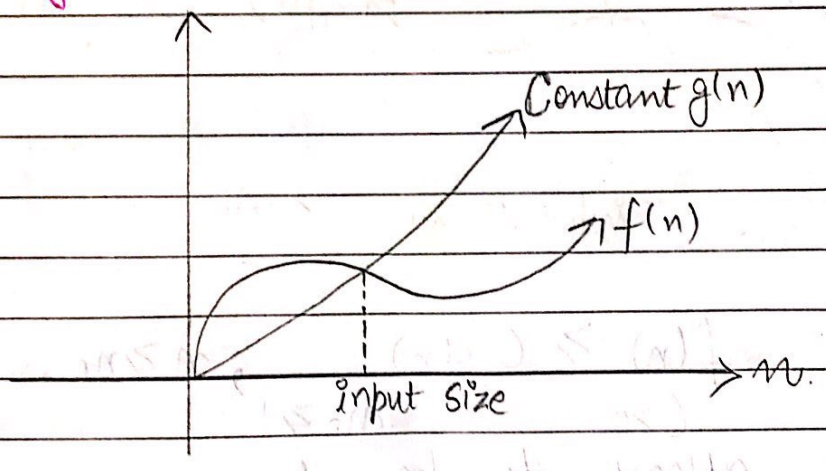
19/7/19

PAGE NO. :
DATE : / /

ANALYSIS OF ALGORITHM

Asymptotic Notation (Related to time complexity)

(i) Big oh O



After some input no: $f(n) \leq g(n)$

$n \geq n_0$

$C > 0$ $\quad n_0 \geq 1$

for eg. $f(n) = 3n+2$ $\quad g(n) = n$

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

$f(n) \leq C\ g(n)$ $\quad C>0\ ,\ n_0 \geq 1$

$3n+2 \leq Cn$

$C = 4$; for $n = 0$.

$n = 1$ $\quad 3\times1 + 2 > 4 \times 1$ $\quad [Cn = 4]$

$n = 2$ $\quad 3\times2 + 2 = 4\times2$

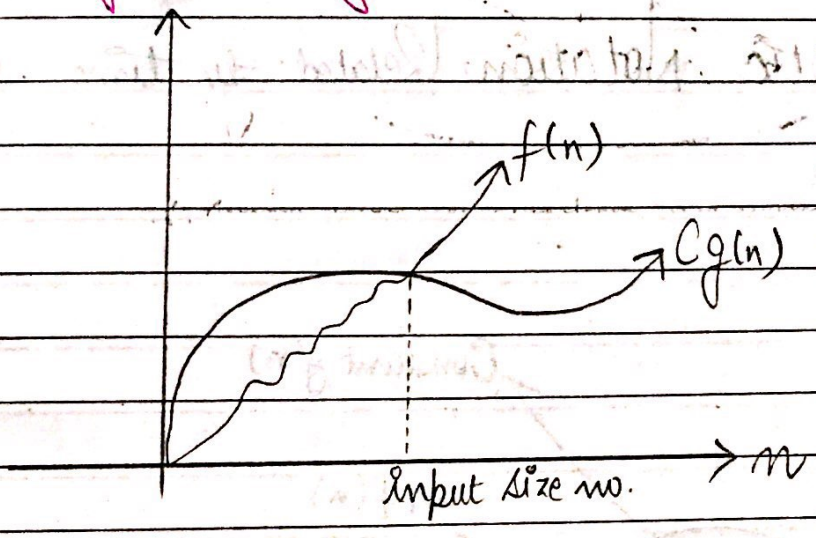
$n = 3$ $\quad 3\times3 + 2 < 4\times3$

$n \geq 2$ $\quad f(n) = O\ C\ g(n)$

PAGE NO. :
DATE : / /

Best case of $g(n)=0$ & worst case $n \geq 2$

(ii) Big omega (Ω)



$f(n) \geq C\,g(n)$, $n \geq n_0$

$C > 0$ $\quad n_0 \geq 1$

I/p is atleast to be 1

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

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

$f(n) \geq C\, g(n)$

$3n+2 \geq n$ $\quad C=1$

$3\times1+2 \geq 1$ $\quad$ True

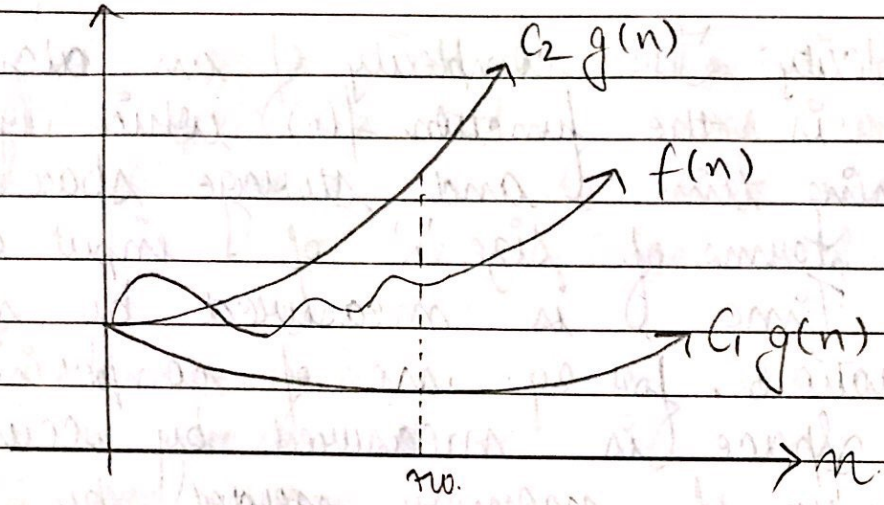
$3n+2 = \Omega(n)$; $f(n) = \Omega(n)$

iii) Big theta Θ (average)

Bound a function $f(n)$ by a function $g(n)$ by upper & lower bound.

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

$C_1\, g(n) \leq f(n) \leq C_2\, g(n)$



C_1 & $C_2 > 0$

$n \geq n_0$ $\quad n_0 \geq 1$

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

$f(n) \leq C\, g(n)$ where $C = 4$

$3n+2 \leq 4(n)$ $\quad n_0 \geq 2$

$f(n) > g(n)$

$3n+2 \geq n$, $n_0 \geq 1$

1) **Best Case:** In any case or in any algo, if we will not achieve better output than previous one then that case is called as best case.

2) **Worst Case:** In any case if time of execution will exceed from previous case or previous input then it is termed as worst case.

3) **Average case:** In any case if a range of input will give optimal result then that case is called as average case.

designed = unit of time

1/7/19

Q: What is algorithmic complexity

PAGE NO. :
DATE : / /

Complexity: The complexity of an algorithm A, is the function f(n) which gives the running time and storage space requirement in terms of size 'n' of input data.
Time is measured by no of operations, for eg. no. of comparisions.
Space is measured by counting the maximum of memory needed by the algo.

Time Complexity: is given by no of steps taken by algorithm to compute the function.
- Comment count as zero step.
- Assignment as one step.
- For iterative statement like for while, do while count as for the control part.

Complexity = f(n) = running time or storage space in terms of size of I/P data.
Time = no. of operations
Space = memory required.

Time complexity : formula = $S(p) = C + Sp(n)$

```
1. {  Sum (arr,n)
2.
       Condition = N
3.     S = 0
4.     for I = 0 to N+1
5.  }  Set S = S + arr[i]
6.     Return
```

PAGE NO. :
DATE : / /

No. of counts for assignment steps = 3.
No. of counts for loop comparision = N
no. of counts for loop increment = N
Total Time = 2N+3.

Space Complexity :
1. Instruction Space
2. Data space
3. Environment stack space
   i) Return address — Formal Parameter
   ii) Value of all local variable & formal para.
4. Recursion stack space
   i) Local & formal parameters.
   ii) Max. no. of nested recursive calls.

→ It is amount of memory required by an algo. needs to run to completion or it can be used to estimate the size of largest problem that a program can solve.

```
{ SUM (Arr, n)
  S = 0.
  for I = 0 to N-1
      Set S = S + Arr[I]
      Returns
  }
```

Space needed by n = 1 word
Arr = N word
I = 1 word

Space needed by S = 1 word.
Total space = N+3 consider n.

- If space c. = n^2+n+1 = consider n^2.

NOTE If there will be 2 loops, & have dependency in them then both complexity will be multiplied for total complexity. & in case of no dependency, they will be sum.

→ If there is no recursiveness in program then its complexity will be O(1).

Time Complexity:

f(n) = approximate time take by an algo

Algo.

Iterative

```
A()
{ for i=1 to n
    max(a,b)
}
```

Recursive

```
A(n)
{ if ()
    A(n/2)
}
```

eg.1 A()
{
int i
for(i=1 to n)
{ Pf("Name")

⇒ O(n)

PAGE NO. :

DATE : / /

2)
```
A()
int i
for( i=1 to; i*i <= n ; i++)        => O(√n)
    pf ("Name")
}
```

3)
```
A()
{
  int i, j
for (i=1 to n)                      => n x n
    for(j=1 to n)                   => O(n^2)
     pf ("Name")
}
```

4)
```
A()
{
int i, j, k, n;
for (i=1 ; i<=n ; i++)
{
  for (j=1 ; j<=i ; j++)
    for (k=1 ; k<=100 ; k++)
    {
      pf ("Name")
    }
}
}
```

i=1	j = 1 time	k = 100
i=2	j = 2 time	k = 2x100
i=3	j = 3 time	k = 3x100

PAGE NO. :
DATE : / /

i = n
j = n times
k = n x 100 times.

Sum all 100 + 2x100 + 3x100 - - - nx100
= 100 (1+2+3 - - - n)
= 100 (n (n+1) / 2)
⇒ O(n²)

5) A( )
{
i = 1, S = 1
while (S ≤ n)
i++
S = S + i
pf ("name")
}

S	1	3	6	10	- -	n
i	1	2	3	4	- - -	k

S = 1 , Sum of 1st natural no.
S = 3 , Sum of first 2 natural no.
S = 6 , Sum of first 3 natural no.
:
S = n , Sum of first k natural no.

k(k+1)/2 , S > n.

