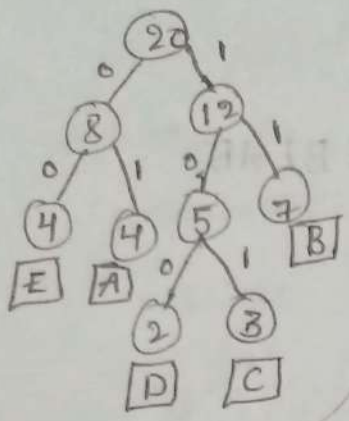






* The characters allway be in the leaf node
* 2nd step is we have to traverse this tree and assign 0 & 1

* 0 would be assigned to left side
* 1 would be assigned to right side.



char	frequency/count	(how many times A repeated)
A	4	01
B	7	11
C	3	101
D	2	100
E	4	00

Ex 2 prefix code :-

A - 0
B - 1
C - 01

Huffman

-> David
-> encodin
-> Most least
-> Time

* The letter of the highest count that character having huffman code is lesser

Huffman code is less than

A (1 → no of bits) B C (3 → no of bits) D

$4 \times 2 + 7 \times 2 + 3 \times 3 + 2 \times 3 + 4 \times 2$

$= 8 + 14 + 9 + 6 + 8$

$= 45$ bits

ASCI code - 5 (charact A,B,C,D,E) × 8 (8 bit binary form) → each char would be represented in 8 bit code in ASCI code

$= 40$

$= 40 + 12$ → 12 bits are there

$= 52$

01 11 101 100
A B C D → like that msg will be sended and they are recieve

The total bits to be transfered from sender to Reciever would be $45 + 52$

$= 97$ bits

116 bits compressed to 97 bits in Huffman coding

* How the decode will decode the msg ?

A B B C
01 11 11 101

→ like thise the msg will be decode with this huffman tree

Time complexity of this would be $O(n \log n)$

prefix code :- No code is prefix of another code

A - 0
B - 1
C - 01

suppose sender to reciever a code

001

0 0 0 01
A A B A C

code A → 0 → Prefix of code C
01
Prefix

Huffman coding

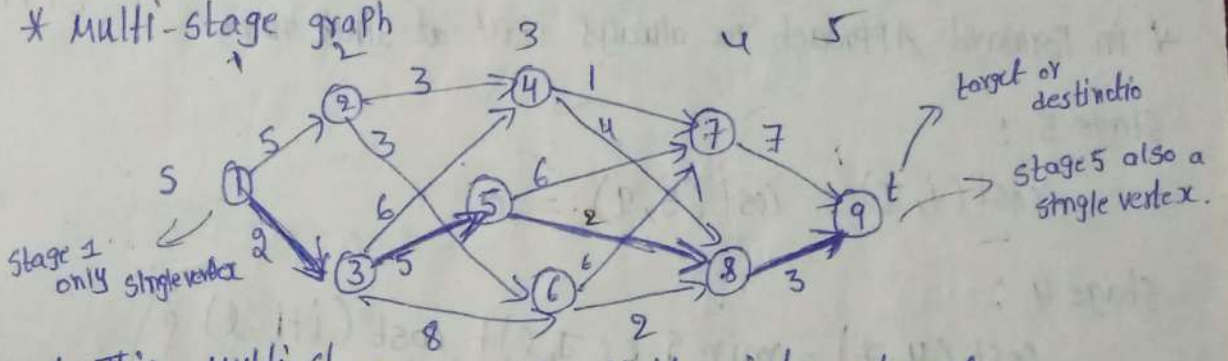
-> David Huffman in 1951
-> encoding follows the prefix rule
-> Most generated character will get the small code & least generated character will get the large code
-> Time complexity O(nlogn)

3. Dynamic programming

multistage Dynamic programming

* Multistage graph is one of the Application of Dynamic programming
* Dynamic programming is an Algorithm designed method
* it is used for solving optimization problems.
* optimization problem is nothing but a problem may require a maximum result or minimum result. it depends on the given problem

* Multi-stage graph



* This multi-stage graph divide into stages.
* we have to find out minimum cost path from source vertex s to destination vertex.

stage 1 | stage 2 | stage 3 | stage 4 | stage 5

* The minimum cost from s to t in multi-stage graph is calculated by using two Approaches.
  1) Forward Approach
  2) Backward Approach

difference of forward & backward.

* in forward Approach we always starts at sink node (t) Again to travel to back at source node.
* in Backward Approach we starts at source vertex again we have to move to target vertex.

* 1st we have to do forward Approach by using forward approach How to find the minimum cost path from source node s to target node t.

* Forward approach :-

In forward approach we find out the minimum cost by using cost of (i, j) (i → stage no., j → vertex number)

$$cost(i,j) = \min_{l \in V_{i+1}, (j,l) \in E} \{ c(j,l) + cost(i+1,l) \}$$

$D(i,j) = l$ (l is a node) that minimizes $c(j,l) + cost(i+1,l)$

* in Forward Approach we always start at single node.

Stage 5 :

$$cost(i,j) = cost(5,9) = 0$$

Stage 4 :

$$cost(4,7) = \min\{ c(j,l) + cost(i+1,l) \}$$
$$= \min\{ c(7,9) + cost(4+1,9) \}$$
$$= \min\{ 7 + cost(5,9) \}$$
$$= \min\{ 7+0 \} = \min\{7\} = 7$$

$D(4,7) = 9$ (value)

$$cost(4,8) = \min\{ c(8,9) + cost(4+1,9) \}$$
$$= \min\{ 3 + cost(5,9) \}$$
$$= \min\{ 3+0 \}$$
$$= \min\{3\} = 3$$

D(4,8)

Stage

cost

Stage 3 :-

$$cost(3,4) = \min\{ c(4,7) + cost(3+1,7),\ c(4,8) + cost(3+1,8) \}$$

$$= \min\{ 1 + cost(4,7),\ 4 + cost(4,8) \}$$

$$= \min\{1+7, 4+3\}$$

$$= \min\{8,7\}$$

$$= \min\{7\}$$

$D(3,4) = 8$ (l value)

$$cost(3,5) = \min\{ c(5,7) + cost(3+1,7),\ c(5,8) + cost(3+1,8) \}$$

$$= \min\{6+7, 2+3\}$$

$$= \min\{5\}$$

$D(3,5) = l = 8$

$$cost(3,6) = \min\{ c(6,7) + cost(3+1,7),\ c(6,8) + cost(3+1,8) \}$$

$$= \min\{ 6 + cost(4,7),\ 2 + cost(4,8) \}$$

$$= \min\{6+7, 2+3\}$$

$$= \min\{5\}$$

$D(3,6) = l = 8$

Stage 2 :-

$$cost(2,2) = \min\{ c(2,6) + cost(2+1,6),\ c(2,4) + cost(2+1,4) \}$$

1st, 2nd

$$= \min\{3+7, 3+5\}$$

$\min = \{8\}$

$D(2,2) = l = 6$

$$cost(2,3) = \min\{\, c(3,4) + cost(3+1, 4),\ c(3,5) + cost(3+1, 5),\ c(3,6) + cost(3+1, 6) \,\}$$

$$= \min\{\, 6+7,\ 5+3\ 5,\ 8+5 \,\}$$

$$= \min\{13, 10, 13\} = 10$$

$D(2,3) = 5$ (l value)

stage 1

$$cost(1,1) = \min\{\, c(1,2) + cost(1+1, 2),\ c(1,3) + cost(1+1, 3) \,\}$$

$$= \min\{\, 5 + cost(2,2),\ 2 + 3\ cost(2,3) \,\}$$

$$= \min\{\, 5+8,\ 2+10 \,\}$$

$$= \min\{12\}$$

$D(1,1) = \min(l) = 3$

$D(2,3) = 5$

$D(3,5) = 8$

$D(4,8) = 9$

$1-3-5-8-9 = 12$

$2+5+2+3 = 12$ ✓

optimal

Approach

2) Backwar

The m

calculate

C

* Ba

Ver

* fr

2nd Path

1st

S

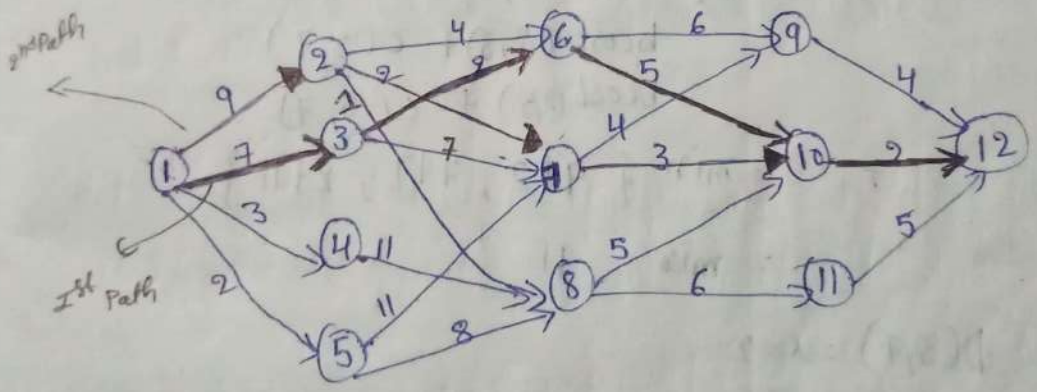
optimal (or) minimum shortest path using backward Approach in multistage graph in Dynamic programming

1) Backward Approach

The minimum shortest path in backward Approach is calculated by using the following formula

$$cost(i,j) = \min \{ bcost(i-1,l) + c(l,j) \}$$
$$\langle l,j \rangle \in E$$
$$l \in V_{i-1}$$
$$D(l,j) = l$$

* Backword Approach starts with source vertex to sink vertex
* from source vertex to source vertex cost is 0 $bcost(1,1) = 0$



stage 1

$b\,cost(1,1) = 0$

stage - 2

$$b\,cost(2,2) = \min \{ bcost(i-1,l) + c(l,j) \}$$
$$= \min \{ bcost(1,1) + c(1,2) \}$$
$$= \min \{ 0 + 9 \}$$

$$b\,cost(2,3) = \min \{ b\,cost(i-1,l) + c(l,j) \}$$
$$= \min \{ b\,cost(1,1) + c(1,3) \}$$
$$= \min \{ 0 + 7 \} = 7$$

bcost (2,4) = min { bcost (1,1) + c (1,4) }
= min { 0 + 3 }
= 3

bcost (2,5) = min { bcost (1,1) + c (1,5) }
= min { 0 + 2 }
= 2

D ⟶ 2

step 2:

bcost (3,6) = min { bcost (2,2) + c (2,6)
bcost (2,3) + c (3,6) }

= min { 9 + 4, 7 + 2 }
= 9

D (3,6) = 3

bcost (3,7) = min { bcost (2,2) + c (2,7)
bcost (2,3) + c (3,7)
bcost (2,5) + c (5,7) }

= min { 9 + 2, 7 + 7, 2 + 11 }
= min = 11

D (3,7) = 2

bcost (3,8) = min { bcost (2,2) + c (2,8)
bcost (2,4) + c (4,8)
bcost (2,5) + c (5,8) }

= min { 9 + 1, 3 + 11, 2 + 8 }
= min { 10 }

D (3,8) = 2 or 5

stage 4

bcost (4,9) = min { bcost (3,6) + c(6,9)
bcost (3,7) + c (7, 9) }

= min { 9+6 , 11+4 }

= min = 15

D (4,9) = l = 6 (or) 7

bcost (4,10) = min { bcost (3,6) + c (6,10)
bcost (3,7) + c (7, 10)
bcost (3,8) + c (8, 10) }

= min { 9+5 , 11+3 , 10+5 }

= 14

D(4,10) = l = 6 or 7

bcost (4,11) = min { bcost (3,8) + c (8,11)

= min { 10+6 }

= 16

D(4,11) = l = 8

step-4

stage 5

bcost (5,12) = min { bcost (4,9) + c(9,12)
bcost (4,10) + c (10,12)
bcost (4,11) + c (11,12) }

= min { 15+4 , 14+2 , 16+5 }

= min = 16

D(5,12) = l = 10

Now we have to find minimum class path

$D(1,1) = 1$

$D(5,12) = 10$

$D(4,10) = 6$

$D(3,6) = 3$

$D(2,3) = 1$

$D(5,12) = 10$

$D(4,10) = 7$

$D(3,7) = 2$

$D(2,2) = 1$

$1-2-7-10-12$

$1-3-6-10-12$

$= 7+2+5+12$

$= 16$

$9+2+3+2$

$= 16$

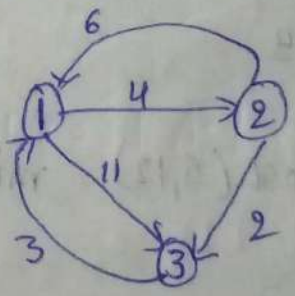
∴ we have two Alternative Paths

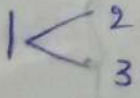
For this graph by using backward Approach there are 2 minimum costs paths are there

All pairs shortest path Algorithm using Dynamic programming

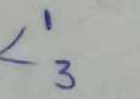
* The main Aim of this Algorithm is we need to find shortest path between Every pair of vertices

Ex If we consider 1st vertex we need to find shortest Path. from 1st vertex to 2 (or) 3 find short

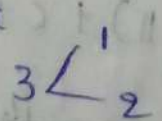




Ex



Ex

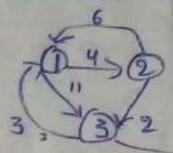


* First we need to construct Adjacency matrix
* Here the Adjacency Matrix represented with the help of A^0
* grap contain 3 vertices so we have to calculate A^1, A^2, A^3
* Suppose graph contains 5 vertices we have to calculat A^1, A^2, A^3, A^4, A^5

* Dynamic Programming is implemented based upon a series of Actions
* Here calculating several matrices (graphs) is Nothing but actions
* first we calculate A^0, A^0 is Ntg but Adjacency matrix.