$n(2n+1)$ n^2

PAGE NO. : 4

DATE : / /

$$\frac{k(k+1)}{2} > n.$$

$$\frac{1}{2}(k^2+k) > n.$$

$$k^2 > n.$$

$$k > \sqrt{n}.$$

$$k = O(\sqrt{n})$$

6)

```
A()
{
int i, j, k, n
{ for (i=1 , i<=n ; i++)
  { for (j=1 ; j<=i^2 ; j++)
    { for (k=1 ; k<=n/2; k++)
      { pf ("Name");
      }
    }
  }
}
```

$i=1$	$i=2$	$i=3$	$i=4$
$j=1$	$j=4$	$j=9$	$j=16$
$k=n/2$	$j=4n/2$	$k=9n/2$	$k=16n/2$

$$\Rightarrow \frac{n}{2}(1+4+9+16 \cdots n^2)$$

$$\Rightarrow \frac{n}{2}(i^2+2^2+3^2+4^2 \cdots i^2)$$

$$\Rightarrow \frac{n}{2}\left[\frac{n(n+1)(2n+1)}{8}\right] = O(n^4)$$

24/7/19

PAGE NO. :
DATE : / /

Time complexity of iterative codes.

i)
```
A()
{
    for( i=1 ; i<=n ; i=i*2)
        pf("Name")
}
```

$i = 1, 2, 4, \ldots n$
$= 2^0, 2^1, 2^2 \ldots 2^k$
$2^k = n$
$k = \log_2 n$.
Complexity $= O(\log_2 n)$

ii)
```
A()
{
    int i, j, k;
    for(i=n/2; i<=n; i++)          - n/2
    for(j=1; j<=n/2; j++)          - n/2
    {for(k=1; k<=n; k=k*2)         - log2n.
        pf("Name")
}
```
Complexity $= O(n^2 \log_2 n)$

iii) Assume $n \geq 2$.
```
A()
{
    while(n>1)
        n = n/2
}
```

PAGE NO. :
DATE : / /

Case 1 If $n=2$ $n=4$ $n=2^3$

$2>1$ $4>1$ $8>1$

2^1 $n=2$. $n=4$

1 time 2^2 2^3

2 2^2

1. ; 2 time 2^1

1 ; 3 time.

$n = 2^k$.

$k = O(\log_2 n)$

Case 2. If n is not power of 2.

20

10

5

2.

1. $\Rightarrow O(\lfloor \log_2 n \rfloor)$ {floor}

iv)
```
A()
for(i=1 ; i<=n ; i++)
  for(j=1 ; j<=n ; j=j+i)
    pf("Name")
```

$i=1$ $i=2$

$j=1$ to n. $j=1$ to n ; increase of step of 2, $n/2$ t.

n times

$i=3$

$j=1$ to n increase in step of 3, $n/3$ times

PAGE NO. :
DATE : / /

$i = k$
$j = 1$ to n , n/k times.

$i = n$
$j = 1$ to n
n/n Times

$\Rightarrow n\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots - \frac{1}{n}\right)$

$\Rightarrow O(n \log n)$

v)
```
A()
int n = 2^2k.
{ for( i=1; i<=n ; i++)
    j=2
    { while ( j<=n)
        j=j^2
      pf ("Name")
    }
```

$k=1$
$n=4$
$j=2$ $j<4$
$j=4$ $j=4$
$n \times 2$ times

$k=2$
$n=16$
$j=2$ $2<16$
$j=4$ $4<16$
$j=16$ $16=16$.
$n \times 3$ times

$k=3$
$n=2^8$

$n \times 4$ times

$n(k+1)$

$n = 2^{2k}$ $\Rightarrow \log n = 2^k$

$\log(\log n) = k$.

$n(\log(\log n + 1))$

T.Complexity) $O(n \log(\log n))$

25/9/19

* → If a func^n continuously increment / decrement then Back substitution is used.

PAGE NO. :
DATE : / /

Complexity of Recursive Codes.

Recursive paradigm A(n) → size of I/P

```
{ S
    if ( )   // condition
    { ret ( A(n/2) + A(n/2) )
    }
```

To solve A(n) time requires t(n)

If → O(1)

$T(n) = 1 + 2T(n/2)$ Recursive func^n.

There are 3 methods to solve it.

Back substitution.
Recursive tree.
Master's theorem.

i) Back Substitution

```
A(n)
{
    if (n>1)
    { ret (A(n-1))
    }
```

$f(n) \Rightarrow T(n) = 1 + T(n-1)$ — 1

Put $n = n-1$

$T(n-1) = 2 + T(n-2)$ — 2.

Put $n = n-2$

$T(n-2) = 3 + T(n-3)$ — 3.

Substitute 2 in 1.

PAGE NO. :
DATE : / /

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

$T(n) = 2+T(n-2)$ —4.

Substitute 3 in 4;

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

$= 3+T(n-3)$

& : so on.

$T(n) = k + T(n-k)$

It is stopped when $n=1$

$T(n) = 1 + T(n-1)$ $\quad n>1$

$T(1)$ $\quad n=1$

$k + T(n-k)$ will stop when $n-k=1$

$n-k=1$

$k = n-1$

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

$= n-1+1$

$= n$.

Complexity => $O(n)$.

Example 2 :

$T(n) = n + T(n-1)$ —1 ; $n>1$

$= 1$; $n=1$

$T(n-1) = n-1 + T(n-2)$ —2

$T(n-2) = n-2 + T(n-3)$ —3.

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

PAGE NO. :
DATE : / /

$n-(k+1)=1$

$n-k-1=1$

$k=n-2$.

Put $k=n-2$ in last eqn:

$\Rightarrow n+(n-1)+(n-2)+\cdots\cdots (n-(n-2))+T(n-(n-2)+1)$

$\Rightarrow n+(n-1)+(n-2)+\cdots\cdots+2+1$

$\Rightarrow \frac{n(n+1)}{2} = O(n^2) \Rightarrow$ complexity.

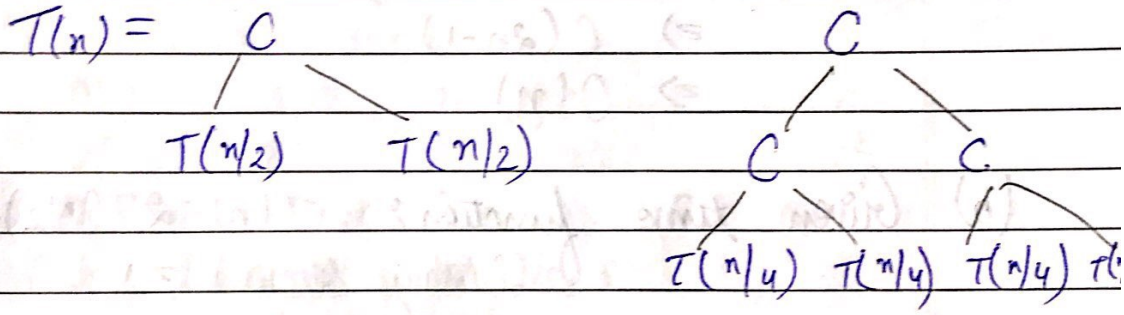
26/7/19

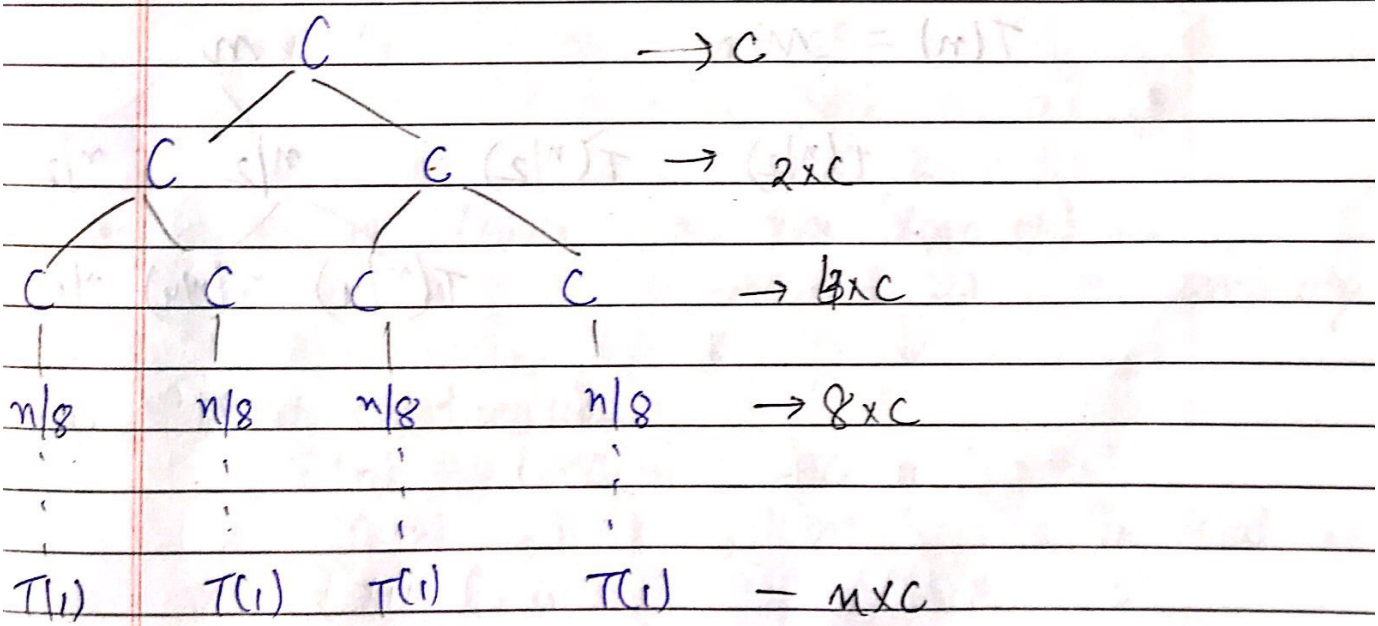
ii) Recursive tree method.

a) Given time function: $T(n) = 2T(n/2)+C \; ; n>1$

(merge sort) $= C \; ; n=1$

Sol^n:





PAGE NO. :
DATE : / /

$T(1) = T(n/n)$ assume $n = 2^k$

$\Rightarrow C + 2C + 4C + 8C \cdots nC.$

$\Rightarrow C(1 + 2 + 4 + 8 \cdots 2^k)$

Geometric progression:

$\Rightarrow C(1 + 2^1 + 2^2 + 2^3 \cdots 2^k)$

$\Rightarrow C\left(\frac{a(r^n - 1)}{(r-1)}\right)$

$\Rightarrow C\left[\frac{1\,(2^{k+1} - 1)}{(2-1)}\right]$

$\Rightarrow C(2^{k+1} - 1)$

$\Rightarrow C(2^k \cdot 2 - 1)$

$\Rightarrow C(2n - 1)$

$\Rightarrow O(n)$

(b). Given time function : $T(n) = 2T(n/2) + n \quad ; n > 1$

(Binary search) $= 1 \quad ; n = 1$

$T(n) = n$

$T(n/2) \quad T(n/2)$

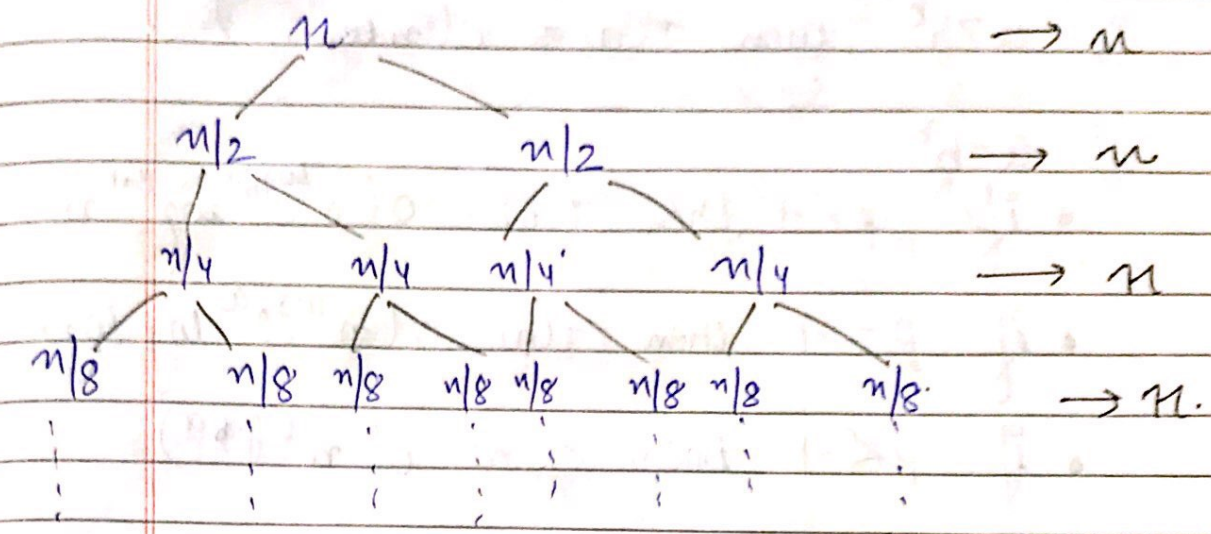
n

$n/2 \quad n/2$

$T(n/4) \quad T(n/4) \quad n/4 \quad n/4$

Recursive can be solved by Masters

PAGE NO
DATE / /



- Series : $n, \frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \ldots \frac{n}{2^k}$.

$\Rightarrow \frac{n}{2^0}, \frac{n}{2^1}, \frac{n}{2^2}, \frac{n}{2^3} \ldots \frac{n}{2^k}$.

- For k levels : $\frac{n}{2^k} = 1$

$n = 2^k$

$\log n = k$

- For k+1 levels $\Rightarrow n(\log n + 1)$

$\Rightarrow O(n \log n)$ $\Rightarrow$ complexity

iii) Masters theorem.

$T(n) = a\,T(n/b) + O(n^k \log^p n)$

$a \geq 1$, $b \geq 1$, $k \geq 0$ & p is real no.

PAGE NO. :
DATE : / /

a) If $a > b^k$ then $T(n) = O(n^{\log_b a})$

b) If $a = b^k$.
- if $p > -1$ then $T(n) = O(n^{\log_b a} \log^{p+1} n)$
- if $p = -1$ then $T(n) = O(n^{\log_b a} \log \log(n))$
- if $p < -1$ then $T(n) = O(n^{\log_b a})$

c) If $a < b^k$
- if $p \geq 0$, then $T(n) = O(n^k \log^p n)$
- if $p < 0$ then $T(n) = O(n^k)$

Eg. • $T(n) = 3T(n/2) + n^2$

$a = 3, b = 2, k = 2, p = 0$

$a \quad b^k$

$3 < 2^2$ true.

Cond^3 applied : $p \geq 0$ $T(n) = O(n^k \log^p n)$

$T(n) = O(n^2 \log^0 n)$

$= O(n^2)$

• $T(n) = 4T(n/2) + n^2$

$a = 4 \quad b = 2 \quad k = 2 \quad p = 0.$

$a \quad b^k.$

$4 = 2^2.$ true

Condn^2 : $p > -1$ $T(n) = O(n^{\log_b a} \log^{p+1} n)$

$T(n) = O(n^{\log_2 4} \log n)$

$T(n) \Rightarrow O(n^2 \log n).$

$a=2 \quad b=2 \quad k=1 \quad p=$

$n^{\log_2 2}$

6) $n \log \log n$
7) $n \log \log \log n$.
8) $O(n^{0.5})$

PAGE NO.
DATE : / /

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

  $a=1 \quad b=2 \quad k=2 \quad p=0$

  $1 < 2^2$

  $T(n) = n^k \log^p n$

  $\Rightarrow n^2 \log^0 n$

  $\Rightarrow O(n^2)$

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

  $a = 2^n \quad b=2 \quad k=2$

  It will solve by using Recursive tree.

- $T(n) = 16\, T(n/4) + n \log n.$

  $a=16 \quad b=4 \quad k=1 \quad p=0$

  $16 > 4^1$

  $\Rightarrow$ ~~$O(n^{\log_4 16})$~~ $O(n^{\log_4 16})$

  $\Rightarrow O(n^4)$. $O(n^2)$

- $T(n) = 2\, T(n/2) + n \log^{-1} n.$

  $\Rightarrow 2\, T(n/2) - n \log n.$

  $a=2 \quad b=2 \quad k=1$

- $T(n) = 2\, T(n/4) + n^{0.51}$

  $\Rightarrow a=2 \quad b=4 \quad k=0.51 \quad p=0$

  $2 \quad 4^{0.51}$

  $2 < 2.027$

  $T(n^{0.51} \log^0 n)$

  $\Rightarrow O(n^{0.51})$.

PAGE NO. :
DATE : / /

- $T(n) = 2T(n/2) + n \log n$.

$a = 2 \quad b = 2 \quad k = 1 \quad p = 0$

$2 = 2$.

$O(n^{\log_b a} \log^{p+1} n)$

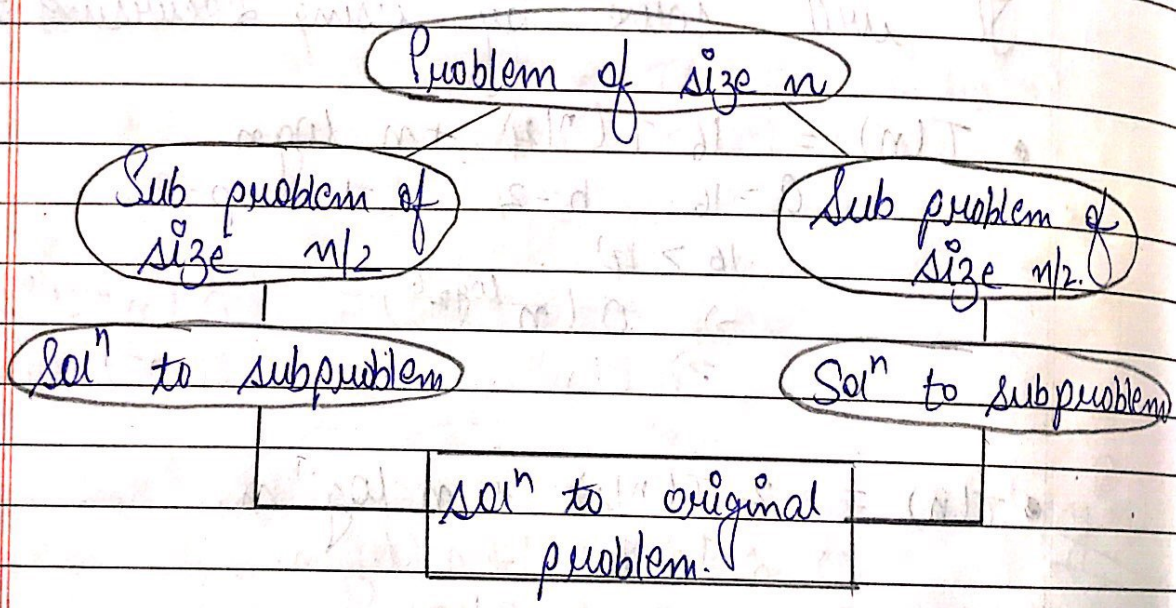
$\Rightarrow O(n^{\log_2 2} \log^{0+1} n)$

$\Rightarrow O(n \log n)$.

30/7/19

UNIT-2

DIVIDE AND CONQUER.



Brief. → It is an algorithm design based on multi branched recursion.

ii) It works recursively breaking down a problem into two or more sub problems of same (or related type) until these become simple enough to be solved directly.

iii) Solution to sub problems are then combined to give a solution to original problem.

iv) Divide & conquer has following 3 steps:

- Divide: Break the problem.

PAGE NO. :
DATE : / /

- Conquer: Recursively solve the subproblems.
- Combine: Combine the sub-solution to form original solution.

Merge Sort:

(Procedure) Merge (A, p, q, r)

$n_1 = q - p + 1$

$n_2 = r - q$

Let $L[1, 2 \cdots n_1+1]$ & $R[1, 2 \cdots n_2+1]$ be new array.

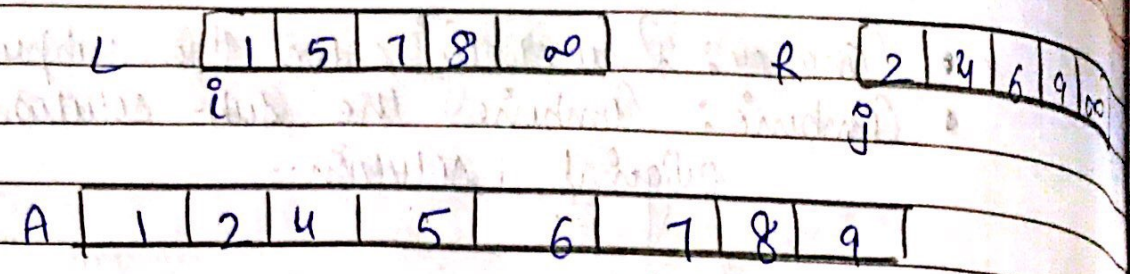
```
for (i = 1 to n1)
    L[i] = A[p + i - 1]
for (j = 1 to n2)
    R[j] = A[q + j]

L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1          j = 1
for (k = p to r)
    if (L[i] ≤ R[j])
        A[k] = L[i]
        i = i + 1
    else
        A[k] = R[j]
        j = j + 1
```

for eg.