$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix} \end{array}$$

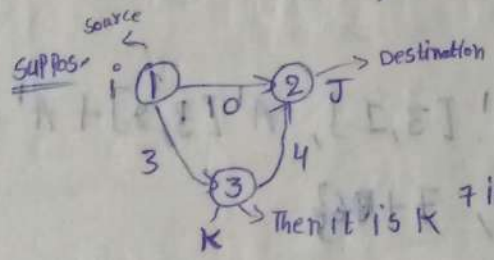
→ it is a 3/3 Matrices. because graph contain 3 vertices



3 to 2 No direction to ∞

* A^1.
* Inorder to calculate a matrix we have to use a formula

$$A^K[i,j] = \min\{A^{K-1}[i,j], A^{K-1}[i,K] + A^{K-1}[K,j]\}$$



7 is minimum value, i.e a intermediate vertex.

$$A^1 = \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & \\ 3 & & 0 \end{bmatrix} \quad \begin{matrix} \to 2 \\ \\ \to 7 \end{matrix}$$

$$A^K[2,3] = \min[A^{K-1}[2,3], A^{K-1}[2,1] + A^{K-1}[1,3]\}$$

$$= \min[A^0[2,3], A^0[2,1] + A^0[1,3]\}$$

$$= \min\{2,$$

$$A^1[2,3] = \min[A^{0}[2,3], A^{K-1}[2,1] + A^0[1,3]\}$$

$$= \min[A^0[2,3], A^0[2,1] + A^0[1,3]\}$$

$$= \min\{2, 6+11\}$$

$$= \{2, 17\}$$

$$= 2$$

$A^1[3,2] = \min\{A^0[3,2], A^0[3,1] + A^0[1,2]\}$

$= \min\{\infty, 3+4\}$

$= 7$

$$A^2 = \begin{bmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

$A^2[1,3] = \min\{A^1[1,3], A^1[1,2] + A^1[2,3]\}$

$= \min\{11, 4+2\}$

$= 6$

$A^2[3,1] = \min\{A^1[3,1], A^1[3,2] + A^1[2,1]\}$

$= \min\{3, 7+6\}$

$= 3$

$$A^3 = \begin{bmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

$A^3[1,2] = \min\{A^2[1,2], A^2[1,3] + A^2[3,2]\}$

$= \min\{4, 6+7\}$

$= 4$

$A^3[2,1] = \min\{A^2[2,1], A^2[2,3] + A^2[3,1]\}$

$= \min\{6, 2+3\}$

$= 5$

This is
* After
wheath
the gra

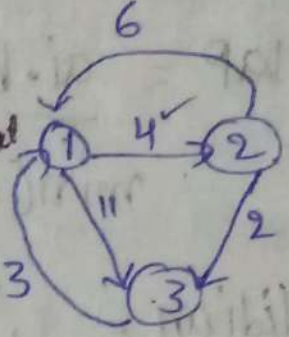
$A^3 =$

This is the final matrix

* After calculating the final matrix we have to check whether our values are correct (or) not. by checking with the graph.

$$A^3 = \begin{bmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

1 to 2, 2 to 3
4 + 2 = 6
So we can conclude that the 1st row is correct.



2 to 1 go u
2 to 3, 3 to 1
2 + 3 = 5

☆ $Sim_j(a,b) = \frac{a_i \cdot b_i}{a_i^2 + b_i^2 - a_i \cdot b_i}$

$Sim_j = \frac{a_i \cdot b_i}{a_i^2 + b_i^2}$

☆ $Sim_D(a,b) = 2 \cdot \frac{\sum_{i=1}^{d} a_i \cdot b_i}{a_i^2 + b_i^2}$

$Sim_O = \frac{2(a_i b_i}{a_i^2 + b_i^2}$

☆ consine similarity $= \frac{a_i \cdot b_i}{\sqrt{a_i^2 \cdot b_i^2}}$

$sim_C = \frac{a_i \cdot b_i}{\sqrt{a_i^2 \cdot b_i}}$

☆ overlap $= \frac{a_i \cdot b_i}{\min(a_i^2, b_i^2)}$

$sim_O = \frac{a_i \cdot b_i}{\min(a_i^2, b$

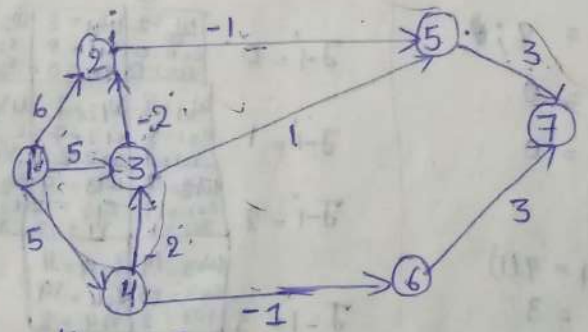
$dist(a,b) = \sqrt{(a_i - b_i)^2}$

Scanned with OKEN Scanner

Single source shortest Path Algorithm using Dyna Programing (or) Bellman ford Algorithm

* single source shortest path using greedy method that Algorithm is called as dijkastra's Algorithm → 2nd reason
* but the major problem with that Approach is .disjkastra Algorithm does not allow Negetive Edges (ie) it doesn't allows the Edges with Negetive cost
* we can overcome this Problem with the help of Dynamic Approach
* what is single source shortest path ?

-> The name itselfs specifies meaning. Here we have to select one vertex as source vertex and we need to find shortest path to remaining all the verticeses of a graph



dist K [1:7]

no. of edges

→ vertices

K	1	2	3	4	5	6	7
1	0	6	5	5	∞	∞	∞
2	0	3	3	5	5	4	∞
3	0	1	3	5	2	4	7
4	0	1	3	5	0	4	5
5	0	1	3	5	0	4	3
6	0	1	3	5	0	4	3

Optimal Binary search Tree using Dynamic programmin

* it mean A Binary search Tree which is optimal
* optimal means A Binary search Tree which produces least no. of comparisons.

Ex: $n=4$ $(a_1, a_2, a_3, a_4) = (do, if, int, while)$

$P(1:4) = (3,3,1,1)$

$q(0:4) = (2,3,1,1,1)$

Formula :

1st formula : $w(i,j) = w(i,j-1) + P(j) + q(j)$

2) $c(i,j) = \min_{i<k\le j} \{ c(i,k-1) + c(k,j) \} + w(i,j)$

3) $r(i,j) = k$

initail conditions

$w(i,j) = q(i)$

$c(i,i) = 0$

$r(i,i) = 0$

$w_{00} = q_i = o_2$ | $w_{11} = q(1) = 3$

j-i	0	1	2	3	4
j-i = 0	$w_{00}=2$, $c_{00}=0$, $r_{00}=0$	$w_{11}=3$, $c_{11}=0$, $r_{11}=0$	$w_{22}=1$, $c_{22}=0$, $r_{22}=0$	$w_{33}=1$, $c_{33}=0$, $r_{33}=0$	$w_{44}=1$, $c_{44}=0$, $r_{44}=0$
j-i = 1	$w_{01}=8$, $c_{01}=8$, $r_{01}=1$	$w_{12}=7$, $c_{12}=7$, $r_{12}=2$	$w_{23}=3$, $c_{23}=3$, $r_{23}=3$	$w_{34}=3$, $c_{34}=3$, $r_{34}=4$	
j-i = 2	$w_{02}=12$, $c_{02}=19$, $r_{02}=1$	$w_{13}=9$, $c_{13}=12$, $r_{13}=2$	$w_{24}=5$, $c_{24}=8$, $r_{24}=3$		
j-i = 3	$w_{03}=14$, $c_{03}=25$, $r_{03}=2$	$w_{14}=11$, $c_{14}=19$, $r_{14}=2$			
j-i = 4	$w_{04}=16$, $c_{04}=32$, $r_{04}=2$				

$w(0,1) = w(0,0) + P(1) + q(1)$

$= 2 + 3 + 3$

$= 8$

$c(0,1) = \min_{0<k\le 1} \{ c(0,0) + c(1,1) + w(0,1) \}$

$k=1$ $\quad 0 + 0 + 8$

$= 8$

$r(i,j) = k$ $\boxed{k=1}$

$= 1$

$w(1,2) : w(1,1) + P(2) + q(2)$

$= 3 + 3 + 1$

$= 7$

$c(1,2) = \min_{1 < k \le 2} \{ c(1,1) + c(2,2) \} + w(1,2)$

$\boxed{k=2}$ $= 0 + 7$

$= 7$

$r(i,j) = k$

$= 2$

$w(2,3) = w(2,2) + P(3) + q(3)$

$= 1 + 1 + 1$

3

$c(2,3) = \min_{2 < k \le 3} \{ c(2,2) + c(3,3) \} + w(2,3$

$k=3$ $\min\{0+0\} + 3$

$= 3$

$r(i,j) = k$

$= 3$

$w(3,4) = w(3,3) + P(4) + q(4)$ $w(3,3) + P(3) + q(3)$

$= 1 + 1 + 1$

$= 3$

$c(3,4) = \min_{3 < k \le 4} \{ c(3,3) + c(4,4) \} + w(3,4)$

$k=4$

$= \min\{0+0\} + 3$

$r(i,j) = 4$ $= 3$

$w(0,2) = w(0,1) + p(2) + q(2)$

$= 8 + 3 + 1$

$= 12$

$w_{02} = 12$
$c_{02} = 19$
$r_{02} = 1$

$$c(0,2) = \min_{0<K\leq 2} \begin{Bmatrix} c(0,0) + c(1,2) \\ c(0,1) + c(2,2) \end{Bmatrix} + w(0,2)$$

$K=1$
$K=2$

$= \min \begin{Bmatrix} 0+7, \\ 8+0 \end{Bmatrix} + 12$

$= \min \{7\} + 12$

$= 19$

$r(0,2) = K$

$= 1$

$w(1,3) = w(1,2) + p(3) + q(3)$

$= 7 + 1 + 1$

$= 9$

$w_{13} = 9$
$c_{13} = 12$
$r_{13} = 2$

$$c(1,3) = \min_{1<K\leq 3} \begin{Bmatrix} c(1,1) + c(2,3) \\ c(1,2) + c(3,3) \end{Bmatrix} + w(1,3)$$

$K=2$
$K=3$

$\min \begin{Bmatrix} 0+3 \\ 7+0 \end{Bmatrix} + 9$

$= 3 + 9$

$= 12$

$r(1,3) = K$

$= 2$

$w_{24} = 5$
$c_{24} = 8$

$w(2,4) = w(2,3) + p(4) + q(4)$

$= 3 + 1 + 1 = 5$

$$c(2,4) = \min_{2<K\leq 4} \begin{Bmatrix} c(2,2) + c(3,4) \\ c(2,3) + c(4,4) \end{Bmatrix} + w(2,4)$$

$K=3$
$K=4$

$= \min \{ 0+3, 3+0 \} + w(2,4)$

$= 3+5$

$= 8$

$r(2,4) = K$

$= 3$

$w(0,3) = w(0,2) + P(3) + q(3)$

$= 12 + 1 + 1$

$= 14$

$w03 = 14$
$c03 = 25$
$r03 = 2$

$c(0,3) = \min_{0 < K \le 3} \{ c(0,0)+c(1,3),\ c(0,1)+c(2,3),\ c(0,2)+c(3,3) \} + w(0,3)$

$K=1$
$K=2$
$K=3$

$= \min \{ 0+12, 8+3, 19+0 \} + 14$

$= 11 + 14$

$= 25$

$r(0,3) = K$

$= 2$

$w14 = 11$
$c14 = 19$
$R14 = 2$

$w(1,4) = w(1,3) + P(4) + q(4)$

$= 9 + 1 + 1$

$= 11$

$c(1,4) = \min_{1 < K \le 4} \{ c(1,1)+c(2,4),\ c(1,2)+c(3,4),\ c(1,3)+c(4,4) \} + w(1,4)$

$K=2$
$K=3$
$K=4$

$= \{ 0+8, 7+3, 12+0 \} + 11$

$= 8 + 11$

$= 19$

$r(1,4) = K$

$= 2$

$w(0,4) = w(0,3) + P(4) + Q(4)$

$= 14 + 1 + 1$

$$c(0,4) = \min_{0<K\le 4} \left\{ \begin{array}{l} c(0,0)+c(1,4) \\ c(0,1)+c(2,4) \\ c(0,2)+c(3,4), \\ c(0,3)+c(4,4) \end{array} \right\} + w(0,4)$$

$K=1$
$K=2$ ✓
$K=3$
$K=4$

$= \min\{0+19, 8+8, 19+3, 25+0\} + 16$

$= 16 + 16$

$= 32$

$r(0,4) = K$

$= 2$

constructing Tree for Ton

To4

$r(0,n) = K$

$r(0,K-1)$ $r(K,n)$

$r(i,j) = K$

$r(i,K-1)$ $r(K,j)$

$r(0,4) = K=2$

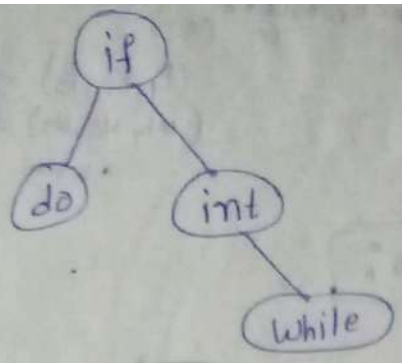
$r(0,1)$ $K=1$ $r(2,4)$ $K=3$

$r(0,0)$ $r(1,1)$ $r(2,2)$ $r(3,4)$ $K=4$

0 0 0

$r(3,3)$ $r(4,4)$

0 0



0/1 Knapsack problem using Dynamic programming

* let us assume 'n' objects are there.
* Each having a profit such as P_i
  " " weight such as w_i

Knapsack 'm'

* our major Aim is we have to place all the objects in the corresponding Knapsack with the maximum profit.
* we can represent a solution for this problem with the help of Knapsack instace = $(x_1, x_2, x_3 \ldots)$

Ex: $n = 3$; $m = 6$, $(P_1, P_2, P_3) = (1, 2, 5)$
$(w_1, w_2, w_3) = (2, 3, 4)$

we have to solve a problem taking series of actions.

We using Dynamic programming approach

$$S^{i+1} = S^i \cup S_1^i$$

$$S^0 = \{$$

$$S^1 = S^0 \cup S_1^0$$

$$S^2 = S^1 \cup S_1^1$$

$$S^3 = S^2 \cup S_1^2$$

$S^0 = \{(0,0)\}$ → initially no object considered so (0,0)

$S^0_1 = \{(0+1), (0+2)\}$

$S^0_1 = \{(1,2)\}$

$(P_1, P_2, P_3) = (1, 2, 5)$

$(w_1, w_2, w_3) = (2, 3, 4)$

1, 2 added

$S^1 = S^0_0 \cup S^0_1$

$S^1 = \{(0,0), (1,2)\}$

↳ check dominance rule

* After calculating a set we have to use a formula called dominance rule (or) purging Rule

(we have 2 pairs)

(P_1, w_1) (P_2, w_2)

$(P_1 \leq P_2)$ & $(w_1 > w_2)$ → suppose $w_1 < w_2$ (less than) there is no problem.

* first object profit is less than 2nd object profit in that occassion.
* The first object weight is less than 2nd object weight.
- we can remove the first pair
* it means P_1 and P_2 should be in ascending order as well as w_1 and w_2 should be in ascending order
* it may not like that
* we can remove the dominant pair (P_1, w_1) from the set
* Here w_1 is greater than w_2. w_1

$S^1 = \{(0,0)\ (1,2)\}$ → weight 0,2 ascending order

profit 0,1 ascending

* So there is no need of any rule

$S_1^1 = \{(0+2, 0+3)\ (1+2, 2+3)\}$

$S_1^1 = \{(2,3), (3,5)\}$

→ check dominance rule

profit 2,3 ascending

weigh 3,5 ascending no problem

$S^2 = S^1 \cup S_1^1$

✓ $S^2 = \{(0,0), (1,2), (2,3), (3,5)\}$

↳ check dom ascending → so we are no need to remove items

~~$S^3 = \{(0,0)\ (1,2)\ (2,3), (3,5)\}$~~

$S^3 = S^2 \cup S_1^2$

$S_1^2 = \{(0,5), (0+4), (1+5), (2+4), (2+5), (3+4), (3+5), (5+4)\}$

$= \{(5,4), (6,6), (7,7)\ (8,9)\}$

Profits & weights are in ascending order no problem

weight 7 ✗

weight 9 max cant ✗

Knapsack m = 6

so maximum profit is 6 only

* suppose if an element beyond the 6 we have to remove that dom

$S^3 = \{(0,0), (1,2), (2,3), (3,5), (5,4), (6,6)\}$

✗ 5,4 Not Ascending

$S^3 = S^2 \cup S_1^2$

✓ $S^3 = \{(0,0), (1,2), (2,3)\ (5,4), (6,6)\}$

* Next step is we have to check. wheather what is the size of Knapsack here 6

* so we have to select a pair which ~~having~~ contains the weight as 6 → (6,6)

* (6,6) weight 6 so we have to consider this pair

* suppose if the pair (6,6) is not available Then what is the value i.e near to 6 (5,4) near considering it.

$(6,6) \in S^3$

* we have to check wheather $(6,6)$ is present in S^3, S^2 or S^1

Step 1. $(6,6) \in S^3$ → $(6,6)$ present in S^3 only. so thats why we have assumed we have to considered 3rd object

$(6,6) \notin S^2$

* Here knapsack soln represented with the help of $x_1\ x_2\ x_3$ (S^3 mean last object x_3)

$x_3 = 1$

* it means we need to place the 3rd object in the knapsack.

* whenever the 3rd object is considered. Then what we have to do is → subtract the 3rd object ($\overset{(5,4)}{\rightarrow}$) from the corresponding pair

$(6-5, 6-4)$

$= (1,2)$.

* Next step is check $(1,2)$ is present in S^2, S^0 or not

Step 2 $(1,2) \in S^2$ T

$(1,2) \in S^1$ T

* $(1,2)$ are present in also S^1. so we no need take 2nd object. no need to place 2nd object in knapsack

$x_2 = 0$

* no placing the object. then no need of substraction.

Step 3. $(1,2) \in S^1$ * $(1,2)$ in present in S^1 or not.

$(1,2) \notin S^0$ * $(1,2)$ not presint in S^0

$x_1 = 1$ * so we have to place 1st object in knapsack

x_1 x_2 x_3

(1, 0, 1)

* This is the solution for this Knapsack problem.
* we have considered the 1st object don't consider 2nd object, consider 3rd obj
* so what the maximum profit that we will get

x x_2 x_3

(1 0 1)

1st obj profit 1
1st obj weight 2

3rd obj profit = 5
3rd obj weight = 4

1+5 = 6

weight = 2+4 = 6

So, Maximum profit = 6
So, Maximum weight = 6

Reliability Design

* Reliability Design is one of the application of dynamic programming.
* dynamic programming is mainly used for solving optimization problem
* The optimal prob may require Maximum result or minimum result.
* So optimazation problem may require maximum result so the problem is called as maximization.
* The optimaization may require minimum result Then the problem is called as minimization.
* So here Reliability design is maximization problem. because we have to improve the Reliability of entire system. ~~is called as a~~ This Reliability design is a maximization problem.

What is

Reliability ?

Reliability means the probability of working good of a device

* suppose a device there it is working with good probability
* so the probability of working condition of device is called as Reliability.
* Now we want to design a system that consists of maximum probability with the consideration of cost.

→ This is the main objective of the Reliability design.

→ Reliability design mainly used for maximize the Reliability of the entire system by the considering the cost of the system.

* This is main objective of Reliability design

* so here there are several number of devices are there suppose n number of devices are there. Thes n number of devices are connected in series. so The cost of the entire system is less and the Reliability of the system also less.
* because. Suppose These n number of ~~connected~~ devices are connected in series. if any one device fails so Then entire system stops working. so Then the Reliability of the entire system is very less
* Next, The cost of the entire system very less. because all the devices are connected in series.

There are n

D1

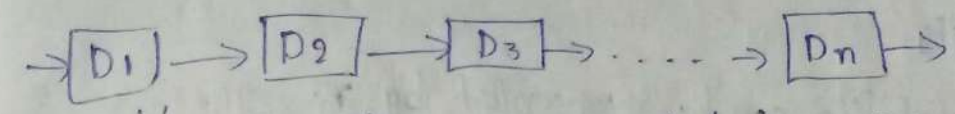
* These n ne
cost of the
in series.
* suppose
also Dn
→ How t
These
are conne
so we
so that de
improve

D1
D1
D1

stage-1

suppose
so Then
and ren
* so The
device
* but
some
* our
with

There are 'n' num of devices are there i.e



* These 'n' no of devices are connected in series Then the cost of the entire system is very less. because 'n' devices are connected in series. if one fails it's stop working.

* suppose device D_2 is fail Then D_3 not working D_4 not working. also D_n not working

→ How to improve the Reliability of entire system.

These devices are connected in parallel. so how these devices are connected in parallel.

so we can take multiple copies of each and every device so that devices are connected in parallel. So Then we have to improve the Reliability of entire system.

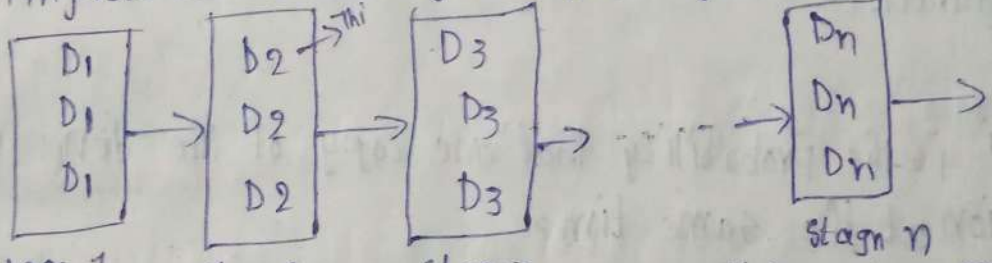


fig: multiple devices connected in Parallel in each stage

suppose ~~D_2 is fail~~ The The copy of device D_2 fails. so Then their work can be taken by the remaining device D_2 and remaining device D_2.

* so The system does not stop working. because each and every device are duplicate supose D_2 duplicated by 2 times

* but compared to the Reliability. The increase of cost is some what not satisfactory (or) neglible

* our main Moto is to improve the Reliability of entire system with the consideration of cost

There are n devices are there each and every devices has a Reliability

* That Reliability can be represented by r_i.

* let r_i the reliability of the device D_i. Then the reliability of entire system is Represented as $\prod r_i$

→

* let m_i be the no. of devices of D_i are connected in Parallel with relability r_i

* m_i is Ntg but How many no. of copies of each device.

* suppose How many copies of device D_3 contain $m_3 = 3$ $m_1 = 3, m_2 = 3$
How many no. of of device D_n contain m_n are equal to 3

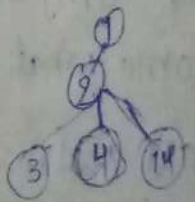
→

* $(1 - r_i)$ is Ntg but the probability that one copy of device will malfunction..

* $(1-r_i)^{m_i}$ is the probability that one copy of the device will malfunction at the same time.

* Hence the reliability of device D_i can be $\phi_i(m_i) = 1-(1-r_i)^{m_i}$ i.e the probability of working good of device

* The main objective of Reliability Design is to maximize the Reliability using device duplication. The maximization is to be carried out value of cost constraint.



let c be the cost of the entire system.

c_i be the cost of i^{th} device.

m_i be the number of copies of device d_i $\phi_i(m_i)$

objective is : maxim Reliability of entire system $\prod_{1 \le i \le n}$

subject to $\sum_{1 \le i \le n} c_i m_i \le c$

where $m_i \ge 1$ and $1 \le i \le n$

How many number of almost copies of each device can be represented as ui it can be calcalated as

$$u_i = \left[\frac{c + c_i - \sum_{j=1}^{n} c_j}{c_i} \right]$$

Dominance Rule

(f_1, x_1) dominates (f_1, x_2) it satisfies 2 conditions

* $(f_1 \ge f_2)$ and $(x_1 \le x_2)$

* Discard all the tuples with spend more cost for less reliability

Example sum

we are to design a Three stage system with devices type D_1, D_2 and D_3. The costs are Rs. 30, 15 and 20 respectively The cost of the system is to more than 105. The reliability of each device to be 0.9, 0.8 and 0.5 respectively

Sol given data

$C = 105$

$c_1 = 30$, $c_2 = 15$, $c_3 = 20$

$r_1 = 0.9$, $r_2 = 0.8$, $r_3 = 0.5$

$$u = \left[\frac{c + c_i - \sum_{j=1}^{n} c_j}{c_i}\right]$$

$$u_i = \left[\frac{c + c_1 - (c_1 + c_2 + c_3)}{c_1}\right] = \left[\frac{105 + 30 - (30 + 15 + 30)}{30}\right]$$

$$= \left[\frac{135 - 65}{30}\right] = \left[\frac{70}{30}\right] = 2$$

$$u_2 = \left[\frac{c + c_i - \sum_{j=1}^{n} c_j}{c_i}\right]$$

$$u_2 = \left[\frac{c + c_2 - (c_1 + c_2 + c_3)}{c_2}\right]$$

$$= \left[\frac{105 + 15 - (30 + 15 + 20}{15}\right]$$

$$= \left[\frac{120 - (65)}{15}\right]$$

$$= \left[\frac{55}{15}\right]$$

$$= 3$$

$$u_3 = \left[\frac{c + c_3 - (c_1 + c_2 + c_3)}{c_3}\right]$$

$$= \left[\frac{105 + 20 - (30 + 15 + 20)}{20}\right]$$

$$= \left[\frac{125 - 65}{20}\right] = \left[\frac{60}{20}\right] = 3$$

we require

2 copies

3 copies

3 copies

we can

Given

let

start

N

N

we require

2 copies of D_1
3 copies of D_2
3 copies of D_3
} -> we can require

we can improve the reliability of 3 stage system.

Given Data $c = 105$

$c_1 = 30, \quad c_2 = 15, \quad c_3 = 20$

$r_1 = 0.9 \quad r_2 = 0.8, \quad r_3 = 0.5$

let us start with $s^0 = (r, x)$

r : reliability

x = cost

start with s^0 initially set of reliability 1, cost is 0

$s^0 = \{1, 0\}$

Now $1 \le m_i \le u_i$ i.e, $1 \le m_1 < u_1$

$1 < m_1 < 2$

$\therefore m_1 = 1$ or 2

Now s_1' calculated as $i = 1$ (consider power value), $j = 1$ (consider suffix value), $m = 1$

$\phi(m_1) = 1 - (1 - r_1)^{m_1} = 1 - (1 - 0.9)^1$

$= 1 - (0.1)^1$

$= 1 - 0.1$

$= 0.9$

$s_1' = \{0.9 * 1, 0 + 30\} = \{(0.9, 30)\}$

$s'_2 = s$ calculated as $i = 1, \quad j = 2, \quad m_2 = 2$

$\phi(m) = 1 - (1 - r_1)^{m_1} = 1 - (1 - 0.9)^2$

$= 1 - (0.1)^2$

$= 1 - 0.01 = 0.99$

$S_2' = \{0.99 * 1, 2 * 30\}$

$= \{0.99, 60\}$

S' can ban obtained by merging S_1' and S_2'

$\therefore S_0' = \{(0.9, 30)\ (0.99, 60)\}$

S^2 can be obtained from S_1^2, S_2^2, S_3^2

$1 \le m_i \le u_i$ i.e $1 \le m_2 \le u_2$

$= 1 \le m_2 \le 3$

$\therefore m_2 = 1, 2$ and 3

S_1^2 calculated as $i = 2$, $j = 1$, $m_2 = 1$

$d_2(m_2) = 1 - (1 - r_2)^{m_2} = 1 - (1 - 0.8)^1$

$= 1 - (0.2)$

$= 0.8$

$S_1^2 = \{(0.9 \times 0.8, 30 + 15), (0.99 \times 0.8 \cdot 60 + 15)\}$

$= \{(0.72, 45), (0.792, 75)\}$

S_2^2 calculated as $i = 2$, $j = 2$, $m_2 = 2$

reliability is $\phi_2(m_2) = 1 - (1 - r_2)^{m_2}$

$= 1 - (1 - 0.8)^2$

$= 1 - (0.2)^2$

$= 1 - 0.04$

$= 0.96$

$S_2^2 = \{0.9 \times 0.96, 30 + 2 \times 15), (0.99 \times 0.96, 60 + 2 \times 15)\}$

S_3^2 calculated as $i=2$, $j=3$, $m=3$

$$\phi_2(m_2) = 1-(1-r_2)^{m_2}$$
$$= 1-(1-0.8)^3$$
$$= 1-(0.2)^3$$
$$= 1-0.008$$
$$= 0.992$$

$$S_3^2 = \{0.9 \times 0.92, 30+3\times15), (0.99 \times 0.992, 60+3\times15)\}$$