A	1	2	7	8	2	4	6	9
	p		q	q+1				r.

PAGE NO. :
DATE : / /

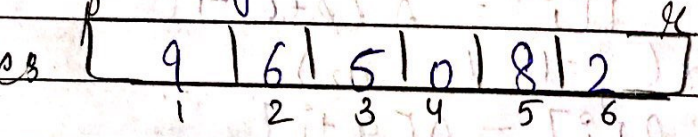


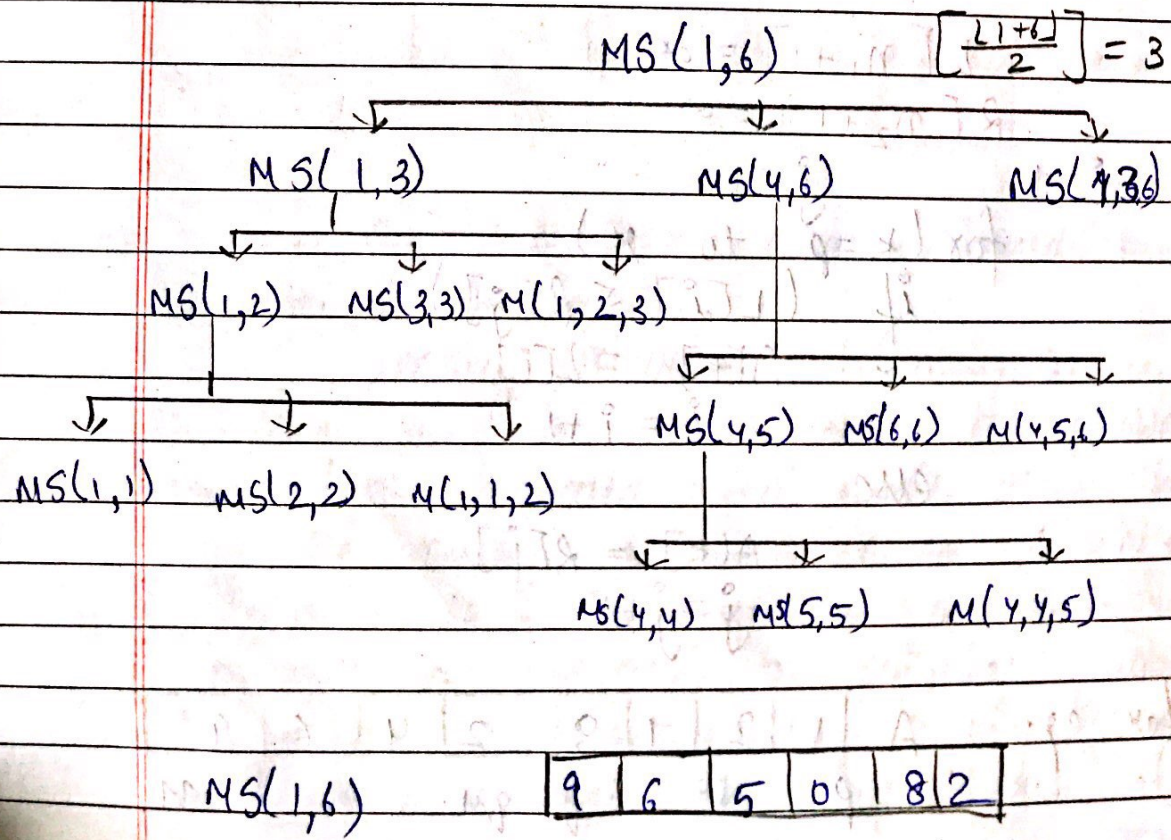
30/7/19

Algorithms

```
Merge sort (A, p, r)
{
    if (p<r)
        q = ⌊(p+r)/2⌋
    merge sort (A, p, q)
    merge sort (A, q+1, r)
    merge (A, p, q, r)
}
```

Examples





PAGE NO. :
DATE : / /

MS (1,3) | 9 | 6 | 5 |
MS (1,2) | 9 | 6 |
MS (1,1) | 9 | ∞ |
MS (2,2) | 6 | ∞ |
M (1,1,2) | 6 | 9 | ∞ |
MS (3,3) | 5 | ∞ |
M (1,2,3) | 5 | 6 | 9 |
M (4,6) | 0 | 8 | 2 |
MS (4,5) | 0 | 8 |
MS (4,4) | 0 |
MS (5,5) | 8 |
M (4,4,5) | 0 | 8 | ∞ |
MS (6,6) | 2 | ∞ |
M (4,5,6) | 0 | 2 | 8 |

M (1,3,6) | 6 | 2 | 5 | 6 | 8 | 9 |

Complexity: $T(n) = 2T(n/2) + n$.

$a = 2 \quad b = 2 \quad k = 1 \quad p = 0$.

(ii) $a = b^k$; $2 = 2$

$p = 0$; then.

(a) $O(n^{\log_b a} \log^{p+1} n)$

$\Rightarrow O(n^{\log_2 2} \log^{0+1} n)$

$\Rightarrow O(n \log n)$

Binary Search.

```
int BinarySearch (int A[], int n, int k)
{
```

PAGE NO. :
DATE : / /

~~int n, int k~~

```
int L=0, mid, R=n-1
while (L<=R)
{
    mid = (L+R)/2
    if (k == A[mid])
        return mid;
    else if (k > A[mid])
        L = mid+1
    else
        R = mid-1;
}
    return -1
}
```

Complexity

Let us consider it for $n=64$ 2^6

$L=0$ $R=64$

$L<R$

mid = $(0+63)/2$ $n=32$ 2^5

$n=32$	$n=16$	2^4
$n=16$	$n=8$	2^3
$n=8$	$n=4$	2^2
$n=4$	$n=2$	2^1
$n=2$	$n=1$	2^0

$n = 2^k$

$k = \log n$.

Complexity = $O(\log n)$.

2/8/19

PAGE NO. :
DATE : / /

Quick Sort

Bidirectional.

First Increment in i, then exchange

Pivot element = Base element, the first element which will get its position.

Partition method:

```
Partition (A, p, r)
{
    x = A[r]
    i = p-1
    for(j=p to j=r-1)
    {
        if (A[j] ≤ x)
        {
            i = i+1
            exchange A[i] with A[j]
        }
    }
    exchange A[i+1] with A[r]
    return i+1
}
```

Example:

Pivot = x.

9	6	5	0	8	2	4	7

i j

j = Scans whole list

9 > 7 no exchange

6 < 7 exchange b/w i+i & j

6	9	5	0	8	2	4	7

i j

5<7

6	5	9	0	8	2	4	7
			j				

0<7

6	5	0	9	8	2	4	7
				j			

6	5	0	9	8	2	4	7
		i			j		

2<7;

6	5	0	2	8	9	4	7
						j	

6	5	0	2	4	9	8	7
				i	i+1		j

(4<7)

j = x.

exchange i++ with x.

6	5	0	2	4	7	8	9
p1						p2	

Time complexity : O(n)

Sorting :

```
Quicksort (A, p, r)
{
if (p<r)
{
q = partition (A, p, r)
Quick sort ( A, p, q-1)
Quick sort ( A, q+1, r)
}
```

PAGE NO :

DATE : / /

Example : A 5 7 6 1 3 2 4

i j → → j r

QS(1,7)

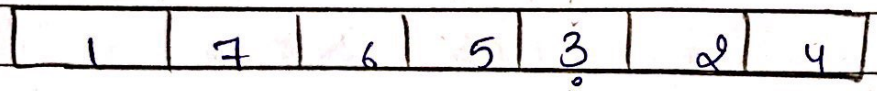
P(1,7) QS(1,3) QS(5,7)

P(1,3) QS(1,1) QS(3,3)

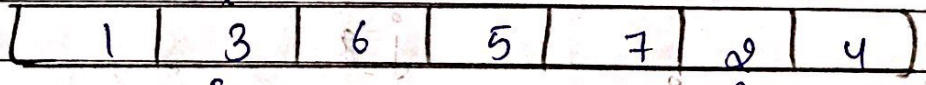
P(5,7) Q(5,4) QS(6,7)

P(6,7) QS(6,6) QS(7,7)

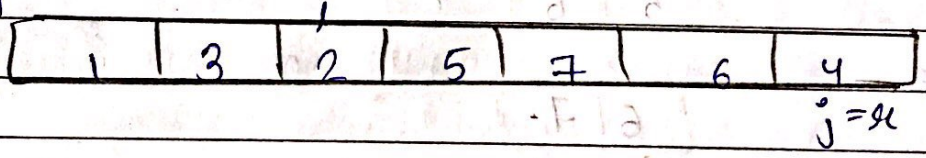
P(1,7)



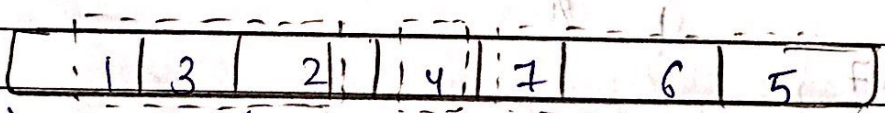
3<4



2<4

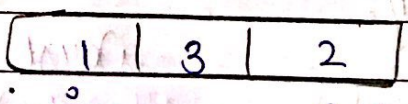


i++



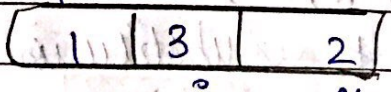
P(1,7) QS(1,3) QS(5,7.

QS(1,3)

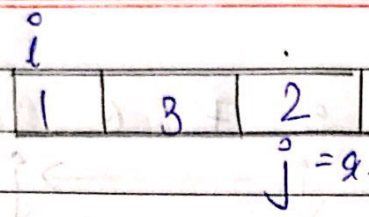


1<2 i++

exchange

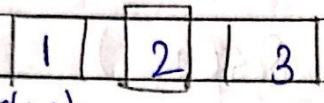


PAGE NO. :
DATE : / /



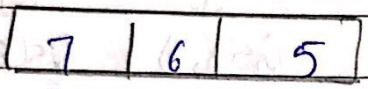
j++

exchange i & j.