:

Now S^2 can be obtained by merging S_1^2, S_2^2 and S_3^2

$$S^2 = \{(0.72, 45), (0.792, 75), (0.864, 60), (0.9504, 90)$$
$$, (0.8928, 75), (0.98208, 105)\}$$

Traveling sales person Problem using Dynamic programming

* let we have a graph called $G = (V, E)$
* V represents set of vertices
* E represents set of edges
* Travelling sales person problem means we have to start at a starting vertex
* let 1 is starting vertex and we need to visit the remaining vertices exactly onces and then. we have to comeback with the starting vertex with the minimum cost

ex: 1 2 3 4

* so This is called as a travelling sales person problem.
* Here we have to visit all the vertices with shortest distance This is called travelling sales person problem and Here we are implementing dynamic programming

* let the graph contains 4 vertices. The vertices are 1 2 3 4
* Here from the graph we have to construct adjacency matrix
* inorder to construct the adjacency matrix The formula is like this

$$cost[i,j] = \begin{cases} 0 & \text{if } i = j \\ c_{i,j} & \text{if } (i,j) \in G \\ \infty & \text{if } (i,j) \notin G \end{cases}$$

$c_{i,j}$ ← it is an edge → belongs to graph we can assume that cost

∞ → cost is infinity

* Here we have to use a formula. based on this formula only we have to solve the problem.

$$g(i,S) = \min_{j \in S} \{ c_{ij} + g(j, \{S\} - j) \}$$

starting vertex → remaining vertices

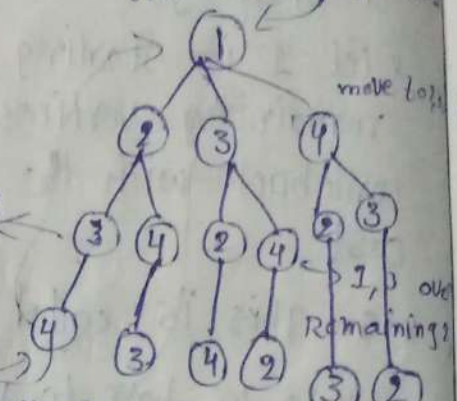
$$g(1, \{2,3,4\}) = \min \{ c_{12} + g(2, \{3,4\}), c_{13} + g(3, \{2,4\}), c_{14} + g(4, \{2,3\}) \}$$

Ex

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	9

→ it is already given on question

assume starting vertex



So all are visited no need to go other vertices simply move to starting vertex

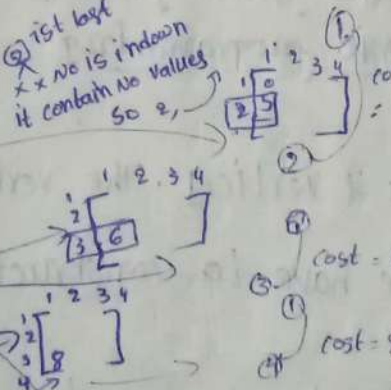
So this is the graph Here. Here we have to solve this problem in 4 stages

First → $|s| = 0$

$g(2, \phi) = 5$

$g(3, \phi) = 6$

$g(4, \phi) = 8$



∴ $|s|$ means set contain how many elements

$|s| = 1$

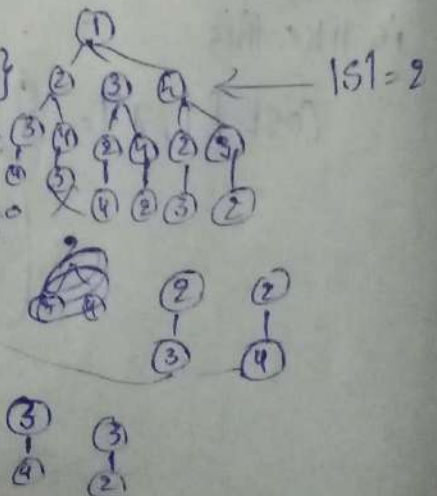
$g(2, \{3\}) = \min \{ c_{23} + g(3, \phi) \} = \min \{ 9 + 6 \}$

$g(2, \{4\}) = 18$

$g(3, \{2\}) = 18$

$g(3, \{4\}) = 20$

g



$g(2, \{4\}) = \{c_{24} + g(4, \{4\} - 4)\}$

$= \min\{10 + g(4, \phi)\}$

$= \min\{10 + 8\} = 18.$

$g(3, \{2\}) = \min\{c_{32} + g(2, \{2\} - 2)\}$

$= \{c_{32} + g(2, \phi)\}$

$= \{13 + 5\}$

$= 18$

$g(3, \{4\}) = \min\{c_{34} + g(4, \{4\} - 4)\}$

$= \{c_{34} + g(4, \phi)\}$

$= 12 + 8$

$= 20$

$g(4, \{2\}) = \min\{c_{42} + g(2, \{2\} - 2)\}$

$\{c_{42} + g(2, \phi)\}$

$= \{8 + 5\}$

$= 13$

$g(4, \{3\}) = \min\{c_{43} + g(3, \{3\} - 3)\}$

$= \{c_{43} + g(3, \phi)\}$

$= 9 + 6$

$= 15$

with this Mod $|s| = 1$ is over. so we have to assume $|s| = 2$

$g(2, \{3,4\})$

$g(3, \{2,4\})$

$g(4, \{2,3\})$

$g(2, \{3,4\}) = \min \begin{cases} c_{23} + g(3, \{3,4\} - 3) \\ c_{24} + g(4, \{3,4\} - 4) \end{cases}$

$= \begin{cases} 9 + g(3, \{4\}) \\ 10 + g(4, \{3\}) \end{cases}$

$= \begin{cases} 9 + 20 \\ 10 + 15 \end{cases}$

$= \begin{cases} 29 \\ 25 \end{cases}$

minvalue = 25

$g(3, \{2,4\}) = \min \begin{cases} c_{32} + g(2, \{2,4\} - 2) \\ c_{34} + g(4, \{2,4\} - 4) \end{cases}$

$= \begin{cases} c_{32} + g(2, \{4\}) \\ c_{34} + g(4, \{2\}) \end{cases}$

$= \begin{cases} 13 + 18 \\ 12 + 13 \end{cases}$

$= \begin{cases} 31 \\ 25 \end{cases}$

3 to 4 is min

min = 25

$g(4, \{2,3\}) = \min \begin{cases} c_{42} + g(2, \{3\} - 2) \\ c_{43} + g(3, \{2,3\} - 3) \end{cases}$

$= \begin{cases} c_{42} + g(2, \{3\}) \\ c_{43} + g(3, \{2\}) \end{cases}$

$= \begin{cases} 8 + 15 \\ 9 + 18 \end{cases}$

$= \begin{cases} 23 \\ 27 \end{cases}$

min = 23

|s| = 3

g(1, {2,

Now le

first ve

1 -

This

program

$= \begin{Bmatrix} 23 \\ 27 \end{Bmatrix}$

min : 23

$|S| = 3$

$g(1, \{2,3,4\}) = \min \begin{Bmatrix} c_{12} + g(2, \{2,3,4\}-2) \\ (c_{13} + g(3, \{2,3,4\}-3) \\ (c_{14} + g(4, \{2,3,4\}-4) \end{Bmatrix}$

$= \begin{Bmatrix} c_{12} + g(2, \{3,4\}) \\ c_{13} + g(3, \{2,4\}) \\ c_{14} + g(4, \{2,3\}) \end{Bmatrix}$

$= \begin{Bmatrix} 10+25 \\ 15+25 \\ 20+23 \end{Bmatrix}$

$= \begin{Bmatrix} 35 \\ 40 \\ 43 \end{Bmatrix}$

$= 35$

Now let us calculate the path, so we have to start from first vertex

$1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$

This is about travelling salesman problem using dynamic programming.

4.
5. Back-tracking

N-Queens problem using Backtracking

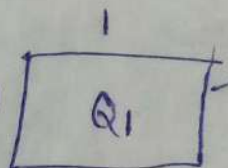
* let us assume n*n chess board is given
* our Main Aim is we have to place n queens on the corresponding chess board in such a way that no 2 queens under attack

* → Three conditions are there
  1) Two Queen's can't on same row
  2) Tow Queen's can't be on same column
  3) Tow Queen's can't be on same diagonal.

* by following these 3 conditions. we have to place the corresponding n queens on the chess board.

* let us solve the problem for N=1, Nxt N=2, N=3, N=4
  Nxt solve 8/8 queen problem

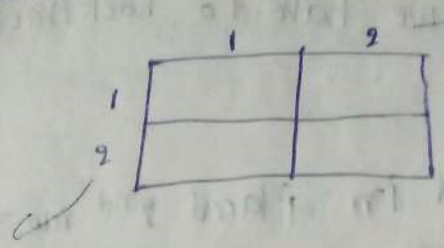
→ first let us solve about 1 Queen problem. 1x1 chess board means we have only one row and only one column.



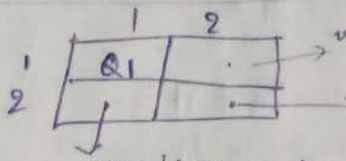
→ Here we can place N Q's
→ The size N=1, so directly Place Q1

* This is called Trivial solution.

→ let us consider ... means we can have 2 rows 2 columns



* let us see we can solve the problem are not.
* 2x2 chess board means we have to places 2 Q's on chess board.
* first is denoted by Q_1 second is denoted by Q_2



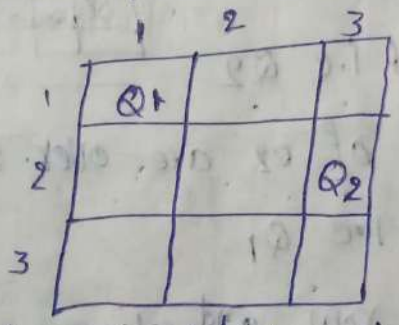
we can't Place, because 1st condition same row

we can't Place, because 3rd condition it is diagonal

we can't Place because 2nd condition same column.

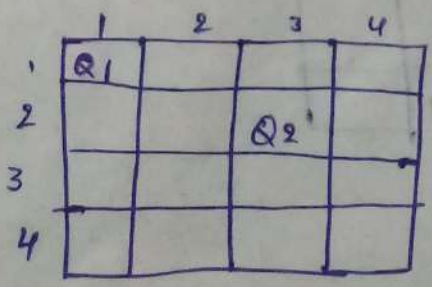
→ * let us take 3*3 (3by3) chess board. so, 3by3 chess board means we can have 3 rows and 3 columns.

* we have to place 3 Queens. because size of chess board is 3.

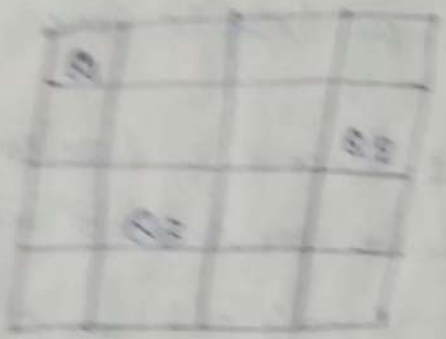


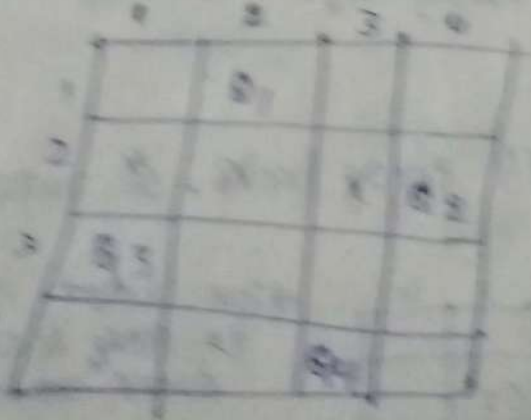
* So, there is no solution. using 3by3 chess board.

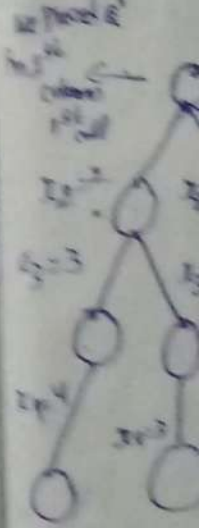
→ let us solve the 4*4 chess board. so number of rows 4 columns 4



* our target is we have to place 4 Queens. no 2 Queens under the attack

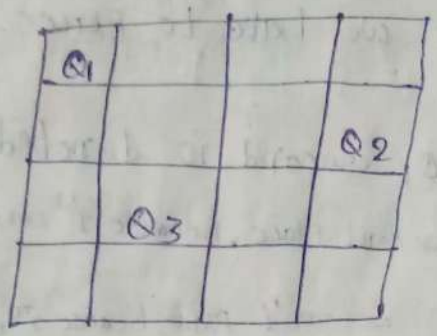




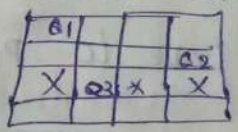


4.
* So, there is no proper position to ~~@~~ inorder to Place Q3. ~~So back~~ in that occasion. we have to backtrack to Previous field

* back tract to Q2. Q2 is Placed in Placed 2nd row 3rd column
* we hale another Possibility is there i.e is 2nd row 4th column

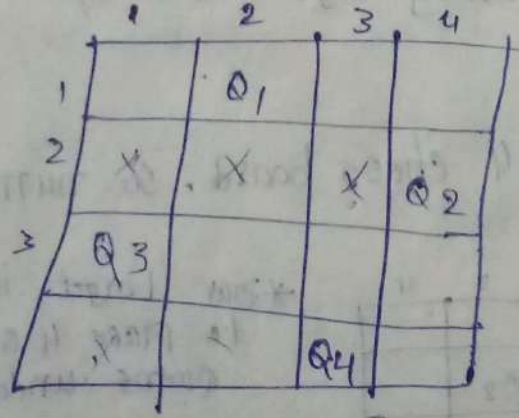


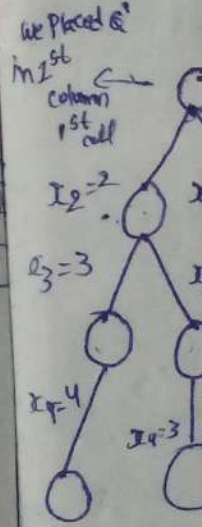
* so, there is no position for placing the Q4 in Chess board
* so we have to back track to previous field.
* The previous field are Q3. but Q3 are not change it position. because it is under attack



* so ~~@~~ backtrack to ~~Next~~ Previous field i.e Q2
* Here all the Possibilities of Q2 are over. so again backtrack to previous field i.e Q1

* Changing the Q2 to First row, 2nd coloumn





* solution is represented with the solution vector

Solution vector = (x_1, x_2, x_3, x_4) → 1st column, 2nd col, 3rd col, 4th col

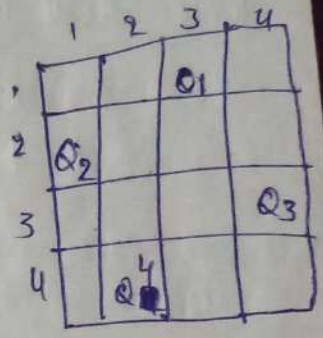
Nlg solu: $= (2, 4, 1, 3)$

$= (\quad)$

- Q_1 is Placed 2nd column
- so
- * 2nd Queen Placed in 4th column

* we have another solution is also there. is Nlg but Mirror image $= (3, 1, 4, 2)$

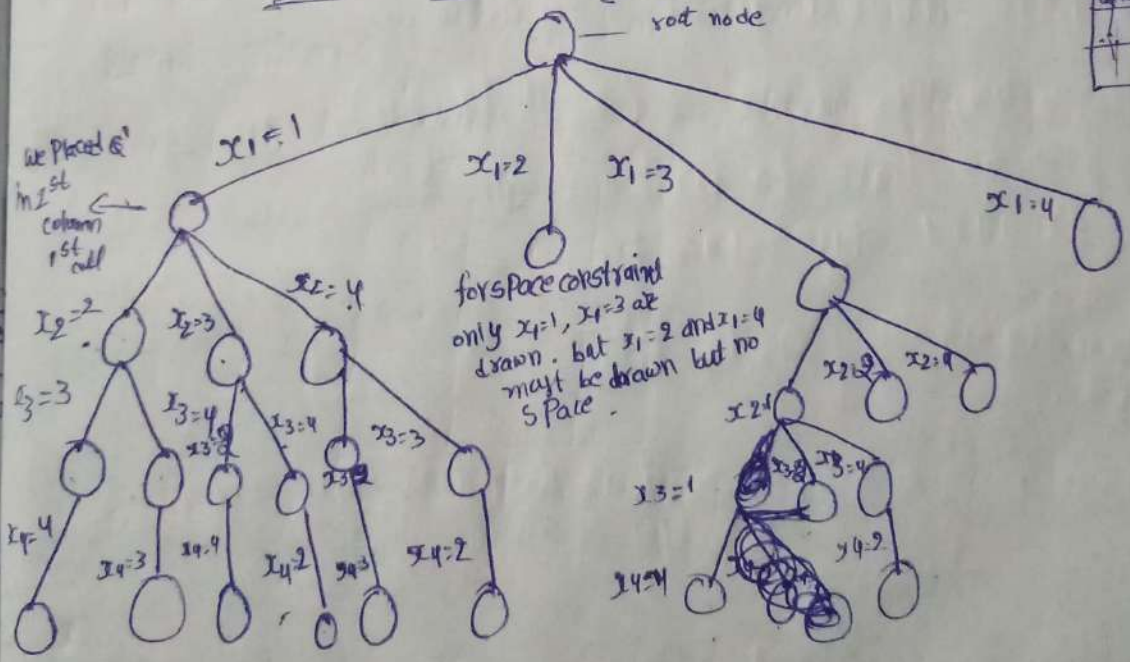
1st Queen Placed in 3rd column

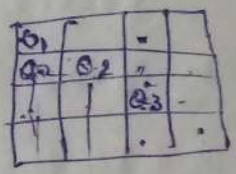


These two are solutions.

* Now, let us draw statespace tree
* State space tree means it contain all the possibles (or) moves solution.

State space tree





sum of subsets problem using Backtracking

* Here 'n' objects are there assume n = 7
* let the weight of objects are denoted by $(w_1, w_2, w_3, \cdots w_7)$
* The value of m = 10 $= (1, 2, 3, 4, 5, 6, 7)$
* The Major Aim of this problem is. The name itself specifies we have to find the subsets combination in which there is sum must be equal to m.

$(x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7)$ → 1+2+3+4

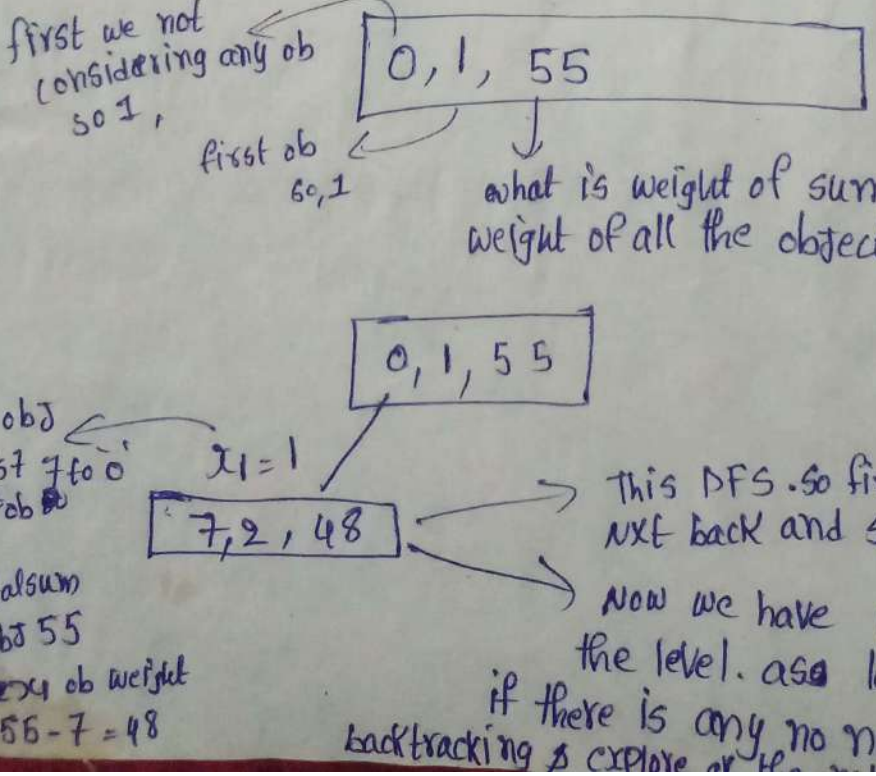
1 1 1 1 0 0 0
1 1 0 0 1 1 0
1 0 1 0 1 0 1
0 1 0 1 0 1 1

* in this way no. of soln we find out all the soln's.

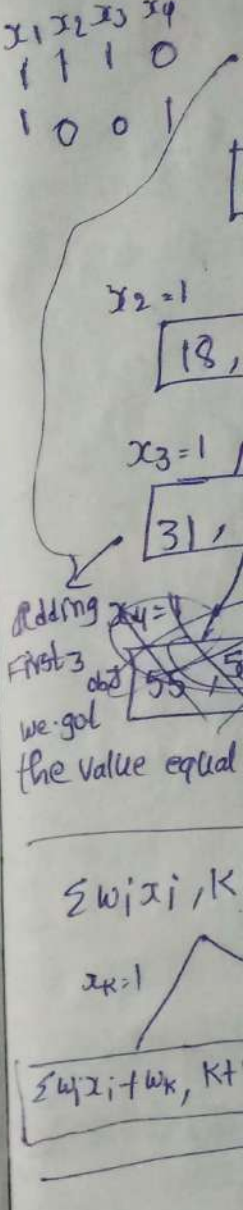
* we add first 4 weight (1,2,3,4) it sum is equt to m.
* This is 1 set of soln
* x_4 x_6 4+6 is also equal to m
* x_7 x_3 7+3 is also eq to m
* This problem have many possible solutions
* so we take limited no. of example.

n = 4, $(w_1, w_2, w_3, w_4) = (7, 11, 13, 24)$, m = 31

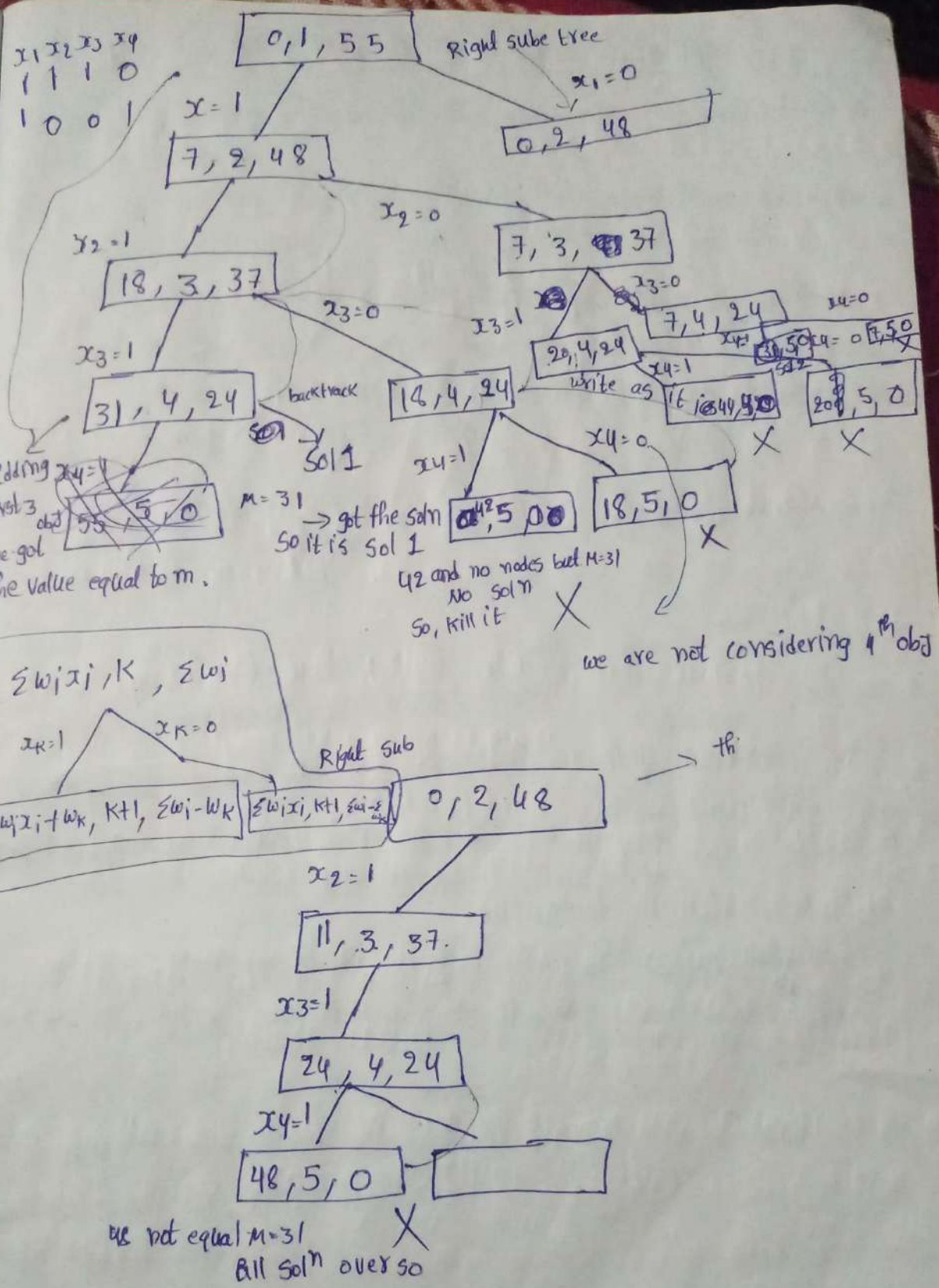
* so we have to find out all the possible solutions
* So we are creating state space tree



Total sum
* obj 55
* x_4 ob weight 55 - 7 = 48

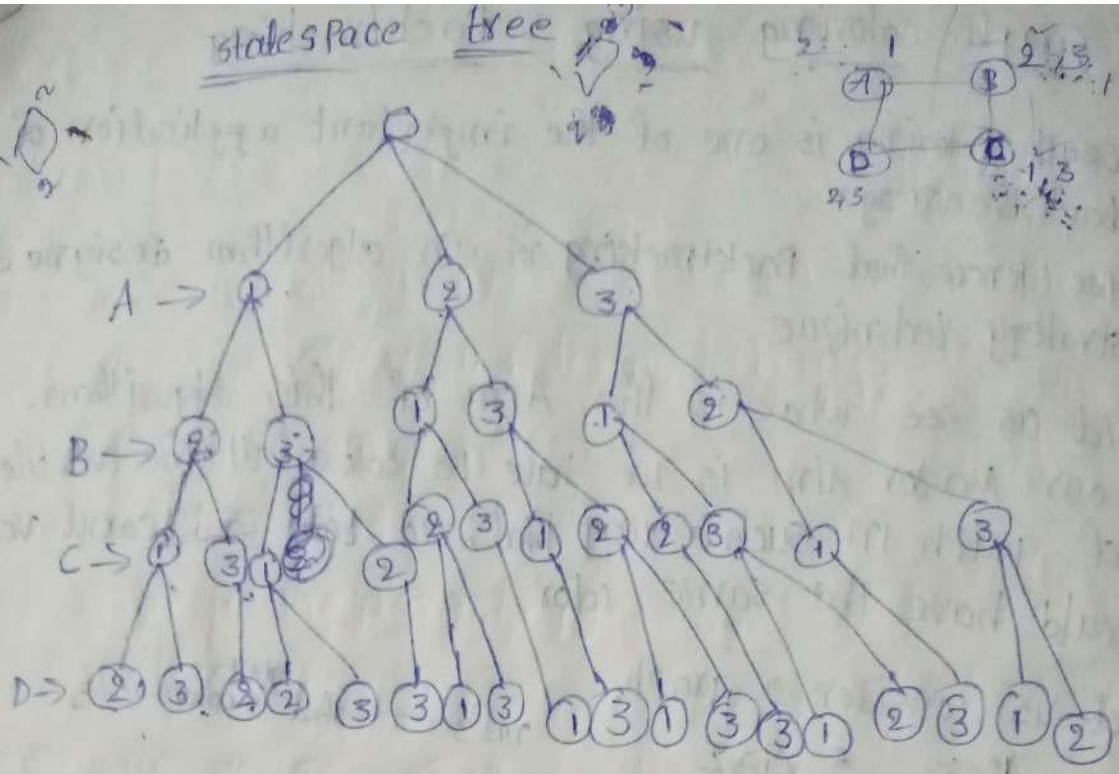


* for this

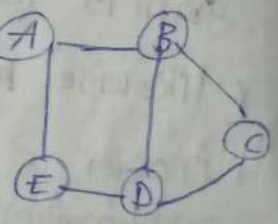


* for this way we calculate the solution.