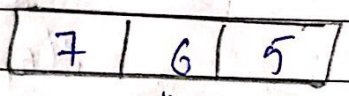


QS(1,1) QS(3,3)

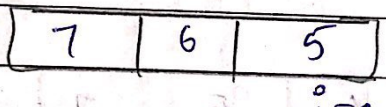
QS(5,7)



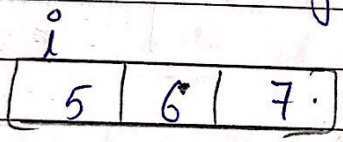
7 > 5



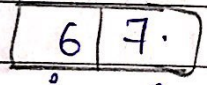
6 > 5



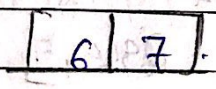
i++



i



6 < 7



Time Complexity:

Best Case

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

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

Worst Case

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

$= O(n^2)$

* Using Back Substitution method.

8/19

PAGE NO :
DATE : / /

Strassen's matrix multiplication. (Theory)

- Simple matrix multiplication

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

$C_{11} = A_{11}B_{11} + A_{12}B_{21}$
$C_{12} = A_{11}B_{12} + A_{12}B_{22}$
$C_{21} = A_{21}B_{11} + A_{22}B_{21}$
$C_{22} = A_{21}B_{12} + A_{22}B_{22}$

8 mul. & 4 addition

Addition $= O(n^2)$

$T(n) = \begin{cases} a & n \leq 2 \\ cn^2 + 8T(n/2) & n > 2 \end{cases}$

$T(n) = O(n^3)$

Description :

- It is a method uses Divide & conquer strategy.
- If A and B are two matrices of $n \times n$. and we assume that n is a power of 2, $n = 2^k$. Then A can be divided into four square matrices of $(n/2, n/2)$ each. This can be done till we get minimum size matrix which can be solved straight away.
- In simple matrix multiplication, requires 8 multiplication & 4 additions.
- Multiplication is more expensive than addition in terms of computation.
- Strassen's method reduces multiplication by seven & same 18 addition & subtractions.

PAGE NO. :
DATE : / /

Strassen's method.

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

(i) Calculation as:

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

example: $A = \begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix}$ $B = \begin{bmatrix} 1 & 2 \\ 4 & 4 \end{bmatrix}$

$n = 2$.

$P = (2+6)(1+4) = 40$
$Q = (5+6)1 = 11$
$R = 2(2-4) = -4$
$S = 6(4-1) = 18$
$T = (2+3)4 = 20$
$U = (5-2)(1+2) = 9$
$V = (3-6)(4+4) = -24$

$C_{11} = 40 + 18 - 20 - 24 = 14$
$C_{12} = -4 + 20 = 16$
$C_{21} = 11 + 18 = 29$
$C_{22} = 40 + (-4) + 11 + 9 = 84$.

PAGE NO.
DATE

Time Complexity

I/P size (n)	No. of multiplication in sequential algo.	No. of mul. in Strassen's
2	8	7
4	64	49
8	512	343
16	4096	2401
:	:	:
?	p^3	$O(n^{2.81})$
n		

$$T(n) = 7\,T(n/2) + 18n^2 \quad ; n>2$$
$$\Rightarrow O(n^{\log_2 7}) = O(n^{2.81})$$

Algorithm.

i) Input two matrices of dimension n.
ii) Multiply two input matrix by using Strassen's matrix. using following steps.

a) Divide matrices into sub-matrices A_0, A_1, A_2 of dimension $n/2$.
b) Using blocked matrix multiply equations.
c) Recursively multiply sub-matrices.
d)

iii) If above algo. is size is not power of 2.
a) Make size near to power of 2.
b) Make the value of all extra rows & columns to zero.

Unit-2.
Greedy Approach

→ Greedy approach is used for optimization problem i.e. from the available alternatives it makes the choice that looks best, optimum at the moment.
It works in two parts -

1.) Optimal substructure -
→ works in stages
→ each i/p is checked
→ if optimal soln added to partial soln.

2.) Greedy choice property -
→ a choice which is best at the moment is chosen which solves the sub-problem.

1.) Knapsack Problem :-
→ Problem statement - It states that a knapsack has capacity of W units. It has to be filled with few items which are 'n' in number, each items has a weight w_i and associated profit value v_i. Knapsack has to be filled till the capacity W is not reached & to earn maximum profit.

* There are 2 variants of knapsack -
(i) 0/1 Knapsack :- Each item put completely or not
(ii) Fractional Knapsack :- Items can be broken & filled into knapsack.

Page No.:
Date:

→ Maximize $\sum_{i=1}^{n} v_i x_i$ subject to $\sum_{i=1}^{n} w_i x_i \leq w$.

$0 \leq x_i \leq 1$ $\quad 1 \leq i \leq n$

Algo :-

v[n] = profit value of n items
w[n] = weight of n items
V[i] w[i] = decreasing order
w = capacity of knapsack
x[n] = soln vector

1.) Repeat step 2 for n items
2.) Initialize vector x
set x[i] = 0 for $1 \leq i \leq n$.
3.) Repeat step 4 for n items
4.) for i = 1 ton
if w[i] < w then
set x[i] = 1
set w = w - w[i]
else
exit.
5.) if all items have not chosen or knapsack is still empty. ie.
if $i \leq n$ then
set x[i] = w / w[i].
6.) exit.

Page No.:
Date:
YOUVA

Ex: $n=7$, $w=15$

$(v_1, v_2, \ldots v_7) = (10, 5, 15, 7, 6, 18, 3)$

$(w_1, w_2, \ldots w_7) = (2, 3, 5, 7, 1, 4, 1)$

$I_1 = v_1/w_1 = 10/2 = 5$

$I_2 = v_2/w_2 = 5/3 = 1.6$

$I_3 = v_3/w_3 = 15/5 = 3$

$I_4 = v_4/w_4 = 7/7 = 1$

$I_5 = v_5/w_5 = 6/1 = 6$

$I_6 = v_6/w_6 = 18/4 = 4.5$

$I_7 = v_7/w_7 = 3/1 = 3$

items	v_i/w_i	w_i	v_i	$x[i]$	w
I_5	6	1	6	1	14
I_1	5	2	10	1	12
I_6	4.5	4	18	1	8
I_3	3	5	15	1	3
I_7	3	1	3	1	2
I_2	1.6	3	5	2/3	0
I_4	1	7	7	0	

1.) $w_5 = 1 < w$

$x[5] = 1 \quad w = w - w[5] = 15 - 1 = 14$

2.) $w_1 = 2 < 14$

$x[1] = 1 \quad w = 14 - 2 = 12$

6.) $w_2 = 3 < 2$

Knapsack can accomodate 2 units.

$w/w[2] = 2/3$

$x[2] = 2/3$

$w = w - w[2] \times \frac{2}{3}$

$= 2 - 3 \times 2/3 = 0$

total profit $= \sum_{i=1}^{n} v_i x_i$

$= 1 \times 6 + 1 \times 10 + 1 \times 18 + 1 \times 15 + 1 \times 3 + \frac{2}{3} \times 5$

$=$

* Job Sequencing Problem :-

Problem statement :- A set of 'n' jobs with some deadline is given if job is completed within its deadline then only profit is added. All jobs are executed on a single machine for 1 unit of time, a job with maximum profit value is selected from the set of jobs 'j' and checked if it can be completed within specified deadline only then sol^n of the sub-problem is added to original problem. It follows both optimal sub-structure & greedy approach.

Algorithm :-

'j' is set of jobs with deadline 'd' there are 'n' jobs

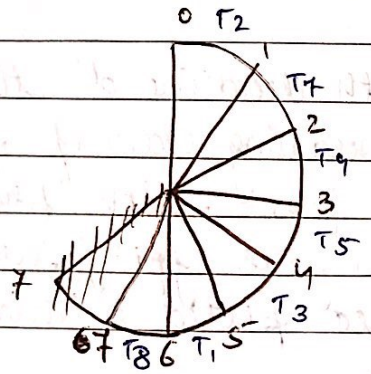
(i) put the jobs in decreasing order of profits.
(ii) Initialize the set of jobs with first job set $j = \{j_1\}$
(iii) Repeat this step for jobs ~~j[i]~~ j_i where $i = 2$ to n.
(iv) If job j_i can be completed within deadline then set $J = J \cup \{J_i\}$
(iv) Exit.

Example:

Tasks	Deadline	Profit
T_1	7	15
T_2	2	20
T_3	5	30
T_4	3	18
T_5	4	18
T_6	5	10
T_7	2	23
T_8	7	16
T_9	3	25

→ arrange in descending order-

Tasks	Deadline	Profit
T_3	5	30
T_9	3	25
T_7	2	23
T_2	2	20
T_4, T_5	3,4	18,18
T_8	7	16
T_1	7	15
T_6	5	10



Page No.:
Date:
YOUVA

Total profit = 30+25+23+20+18+16+15

* Optimal Merge Pattern :-

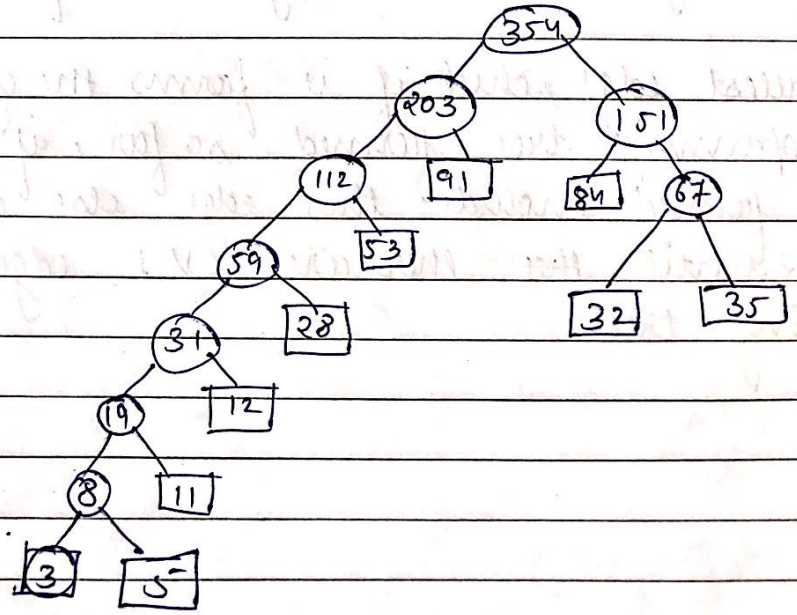
Problem statement :- Let there are 2 files with m & n records to merge. 2 files it will take m+n moves if more than 2 records are required to merge it can be done in pairs. Is there any soln that minimize the feasible moves.