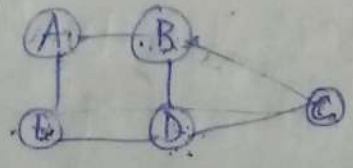
State Space tree



Hamiltonian cycle using Backtracking



* Hamiltonian cycle is one of the important application of backtracking.
* we know the backtracking is an algorithm designed strategic technique.
* The main Aim of this Algorithm is to find all the Hamiltonian cycle in the given graph.
* Hamiltonian means it is a part in which all the vertices are visited exactly onces. and after that we have to visit source vertex. → the starting vertex
* let us take this graph.



* from this graph let us find out what are the Hamiltonian cycle.

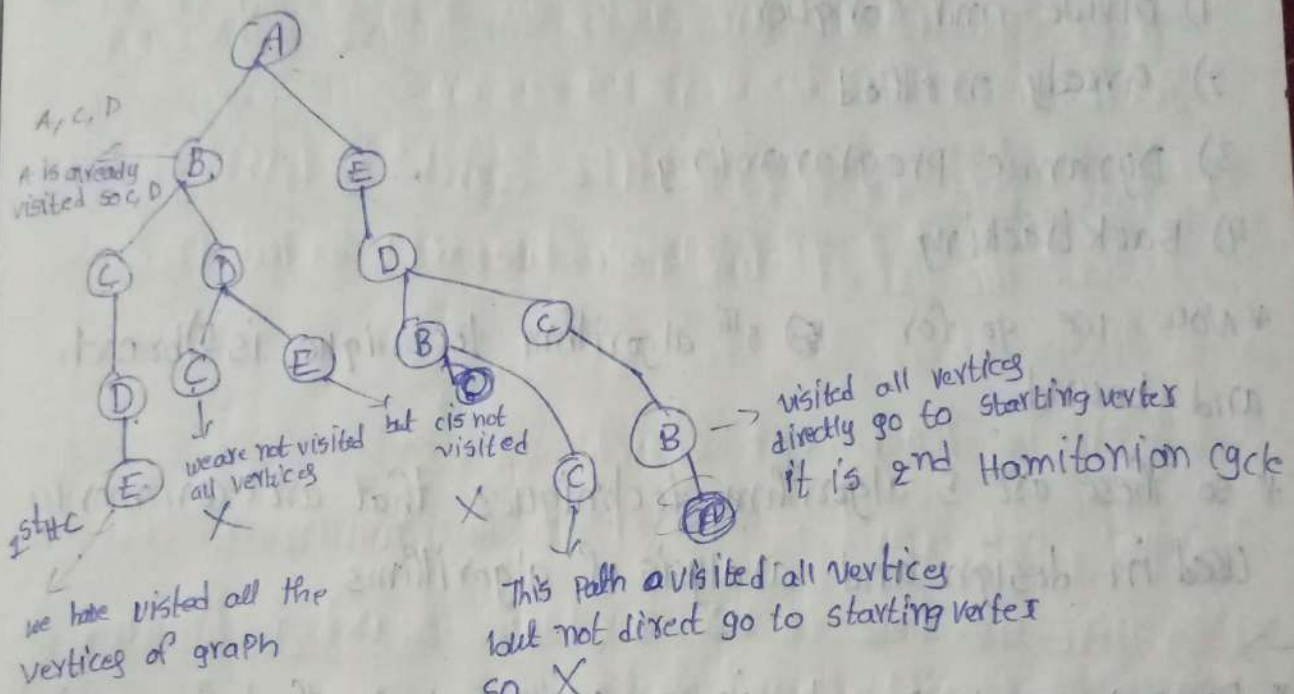
let us assume the first visited vertex that is A next B next C

A B C D E A

A E D C B A

* Now let us construct a state space tree inorder to ~~calculate~~ findout all the hamiltonian cycle.

state space tree



5. Branch and Bound

* Branch and Bound is one of the technique

* Befor going to Branch and Bound we are already discussed 4 algorithms design techniques
  1) Divide and conquer
  2) Greedy method
  3) Dynamic programming
  4) Backtracking

* Now, we go for 5th algorithm technique is 5) branch and bound.

* so these are 5 algorithm techniques that are commonly used in design and analysis of algorithms

→

* Branch and Bound is a systematic method for solving optimization.
  * optimization is Ntg but a problem may require either maximum result or minimum result. it depends on the given problem.

* Here, Branch and Bound method is used, when greed method and Dynamic programming will failed. it means suppose we can take one problem for that problem first we apply greedy method and dynamic programming design technique
* so at the time for that ~~algorithm~~ Problem these two ~~methods~~ doesn't provide any solution

* So, we can say that greedy method and dynamic programming will fail. for finding the optimal soln for the given problem

* so, at that time we have to use branch and bound algorithm design technique to solve that optimization problem.

* Branch and Bound technique is much slower it often lead to exponential time compolexities in the worst case

* it mean Branch and Bound technique is very slower. so because of that reason it can lead to exponential time complexities in the worst case.

* on the other hand, we can applied carefully. it can lead to algorithms that runs reasonable fast on average.

* The general idea of branch bound technique is it is a BFS like search for optimal solution. but all the nodes get expanded

* so this technique can be implemented in the form of a state space tree

* in that state space tree we have to use BFS search technique for finding the optimal solution. in that state space tree all the nodes do not get an expended. but we have to use from selected criteria. ~~that~~ where

* that determines ~~which~~ node to expand, and when the node can get expanded.

* another selection criteria tells the algorithm when the optimal solution has been followed.

* So, these are 2 selected criterial can be used in the branch and bound technique to solve.

* Both Bfs search technique and Dfs search technique are used in Banch and Bound technique

* Bfs is a fifo like search in terms of live nodes. The list of live nodes are represented in a form of Queue

* Here Dfs is a LIfo like search in terms of live nodes. The list of live nodes are represented in the form of stack.

→ 3rd one is least cost search

* least cost search is based on the minimum cost

* So, Here we have to use 3 basic search techniques in Branch and Bound technique.

1) Bfs search technique
2) Dfs search technique
3) least cost search technique.

* like Backtracking in the Branch and Bound we also use bounding functions to avoid sub trees that do not get an answered node.

* if we are getting the answered node we generate the sub trees

* if we are not getting the answered node. now we have to void these sub tree by using branch and bounding function

* Branch and Bound method refers to all the statespace search methods in which all the children of an E-node are generated before any other live node become the E-node

* So, in the statespace tree in the Branch and bound we use 3 types of nodes

1) live nodes
2) E- node
3) dead - node

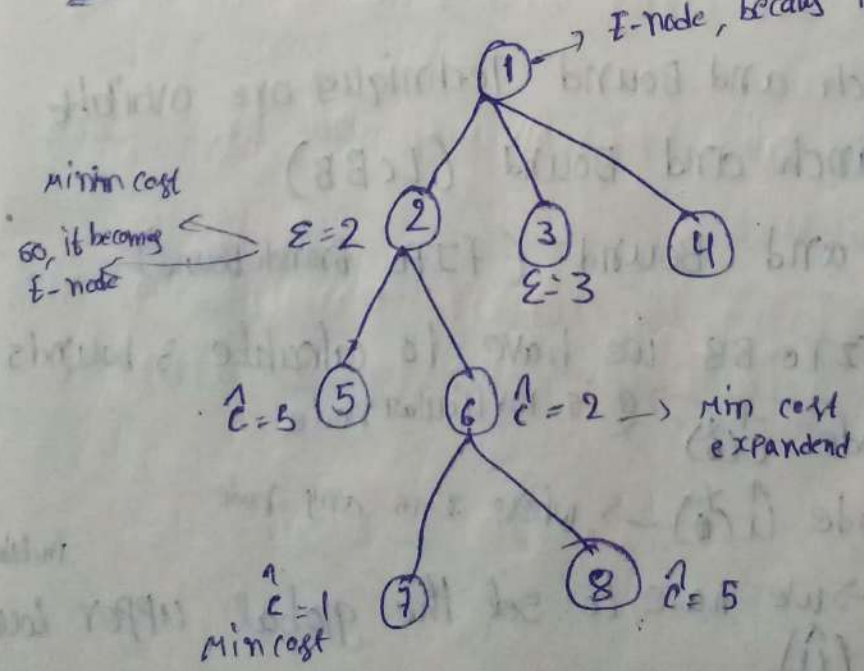
* in the case of live node, a node will gent generated but it's children are not generated - that type of node is called livenode

* E-node : A node will generated and also it's chidrens are generated.

* Dead-node : because the expansion of this node, no use in the future. at that time we will kill the node

Statespace tree Banch and Bound

EX



* node 7 is answered procell, so we teriminate the node

0/1 Knapsack problem using FIFO Branch & Bound

* First let us see what is 0/1 knapsack
* let 'n' objects . each object is having profit $(P_1, P_2 \ldots)$ and weight $(w_1, w_2 \ldots)$
* Knapsack is also given . it Ntg but bag . it is represent by 'm'
* our target is we have to get maximum profit by place all these objects in the knapsack.
* in order to represent the solution vector that is represent by $x_i = (x_1, x_2 \ldots x_n)$
  $= (1, 0, 1, 1)$
  1 mean the x_1 object placed in knapsack
  0 means not present in knapsack.
* our target is we have to findout the knapsack instance (placed) that is Ntg but soln vector as well as what is the profit we are getting
* So, we need to findout these two details.
* Here, Knapsack problem means it deals with maximization maximum profit
* Branch and bound means it deals with minimization problems
* The best example for branch and bound is travelling sales person problem
* it is Ntg but the sales person visit all the cities with minimum distance
* Now, what we have to do is convert maximization into minimization that should be done by Negating the problems
* Profit Ntg but (10, 10 12.) -> we have to Negate (-10, -10, -12)

* Branch b
problem, s
* in order
that prof
* in that
* Here c
bound an
* lower b
* upper

Ex:

n:
(P1, P
(w1,

node

-6
-12
-10
-10

15
=

* The re
* 4th ob
first we

fractions allowed ← C^1

* Branch bound means it deals with minimization problem, so after getting the negative profit.
* in order to get maximization result we have to negate that profit
* in that way we can solve the problem.
* Here each and every object we have to find upper bound and lower bound
* lower bound represented as ($\hat{c}$) Ntg but cost → fractions allowed
* upper bound represented as ($\hat{u}$) → fractions not allowed

Ex:

$n = 4$.

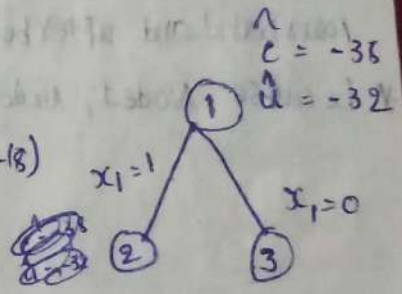
$(P_1, P_2, P_3, P_4) = (10, 10, 12, 18)$

$(w_1, w_2, w_3, w_4) = (2, 4, 6, 9)$

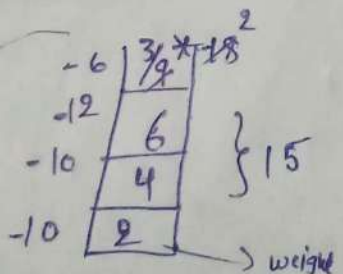
$m = 15$

inorder to calculate we generate state space tree

* negate the profits = $(-10, -10, -12, -18)$



node 1



$15 - (12)$

$= 3$

* The remaining size of Knapsack is 3kg.
* 4th object is 9 we can place 9 on 3Knapsack

first we find lower bound

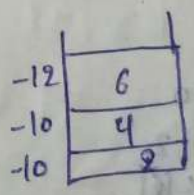
fractions allowed ← $\hat{c} = -10 - 10 - 12 - 6$

$= -38$

$\hat{u} = -10 - 10 - 12$

* After calculation lower & upper bound
* we have to take global upper bound

gub contains root node value $= -32$



→ fractions allowed in upper cost

make to gub

No update

7th Node No update

2nd condition $\hat{u} = -30$

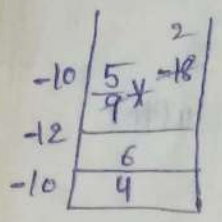
lb > gub -32

$-30 > -32$

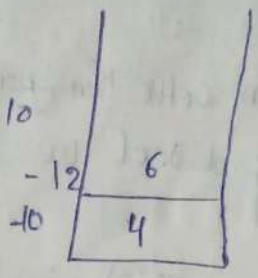
True

Node 3

$x_1 = 0$ → not consider first obj.

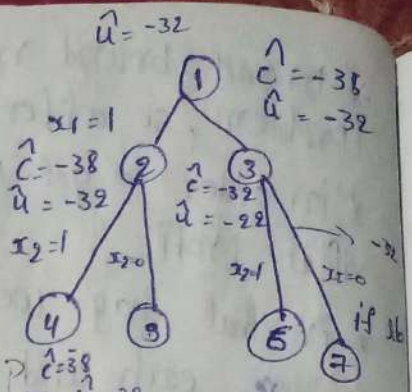


$\hat{c} = -10 - 12 - 10$
$= -32$



$15 - 10 = 5$

we can't place 4th obj 9 on 5kg knapsack



is greater than -32 so no update

-36 then kill the

Node 4

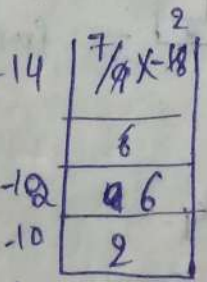
$x_1 = 1, \; x_2 = 1$

* that means we considering first obj as well as 2nd obj.
* Then, lower bound and upper bound will becomes Node2 lower bound and upper bound, Node1 lower & upper bound
* because Node1, Node 2 we have considered both x_1 and x_2 objs

$\hat{c} = -38$
$\hat{u} = -32$

Node 5 :-

$x_1 = 1, \; x_2 = 0$



* it is with min.
* Now, what u minimization Problems
* Profit ntg but (

$\hat{c} = -10 - 12 - 14$
$= -36$

$15 - (6+2)$
$15 - 8$
$= 7$

we can't plac 4th obj 9 on 7kg knapsack

Node 6

x

-10
-12
-10

W

Node 7

$x_1 = 0$

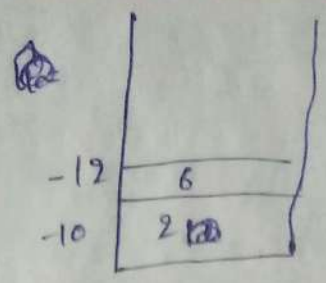
-18
-12

*

= -38
= -32

-32
$x_2=0$
if lb > gub then kill the node

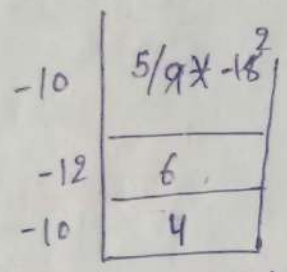
is greater than -22
so no update



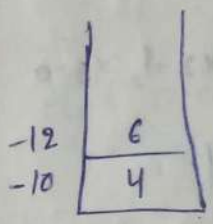
$\hat{u} = -10 - 12$
$= -22$

Node 6

$x_1 = 0, \; x_2 = 1$



$\hat{c} = -10 - 12 - 10$
$= -32$

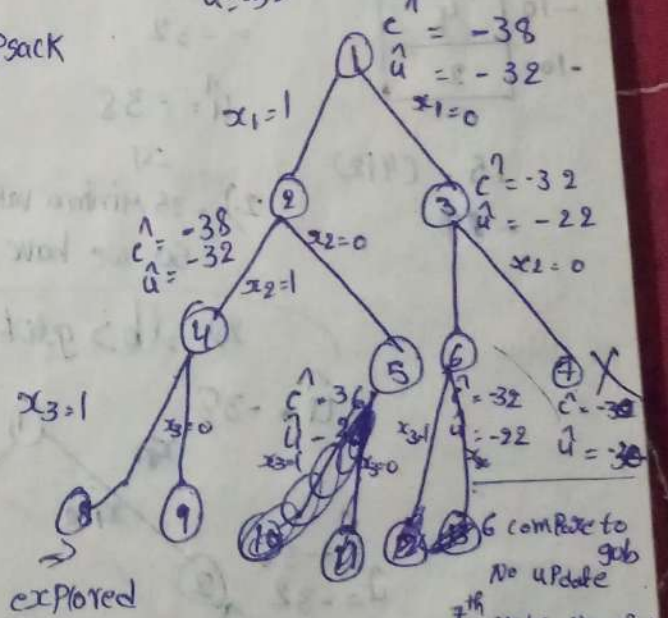


$\hat{u} = -22$

$15 - (6+4)$
$= 5$

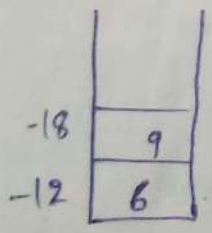
we can't place 9kg on 5kg knapsack

$\hat{u} = -32$



Node 7

$x_1 = 0, \; x_2 = 0$



$\hat{c} = -12 - 18$
$= -30$

$\hat{u} = -30$

2nd condition $\hat{u} = -30$
-32
lb > gub
-30 > -32
True

* There is No need to explore Node 7.

Node 8

$x_1 = 1, x_2 = 1, x_3 = 1$

we are considering 1st, 2nd, obs

* so, the upper bound lower bound vales are previous node values

$\hat{c} = -38$

$\hat{u} = -32$

Node 9

$x_1 = 1, x_2 = 1, x_3 = 0$

-18	9
-10	4
-10	2

$\hat{c} = -10 + -10 - 18$

$= -32$

$\hat{u} = -38$

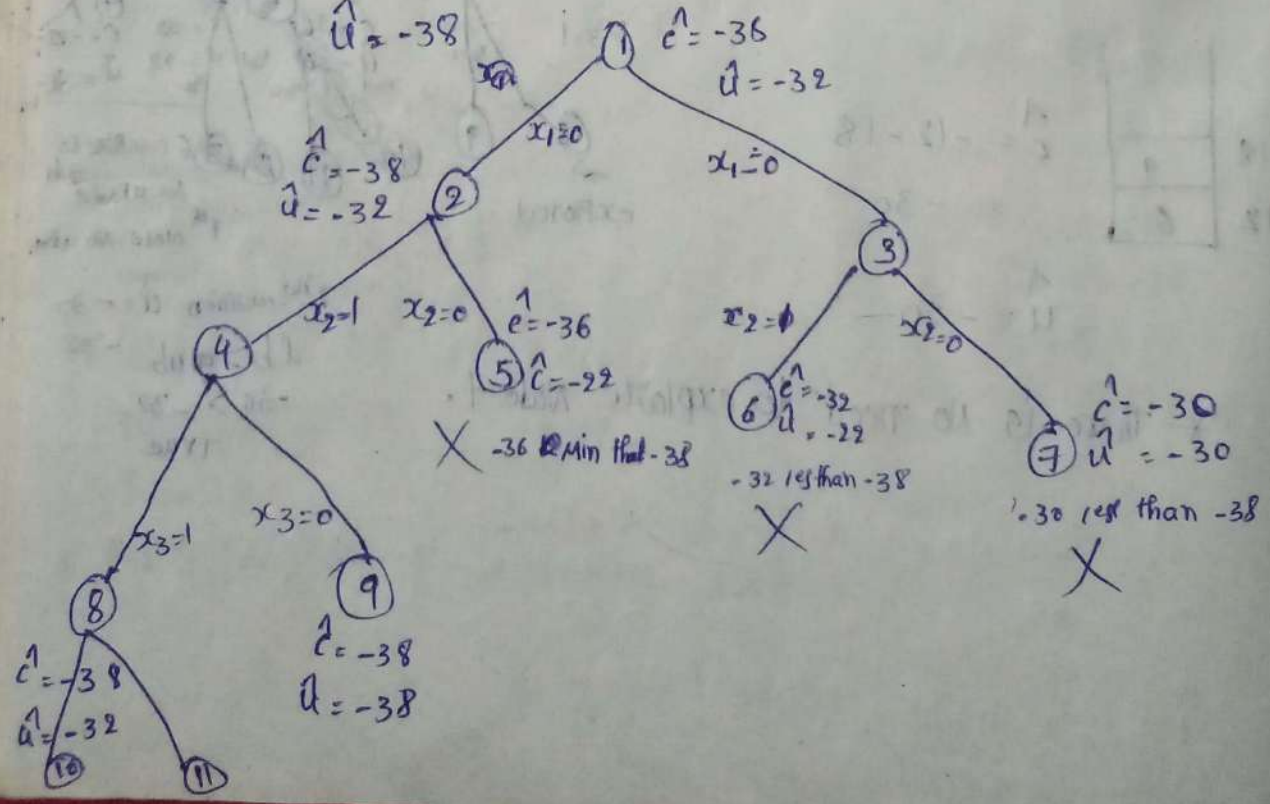
$15 - (4+2)$

$= 9$

1.) -38 minima value than -32

so we have to updata glub as -38

2.) lb > glob



node 10

$x_1=1, x_2=1, x_3=1, x_4=1$

but we have all the 4 objects

* but we can't place 4 object in the knapsack. because all obj size is 21 but our knapsack is only 15.

* This is called infeasible. we didn't get any solution in this path

Node 11

$x_1=1, x_2=1, x_3=1, x_4=0$

* it means we consider first 3 object we don't consider 4th obj

-12	6
-10	4
-10	2

$\hat{c} = -10-10-12$

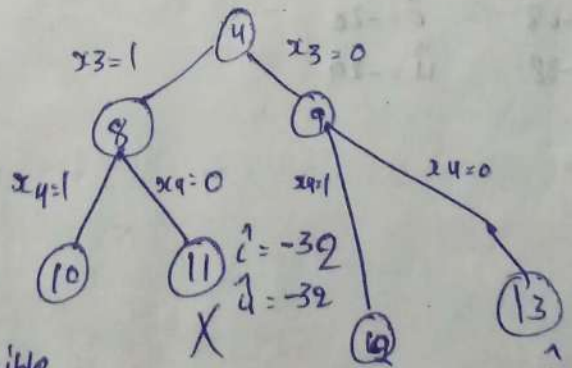
$= -32$

$\hat{u} = -32$

-38 min that -32 No update glob

-32 is > -38

✓ Kill the node



-20 > -38 ✓

Node 12

$x_1=1, x_2=1, x_3=0, x_4=1$

-18	9
-10	4
-10	2

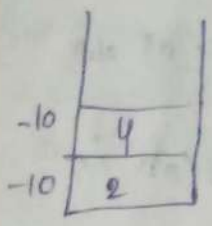
$\hat{c} = -38$

$\hat{u} = -38$

This is the feasible soln because all the paths are killed only one path is there

Node 13

$x_1=1,\ x_2=1,\ x_3=0,\ x_4=0$



$\hat{c} = -20$

$\hat{u} = -20$

13
$\hat{c} = -20$
$\hat{u} = -20$

sub = -38

(1, 1, 0, 1)

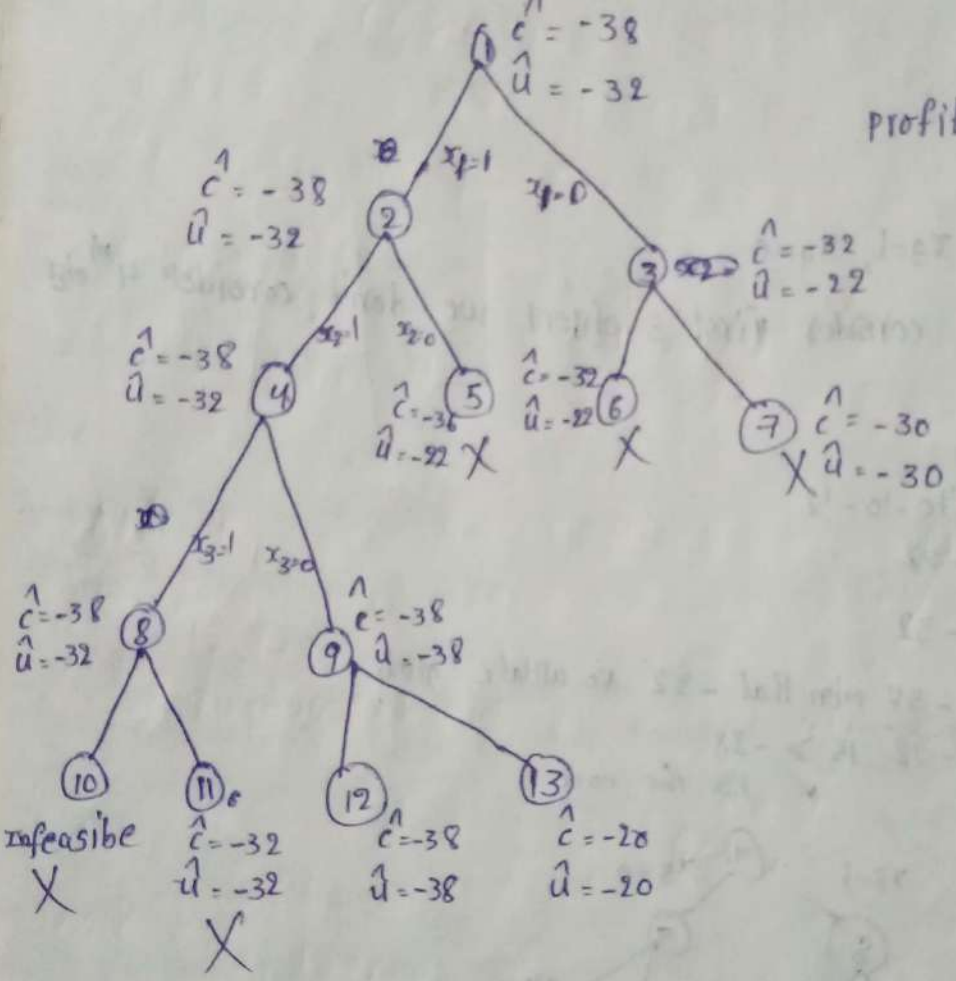
profit = -10, -10, -18

= -38

we convert minimization into maximation, so we have to negative. Now

Max profit = +38

This is knapsack using FIFO



0/1 Knapsack problem using least cost Bound & Bound

* our target is we have to get maximization problems.
* but branch and bound means least minimization problem
* but knapsack is maximization problem
* After getting we have to negate the profit so that we will get the original profit.

$x_i = (x_1, x_2, x_3, x_4)$ → present

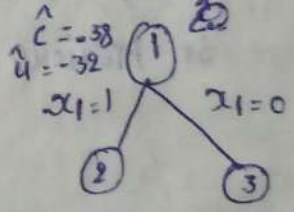
$= (1, 1, 0, 0)$ → can not place on object

* we have to calculate cost & upper bound in every node ($\hat{c}$ → $\hat{u}$)
* cost can also be called lower bound.

$m = 15$, $(P_1, P_2, P_3, P_4) = \{10, 10, 12, 18\}$

$(w_1, w_2, w_3, w_4) = (2, 4, 6, 9)$

* we compict breadth wise search that is level by level.



Node 1.