Algorithm :

1) Sort files with increasing value of length of files.
2) Two variables min1 & min2 are initialized with first two files.
3) Set min1 = file1 & min2 = file2.
3) Merge min1 & min2 to give file x.
4) Remove original file1 & file2. Put x in the list go to step1 and repeat till one file remains.
5) Exit.

Example :- For 10 files of length 28, 32, 12, 5, 84, 53, 91, 35, 3, 11 arrange in ~~a~~ increasing order.

3, 5, 11, 12, 28, 32, 35, 53, 84, 91.



Page No.:
Date:

Level = depth + 1

total no. of record moves = $\sum_{i=1}^{n} d_i g_i$

$= (7 \times 3) + (7 \times 5) + (6 \times 11) + (5 \times 12) +$
$(4 \times 28) + (3 \times 53) + (2 \times 91) + (3 \times 32) +$
$(3 \times 35) + (2 \times 84)$

$= 1004$

Total 1004 required to merge 10 files.

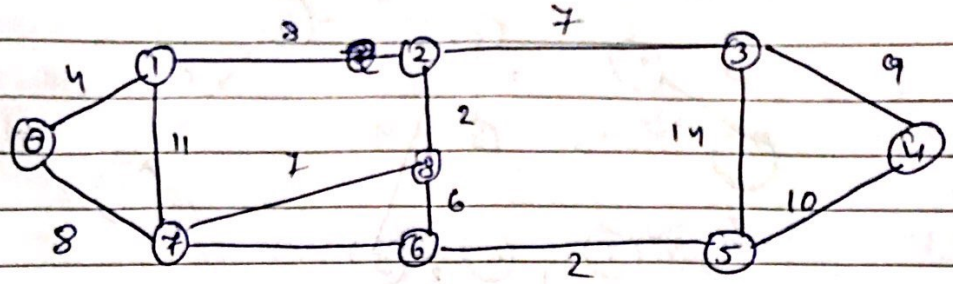
* MST using greedy Algorithm:-

Given a connected an undirected graph, a spanning tree of that graph is a sub-graph i.e. a tree and connects all the vertices together. A minimum spanning tree is a spanning tree with weight less than or equal to the weight of every other spanning tree.

→ A minimum spanning tree has V-1 edges where V is the no. of vertices in the given graph.

Kruskal's algorithm -

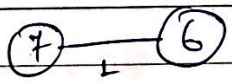
1.) Sort all the edges in increasing order of their weight.
2.) Pick the smallest edge, check if it forms the cycle with the spanning tree formed so far, if cycle is not formed include this edge else discard.
3.) Repeat step 2 until there are V-1 edges in the spanning tree.



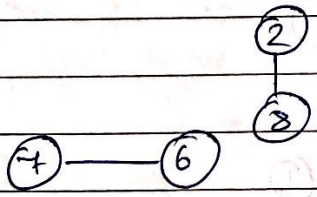
arranged all weights in increasing order.

weight	Src	Dst.	
1	7	6	✓
2	8	2	✓
2	6	5	✓
4	0	1	✓
4	2	5	✓
6	8	6	✗
7	2	3	✓
7	7	8	✗
8	0	7	✓
8	1	2	✗
9	3	4	

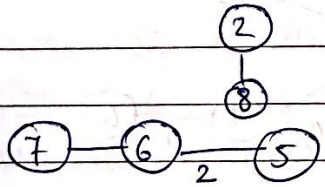
1.)



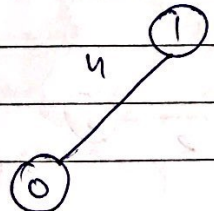
2.)

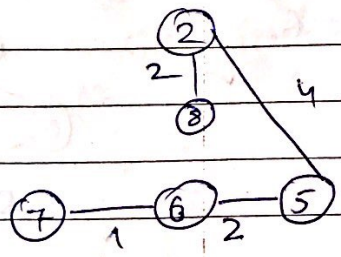


3.)

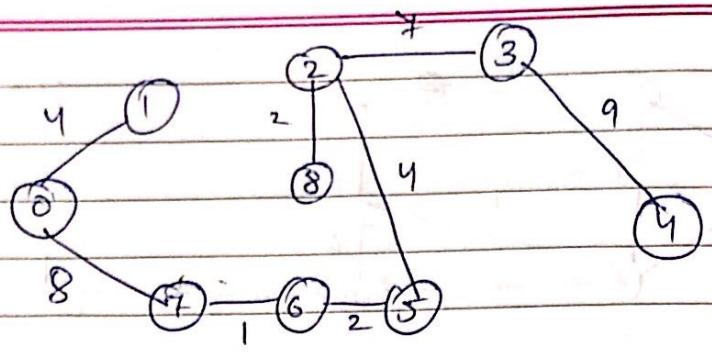


4.)

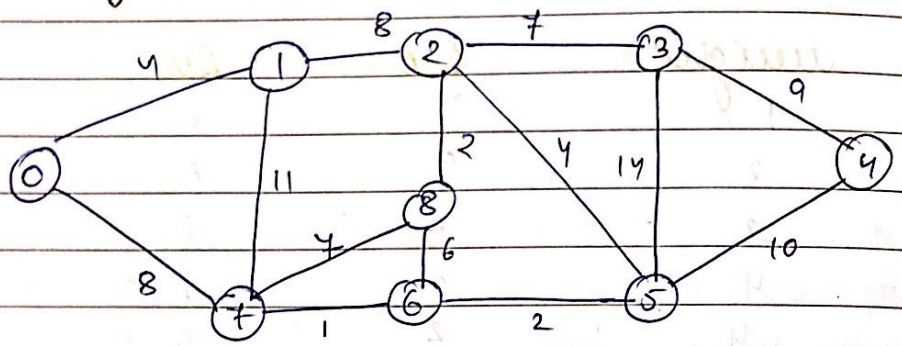




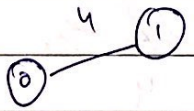
5.)



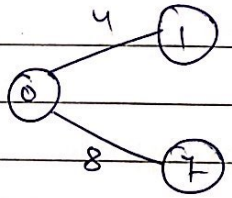
* Prim's Algorithm.



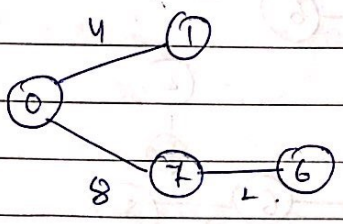
1.) 0-1 = 4
0-7 = 8
min. edge weight 4.



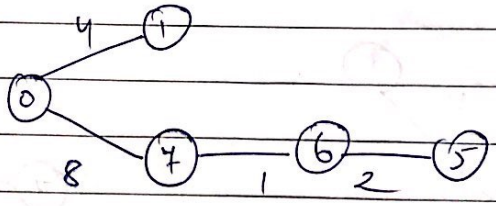
2.) 0-7 = 8 ✓
1-2 = 8
1-7 = 11



3.) 7-8 = 7
7-1 = 11
7-6 = 1 ✓

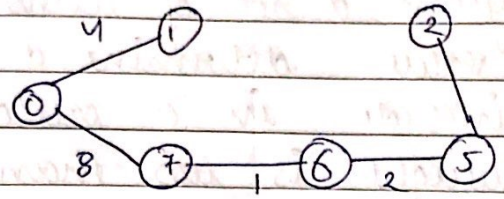


4.) 6-8 = 6
6-5 = 2 ✓

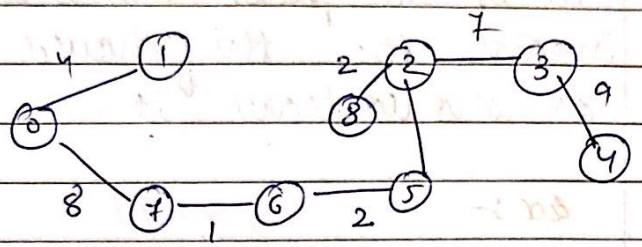


Page No.:
Date:

5.) 5-2 = 4 ✓
5-3 = 14
5-4 = 10



6.) 2-3 = 7 ✓
2-1 = 8 ✗



7.) 3-5 = 14 ✓
3-4 = 9 ✓

Dynamic Programming :-

It is similar to divide and conquer in breaking down the problem into smaller possible sub-problems. But these problems are not solved independently rather results of these smaller sub-problems are remembered and used for similar or overlapping sub-problem.

→ In contrast to greedy algorithms where local optimization is addressed, dynamic algorithms are motivated for an overall optimization of the problem

→ In contrast to divide & conquer algorithm where solutions are combined to achieve & overall solution dynamic algorithms use the o/p of a smaller sub-problem & then try to optimize a bigger sub-problem. Dynamic algorithms use memorization table to remember the o/p of already solved sub-problems.

(1) 0/1 Knapsack Problem:-

Given a set of items each with a weight & a value determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit & the total value is as large as possible. In 0/1 knapsack items cannot be broken which means the thief should take the item as a whole or should leave it.

ex:-

recursive soln

$$c[i,w] = \begin{cases} 0 & \text{if } w=0 \text{ or } i=0 \\ c[i-1,w] & \text{if } w_i \geq 0 \\ \max\{v_i + c[i-1, w-w_i], c[i-1,w]\} & \text{if } i>0 \text{ \& } w>w_i \end{cases}$$

Algorithm:-

1) Initialization
set $i \leftarrow n$
set $w \leftarrow W$

2) while($i>0$ & $w>0$)
do if $c[i,w] = c[i-1,w]$ then item i is not part of knapsack)
set $c[i,w] \leftarrow c[i-1,w]$
else
item i is part of knapsack)
set $c[i,w] \leftarrow c[i-1,w]$

3) exit

Page No.:
Date:

Eg: $W = 6$, $n = 3$

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

$(V_1, V_2, V_3) = (1, 2, 4)$

1.) $i \rightarrow 1$ to ~~m~~ n

$w \rightarrow 1$ to W

2.) $i = 1$

$w = 1$, $w_1 = 2$

$w_1 > w$.

$C[1,1] \leftarrow C[0,1]$

if $C_1 + C[0,0] > C[0,2]$

$C_1 + 0 > 0$

$C(1,2) \leftarrow 1$.