$x_1 = 1$

-6	3/9 × 18 = 6
-12	6
-10	4
-10	2

$\hat{c} = -6 - 12 - 10 - 10$

$= -38$

$15 - 12 = 3$

-12	6
-10	4
-10	2

$\hat{u} = -10 - 10 - 12$

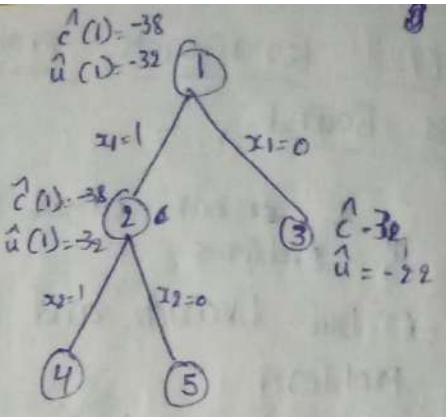
$= -32$

Node 2

$x_1 = 1$ so the previous node values

$\hat{c}(1) = -33$

$\hat{u}(1) = -32$



Node 3

$x_1 = 0$

-10	5
-12	6
-10	4

$\hat{c} = -10 - 12 - 10$

$= -32$

$15 - 10 = 5$

-12	6
-10	4

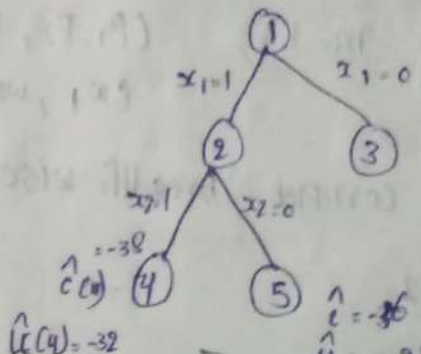
$\hat{u} = -10 - 12$

$= -22$

Node 4

$x_1 = 1$, $x_2 = 1$

So previous node value



Node 5

$x_1 = 1$, $x_2 = 0$

-14	
-12	6
-10	

$\hat{c} = -10 - 12 - 14$

$= -36$

$-15 - 8$

-7

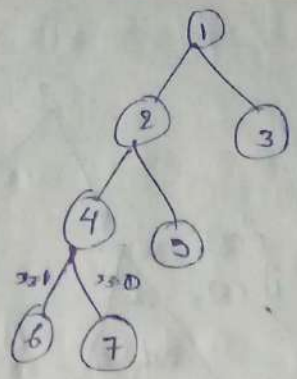
-12	6
-10	8

$\hat{u} = -10 - 12$

$= -22$

-32 is less cost value than

so expand node 4



Node 6

$x_1 = 1, x_2 = 1, x_3 = 1$, so previous node values

$\hat{c} = -38$

$\hat{u} = -32$

Node 7

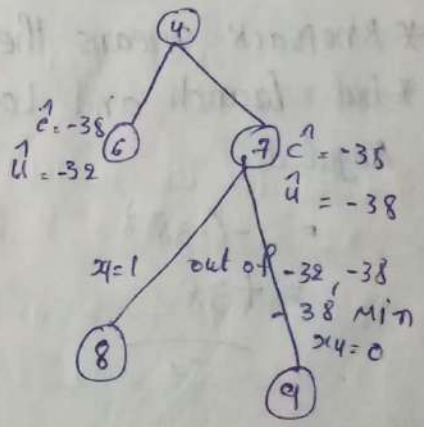
$x_1 = 1, x_2 = 1, x_3 = 0$

-18	9
-10	4
-10	2

15 - 6 = 9

$\hat{c} = -10 - 10 - 18 = -38$

$\hat{u} = -38$



-32 is least cost value that -22 Node 4

Node 8

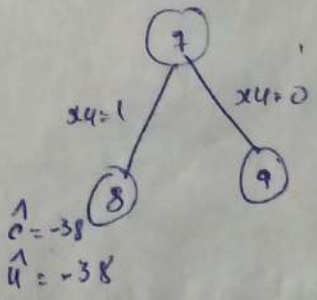
$x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1$

-18	9
-10	4
-10	2

15 - 6 = 9

$\hat{c} = -10 - 10 - 18 = -38$

$\hat{u} = -38$



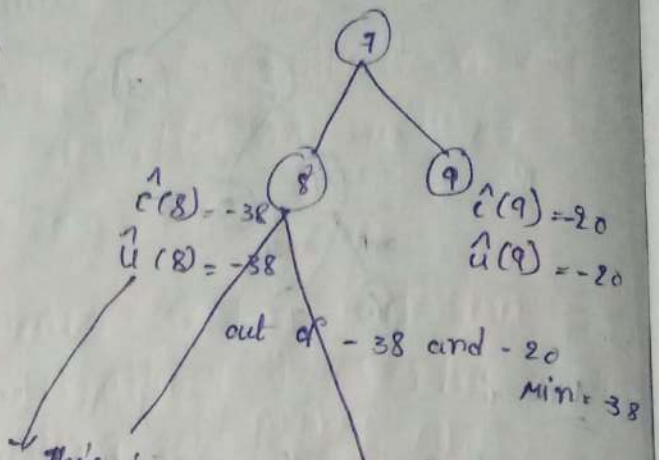
node 9

$x_1=1,\ x_2=1,\ x_3=0,\ x_4=0$

-10	4
-10	2

$\hat{c} = -20$

$\hat{u} = -20$



(x_1, x_2, x_3, x_4)

$(1, 1, 0, 1)$

$= -10 - 10 - 18$

Profit $= -38$

* Knapsack means the maximized
* but branch and bound are minimum. so the profits are Negated

$= -(-38)$

$= +38$

∴

Traveling Salesman Problem cost matrix

let us a graph $G = (V, E)$

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

* our Major aim is we have to visit all the vertices of a graph and after that we need to visit starting vertex of the graph with minimum distance, or with shortest distance
* Here each vertex is ntg but a city
* it is the duty of the salesperson to visit all the cities exactly ones and after that, He has to visit the starting city
* let us assume that 5 cities are given A B C D E. so Here the target is the corresponding salesperson has to visit all the cities in any order C A B D E
* And after that we need to visit starting vertex with the minimum distance with the shortest distance
* so, This is the Major aim of this problem
* Here cost matrix is given, if there is an edge b/w graph is ntg but directed graph.
* There is an edge b/w 2 vertices (A → B) There will be some cost [20 30]
* ∞ → represent there is no edge b/w the corresponding vertices
* A cost matrix is given, No we have to find out the shortest path by using sales person way to visit all the cities, after that we have to visit source city.

* first we find out reduced cost matrix

* inorder to find out the reduced cost matrix. we find row reduction as well as column reduction

row reduction: it means Here each row contain 1 entry. if atleast 1 entry of the corresponding row is '0'

* Then we can say that the corresponding row is reduced

Column reduction: it means, Here we have 5 columns are there. if atleast 1 entry in the each colown is '0'.

* Then we can say that the corresponding colmn is reduced.

* first let us How to find out reduced cost matrix. first we find row reduction.

row reduction mean 1 entry of corresponding row is '0'. for that Purpose what we have to do is

* we have to find out minimum value in each row. suppose in given Matrix is first '0' is Min value.

					Min
∞	20	30	10	11	10
15	∞	16	4	2	2
3	5	∞	2	4	2
19	6	18	∞	3	3
16	4	7	16	∞	4

* Now we have to subtract each entry of the first row with 10

* Next we had to Substract each entry of the 2nd row with 2 likewise

∞	10	20	0	1
13	∞	14	2	0
1	3	∞	0	2
16	3	15	∞	0
12	0	3	12	∞

we observe this, we got 0 that means this row in reduction format

it is also have 0 this also reduction form

* So that means we can say that this matrix in row reduction form

* Now, we to check wheather columns are reduced are not.

* when we can say that column reduction is implemented if 1 entry is 0. we observe the 1st column have no zeroes. so column reduction is not done.

* inorder to find that we have to find out the min value

∞	10	20	0	1
13	∞	14	2	0
1	3	∞	0	2
16	3	15	∞	0
12	0	3	12	∞
Min	1	3		

we have zero so, column is reduced

we have zero so, column is reduced

∞	10	17	0	1
12	∞	11	2	0
0	3	∞	0	2
15	3	12	∞	0
11	0	0	12	∞

→ it is reduced cost Matrix

* if 1 entry in each row as well as column is zero. we can say that it is a Reduced cost matrix.

* Now we find out Reduced cost

~~* Ignore~~

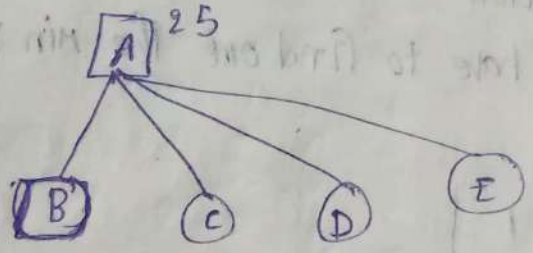
* Inorder to find out Reduced cost we have to sum all these values
* Row Reduction sum + Column Reduction sum.

$10+9+2+3+4 + 1+3$

$21+4$

$= 25$

* Here the 25 is Ntg but Reduced cost.
* Instead of performing on the cost matrix we need to perform on the Reduced cost matrix
* Her we assume that the nodes are A B C D E first one is A



~~Reduced cost matrix~~

(i, j)

1. change all entries of ith row & jth column to infinity
2. (j, 1) → we have change (j, 1) to infinity → This is formula for Branch & Bound.
3. Reduced cost matrix
   ↘ we have to make the matrix as Reduced cost matrix
   ↓
   * We have to perform row reduction and column reduction
   * entry are zero.

cost for Path (1,2)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$$

So, this is in reduced formal only

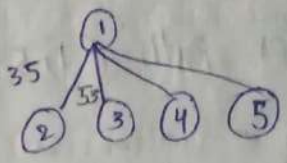
Now, we to find out Path. inorder find out the cost of this node

The formula we have to find out the cost of it from Parent node

we have to findout the edge cost

Cost = 25 + 10 + 0 → becay it is in reduced formal

= 35



* cost for the Path 1,2 is over
* Now let us find out the cost for (1,3)

we find out check it is Reduced cost matrix or not

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{bmatrix}$$

don't take the zero

So Min value 11

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

Reduced cost = 11

Cost = 25 + 17 + 11

= 53

cost for path (1,4)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix}$$

It is already in row reduction format

cost = 25 + 0 + 0

= 25

cost for Path (1,5)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & 2 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 15 & 3 & 12 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

No zero min = 2 → row value to - 2

→ 3 row value to subtract 3

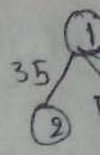
$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

reduced cost = 2+3 = 5

= 25 + 1 + 5

= 31

* our target is we have to visit all the vertices of the graph with shorter distance.



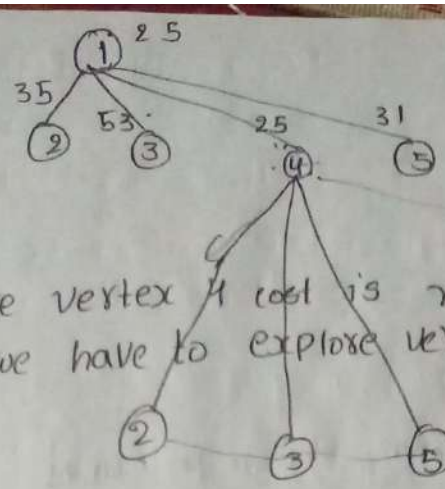
* Here ve
So, we

* The Pa
Now.

* Now

cost

(2,1) →



1 to 4 is min cost

1 and 4 are visited so 2, 3, 5 added

* Here vertex 4 cost is minimum so, we have to explore vertex 4

* The Path from 4 to 2, 4 to 3, 4 to 5. This is our target Now.

* Now to perform operation on this matrix. (1,4) suppose this is in our place.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

cost for path $(4,2)$ (i, j)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix}$$

$(3,1) \rightarrow \infty$

$(4,2)$

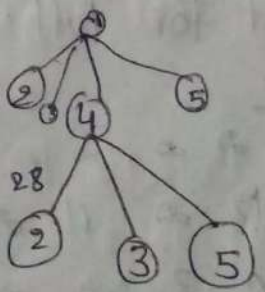
$25 + 3 + 0$

$= 28$

first check Reduction matrix

1st row Reduction any zero

2nd column reduction zero



cost = 2+3 = 5

Cost for Path (4,3)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ \infty & 3 & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & \infty & \infty & \infty \end{bmatrix} \rightarrow 2$$

(3,1)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ 11 & 0 & \infty & \infty & \infty \end{bmatrix}$$

rows over, then go to columns

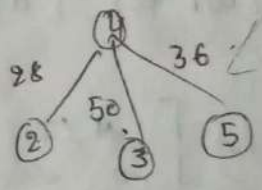
11

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \end{bmatrix}$$

(4,3) = 12

reduced cost = 11+2
= 13

cost = 25 + 12 + 13
= 50



Cost for Path (4,5)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix} \rightarrow 11$$

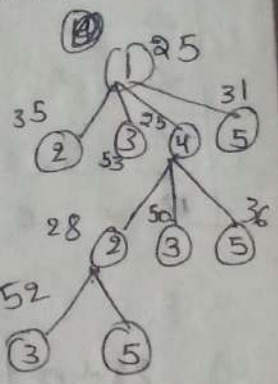
$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

Reduced cost = 11
edge cost (4,5) = 0

= 25+0+11

= 36

out of 2, 3, 5. 4 to 5 Path is minimum so exp/e it.



* So NO we have to determine the Path from 2-3, 2-5.

cost 1 * No we Perform operation (4,2) Matrix → suppose it is question Place

∞	∞	∞	∞	∞
∞	∞	11	∞	∞
0	∞	∞	8	2
∞	∞	∞	∞	∞
11	∞	0	∞	∞

11+2
13

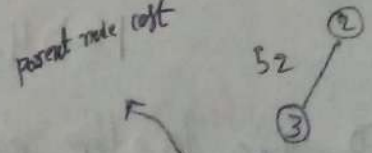
cost for Path i & (2,3)

	∞	∞	∞	∞	∞
(3,1)	∞	∞	∞	∞	∞
(3,1)	0	∞	∞	∞	2 → 2
	∞	∞	∞	∞	∞
	11	∞	∞	∞	∞ 11

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	0
∞	∞	∞	∞	∞
0	∞	∞	∞	∞

Reduced cost = 2+11
= 13

edge cost (2,3) = 11



cost = 28+11+13
= 52

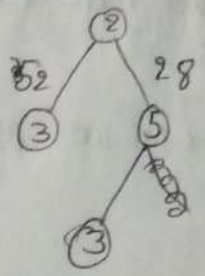
cost for Path (2,5)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix}$$

- Reduced cost - 0
  edge cost (2,5) : 0

-) 28+0+0

= 28



* out 3 and 5 Node, Node is Minimum
* So explore it.

*So we have to Perform operations on (2,5) matrix

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix}$$

Cost for Path (5,3)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Reduced cost = 0
edge cost (5,3) = 0

cost = 28 + 0 + 0

= 28

* So, this is the soln here. travelling sales person in which order to visit the vertices.

$$1 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 1$$

* So, This is the order.
* So, This gives the shortest distance