3.) $i = 1$, $w = 2$, $w_1 = 2$

$w = w_1$.

i / w	0	1	2	3	4	5	6	
w_i 0	0	0	0	0	0	0	0	v_i
2 1	0	0	1	1	1	1	1	1
3 2	0	0	1	2	2	3	3	2
3 3	0	0	1	4	4	5	6	4

② Longest common Subsequence (LCS) -

Given 2 sequences find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order but not necessarily ~~contiguous~~ contiguous.

Eg:

$S_1 =$ | A | B | C | A | A | B | D | A | D |

$S_2 =$ | D | B | D | A | B | C | A | D |

BABAD

Page No :
Date :

Algorithm LCS_Length :-

Given the sequence $X[1..M]$ & $Y[1..N]$

1.) Set $M \leftarrow$ length $[X]$
$n \leftarrow$ length $[Y]$

2.) If length of one of sequence is 0 then $c[i,j] = 0$
for $i \leftarrow 0$ to M.
set $c[i,0] \leftarrow 0$
for $j \leftarrow 0$ to n.
set $c[0,j] \leftarrow 0$

3.) A recursive soln
Let $c[i,j]$ be length of LCS of sequences.

$$c[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0. \\ c[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(c[i,j-1], c[i-1,j]) & \text{if } i,j>0 \text{ and } x_i \neq y_i \end{cases}$$

4.) computing cost of LCS

for $i \leftarrow 1$ to M.
do for $j \leftarrow 1$ to n.
do if $x_i = y_j$ then
set $c[i,j] \leftarrow c[i-1, j-1] + 1$
& $b[i,j] \leftarrow$ "↖"
else
if $c[i-1, j] \geq c[i, j-1]$ then
set $c[i,j] \leftarrow c[i-1, j]$.
& $b[i,j] \leftarrow$ "↑"
else

Set $c[i,j] \leftarrow c[i,j-1]$
& $b[i,j] \leftarrow$ "$\leftarrow$".
Return table c & b and exit.

Example:

$X = \langle MLNOM \rangle$
$Y = \langle MNOM \rangle$

① Length [X] = 5, M = 5, n = 4
Length [Y] = 4

② construct a table of M x n

x_i	y_i	M	N	O	M
	0	1	2	3	4
	0	0	0	0	0
M	1	0	1↑	←1	←1	↑1
L	2	0	1↑	←1	←1	←1
N	3	0	1↑	2↑	←2	←2
O	4	0	1↑	2↑	3↑	←3
M	5	0	2↑	←2	3↑	(4)↑

Algo print_LCS
Table 'b' is the i/p, x is one of sequences i=M & j=n where m & n are length of sequences X & Y respectively
1.) If i=0 or j=0 there is no LCS.
2.) If $b[i,j] =$ "↑" then
Recursively call print_LCS with.
$i \leftarrow i-1$
$j \leftarrow j-1$
print x_i
else if $b[i,j] =$ "↑" then
$i \leftarrow i-1$

else
recursively call print_LCS with
$j \leftarrow j-1$.

3) exit

eg:- c[i,j] => c[5,4] = 4
b[5,4] = "↑"
i = 4, j = 3 = M
b[4,3] = "↑" = O
b[3,2] = "↑" = N
b[2,1] = "↑"
i = 1
b[1,1] = "↖" = M
MNOM

Problem 3: Matrix chain multiplication:-

Given a chain of 'n' Matrices $(A_1, A_2 \cdots A_n)$ to be multiplied the matrix required to be fully parenthesized to resolve ambiguities. The product of parenthesization is same where many ways of parenthesization is possible. Different ways have different cost involved, one with minimum cost is optimized. So, the problem is to parenthesize the product $A_1 A_2 \cdots A_n$ such that it minimize the computation time.

Matrices: $A_1 = 2 \times 3$, $A_2 = 3 \times 4$, $A_3 = 4 \times 3$ } Dimensions

→ To multiply 2 matrices.
. columns of 1st matrix = rows of 2nd matrix.

If $A_1 = p \times q$
$A_2 = q \times r$
resultant product matrix dimension $p \times r$
computation time pqr.

→ Example :- $(A_1(A_2A_3))$

1st parenthesization :

$A_2 \times A_3 = 3 \times 4 \times 3 = 36$ } (3×3)
$A_1(A_2A_3) = 2 \times 3 \times 3 = 18$ }

Total no. of scalar multiplication
$= 36 + 18 = 54$

2nd parenthesization :

$((A_1A_2)A_3)$

$A_1A_2 = 2 \times 3 \times 4 = 24 = 2 \times 4$

$((A_1A_2)A_3) = 2 \times 4 \times 3 = 24$

Total no. of scalar multiplication
$= 24 + 24$
$= 48$

→ Algorithm involves 4 steps :-

1.) for a chain of matrices $A_i, A_{i+1} \dots A_j$.
a.) If $i = j$, no parenthization needed.
b.) If $i < j$, split the product b/w A_k & A_{k+1} for some $i \le k \le j$
first compute $A_i, A_{i+1} \dots A_k$ & then $A_{k+1} \dots A_j$
& then multiplying both to produce final product.

Cost of multiplying given chain of matrices = cost $(A_i \dots A_k)$ + cost $(A_{k+1} \dots A_j)$ + multiplying two resultant matrices.

2.) A recursive soln
Let $m[i, j]$ be minimum no of scalar multiplication

require to compute $A_i \cdots A_j$

a.) if $i = j$ $\quad m[i,j] = 0$

b.) if $i < j$

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

for various value of k, we calculate $m[i,j]$ & select the one with minimum value.

3.) compute optimal cost

→ use tabular bottom up approach.

→ Two auxiliary tables $M[1 \cdots n, 1 \cdots n] = m[i,j]$ costs

$S[1 \cdots n, 1 \cdots n] =$ index of k.

1.) n = no. of matrices multiplied.

$n \leftarrow$ length[p] - 1

2.) for single matrix. Scalar multiplication is zero.

for $i \leftarrow 1$ to n

set $m[i,j]$.

for sub-problems of matrix chain length l do following step:

for $l \leftarrow 2$ to n

do for $i \leftarrow 1$ to $n - l + 1$

set $j \leftarrow i + l - 1$

$m[i,j] \leftarrow \infty$

for $k \leftarrow i$ to $j - 1$

set $q \leftarrow m[i,k] + m[k+1, j] + p_{i-1} p_k p_j$

if $q < m[i,j]$ then

$m[i,j] \leftarrow q$

$s[i,j] \leftarrow k$

Return auxiliary table (arrays) m & s & exit.

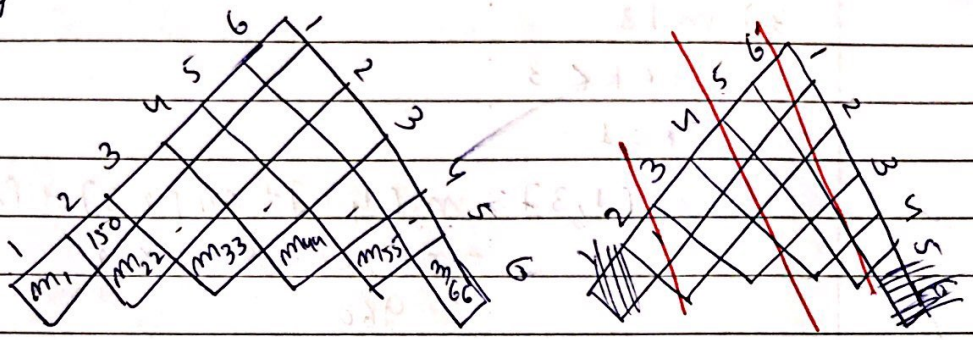
Example: find an optimal parenthesization of a matrix chain product whose sequence of dimension is.

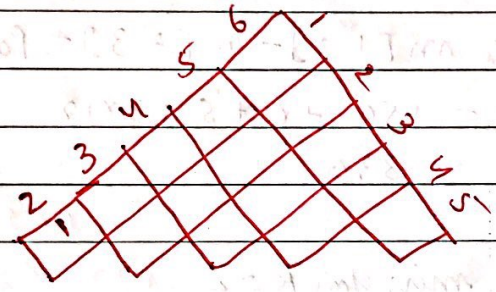
$\langle 5, 10, 3, 12, 5, 50, 6 \rangle$

$P_0 = 5, P_1 = 10, P_2 = 3, P_3 = 12, P_4 = 4, P_5 = 50, P_6 = 6.$

Sol^n. Length [p] = 7 n = length [p] - 1 = 6.

1.) Auxillary table m & S need to be constructed.





1) If $i = j$

$m_{11} = m_{22} = m_{33} = m_{44} = m_{55} = m_{66} = 0$

2.) for $i < j$

Initially $m[i, j] = \infty$

$m_{12} = 1 \le k < 2$

for $k = 1$

$m_{12} = \{ m[i, k] + m[k+1, i] + P_{i-1} P_k P_j \}$

$= \{ m[1,1] + m[2,2] + P_0 P_1 P_2 \}$

$= [0 + 0 + 5 \times 10 \times 3 \} = 150$

$s[1,2] = K = 1$

similarly

$m[2,3] = 360, \; s[2,3] = K = 2$

$m[3,4] = 180, \; s[3,4] = K = 3$

$m[4,5] = 3000, \; s[4,5] = K = 4$

$m[5,6] = 1500, \; s[5,6] = K = 5$

3.) $m|3$

$1 \le K < 3$

$K = 1$

$m[1,3] = m[1,1] + m[2,3] + P_0 P_1 P_3$.

$= 0 + 360 + 5 \times 10 \times 12$

$= 960$

$K = 2$

$m[1,3] = m[1,2] + m[3,3] + P_0 P_2 P_3$

$= 150 + 0 + 5 \times 3 \times 12$

$= 330$

$m[1,3] =$ min for $K = 2$

$s[1,3] = K = 2, \; m[1,3] = 330$.

Similarly $m[2,4] = 330 \quad s[2,4] = K = 2$

$m[3,5] = 930 \quad s[3,5] = K = 4$.

$m[4,6] = 1860 \quad s[4,6] = K = 4$.

4.) $m|4$

$1 \le K < 4$

$K = 1 \quad m[1,4] = m[1,1] + m[2,4] + P_0 P_1 P_4$.

$= 0 + 330 + (5 \times 10 \times 5)$

$= 580$

$K = 2 \quad m[1,4] = 405$

$K = 3 \quad m[1,4] = 630$

$m[1,4] = 405, \; s[1,4] = 2$

Page No.:
Date:
YOUVA

Similarly,

$m[2,5] = 2430 \quad s[2,5] = 2$

$m[3,6] = 1770 \quad s[3,6] = 4.$

5.) $m15 \quad 1 \le k < 5$

$k=1 = 4930$

$k=2 = 1830$

$k=3 = 6330$

$k=4 = 1655$

$m[1,5] = 1655$

$k=4$

Similarly,

$m[2,6] = 1950$

$s[2,6] = 2.$

6.) $m[1,6]$

$k=1 = 2250$

$k=2 = 2010$

$k=3 = 2550$

$k=4 = 2055$

$k=5 = 3155$

$m[1,6] = 2010$

$s[1,6] = k = 2$

4.) Constructing optimal solution

given table s with indices i, j

1.) If there is a single matrix ie.

$i = j$ then

print Ai

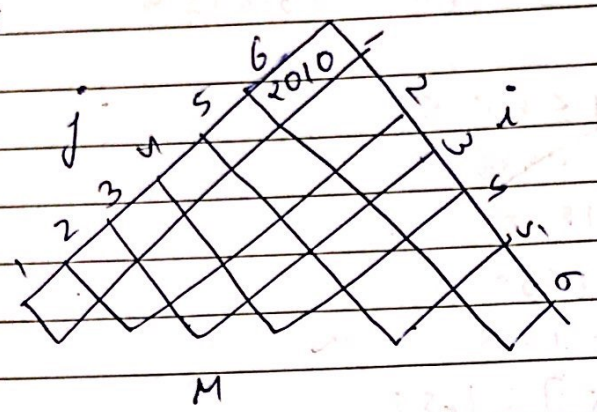
2.) if $i \neq j$

print("recursively call print_optimal_parens(s, i, s[i,j]

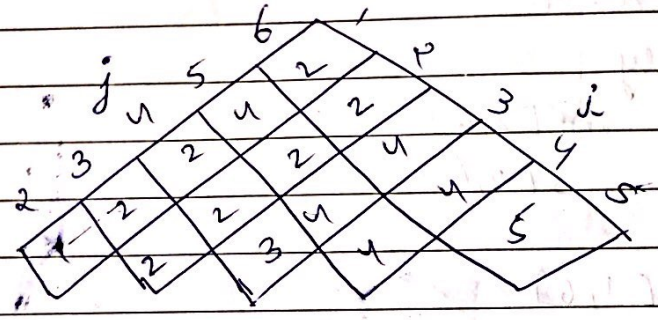
print_optimal_parens(s, s[i,j]+1, j)

3.) exit

eg:



M



S

soln. print_optimal_parens(s, 1, s[1,6])

= print_optimal_parens(s, s[1,6]+1, 6)

=> (s, 1, 2) (s, 3, 6)

=> ((s, 1, s[1,2])(s, 2, 2)) ((s, 3, s[3,6])(s, s[3,6]+1, 6))

=> ((s, 1, 1)(s, 2, 2))((s, 3, 4)(s, 5, 6))

i=j => (A1, A2) [(s, 3, 3)(s, 4, 4)(s, 5, 5)(s, 6, 6))]

=> (A1 A2)[(A3 A4)(A5 A6)]

Solution check :-

<5, 10, 3, 12, 5, 50, 6>

$A_1 = 5 \times 10$	$A_1A_2 = 5 \times 10 \times 3 = 150$	Dimension = 5 x 3 = C
$A_2 = 10 \times 3$	$A_3A_4 = 3 \times 12 \times 5 = 180$	Dim = 3 x 5 = B
$A_3 = 3 \times 12$	$A_5A_6 = 5 \times 50 \times 6 = 1500$	Dim = 5 x 6 = D
$A_4 = 12 \times 5$	$BD = 3 \times 5 \times 6 = 90$	Dim = 3 x 6 = E
$A_5 = 5 \times 50$	$CE = 5 \times 3 \times 6 = 90$	Dim = 5 x 6 = F
$A_6 = 50 \times 6$	Total = 150 + 180 + 1500 + 90 + 90 = 2010	

Unit-5

* Branch & Bound :-

→ Bounding functions are used to help for avoiding the generation of sub-trees that do not contain an answer node.

→ A branch & bound method searches a state space tree using any search mechanism in which all the children of current node are generated before another node is expanded.

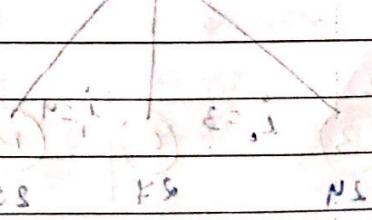
→ Each answer node has a cost associated with it & minimum cost answer node is to be found.

→ At each stage of tree exploration bound for particular node is calculated, the bound indicates how far we are from solution.

→ If at any node we see that further expansion of node will give worst solution, we kill that node i.e. not expanded further.

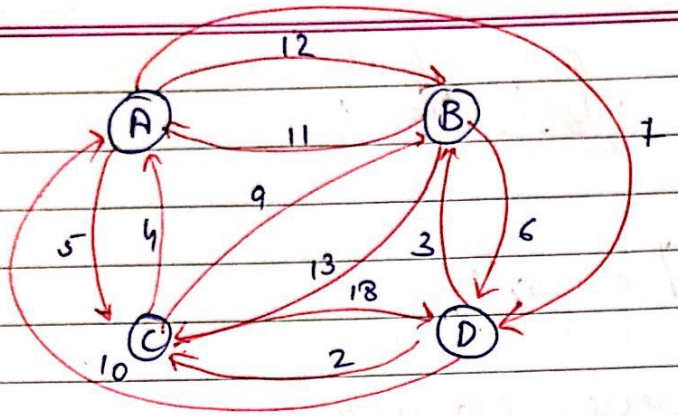
* Travelling Salesperson Problem :-

Problem states that a salesperson wants to visit a certain number of cities, his problem is to select such a route that starts from his home city passes through each city once & returns to his home city in shortest possible distance or least cost or time.



Page No.:
Date:

Example :



Cost matrix :

	A	B	C	D
A	∞	12	5	7
B	11	∞	13	6
C	4	9	∞	18
D	10	3	2	∞

$R_1 = R_1 - 5$

$R_2 = R_2 - 6$

$R_3 = R_3 - 4$

$R_4 = R_4 - 2$

$C_2 = C_2 - 1$

	A	B	C	D
A	∞	7	0	2
B	5	∞	7	0
C	0	5	∞	14
D	8	1	0	∞

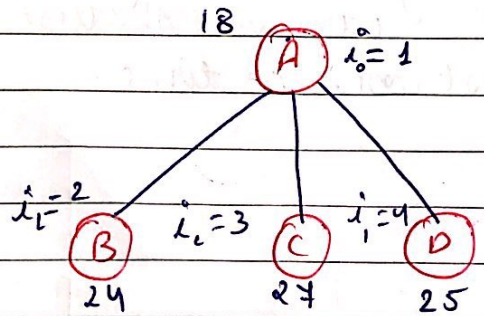
Reduced cost matrix :

RCM =

	A	B	C	D
A	∞	6	0	2
B	5	∞	7	0
C	0	4	∞	14
D	8	0	0	∞

$L = 5+6+4+2+1$

$L = 18$



Page No.:
Date:

★ Path(1, 2); node 2.

Ist row, 2nd column (2,1) = ∞

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	7	0
C	0	∞	∞	14
D	8	∞	0	∞

Total cost = 18+6 = 24.

path 1,3, node

	A	B	C	D
A	∞	∞	∞	∞
B	5	∞	∞	0
C	∞	4	∞	14
D	8	0	∞	∞

$R_3 = R_3 - 4$

	A	B	C	D
A	∞	∞	∞	∞
B	5	∞	∞	0
C	∞	0	∞	10
D	8	0	∞	∞

$C_1 \rightarrow C_1 - 5$

	A	B	C	D
A	∞	∞	∞	∞
B	0	∞	∞	0
C	∞	0	∞	10
D	3	0	∞	∞

cost (1,3) nodes = 18+4+5+0 = 27.

path 1,4, nodes

	A	B	C	D
A	∞	∞	∞	∞
B	5	∞	7	∞
C	0	4	∞	∞
D	∞	0	∞	∞

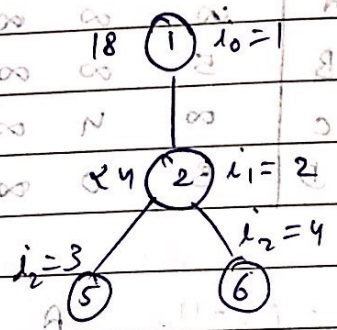
$M_2 - S =$

	A	B	C	D
A	∞	∞	∞	∞
B	0	∞	2	∞
C	0	4	∞	∞
D	∞	0	0	∞

Total cost = 18 + 5 + 2
= 25

RCM =

	A	B	C	D
A	∞	6	0	2
B	5	∞	7	0
C	0	4	∞	1
D	8	0	0	∞



Path 1, 2, node 2

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	7	0
C	0	∞	∞	14
D	8	∞	0	∞

Path 1, 2, 3 node 5

2nd row A
3rd column B
(3,4)(3,1) ∞

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	∞	∞	∞	14
D	8	∞	∞	∞

$R_3 \Rightarrow R_3 - 14$

$R_4 = R_4 - 8$ $\quad 24 + 14 + 8 + 7 = 53$

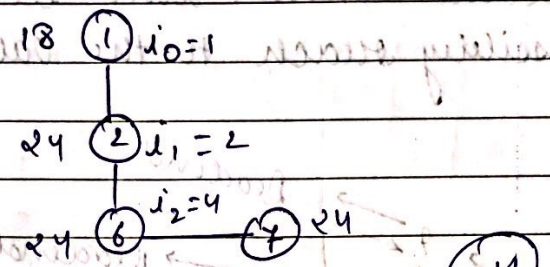
	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	0	∞	∞	0
D	∞	∞	∞	∞

Page No.:
Date:

path 1, 2, 4 node 6

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	0	∞	∞	∞
D	∞	∞	0	∞

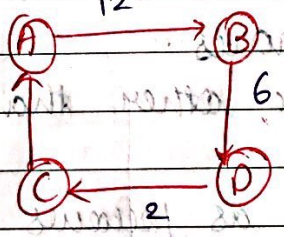
cost = 24 + 0
= 24



24

1, 2, 4, 3 node 7

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	∞	∞	∞	∞
D	∞	∞	∞	∞

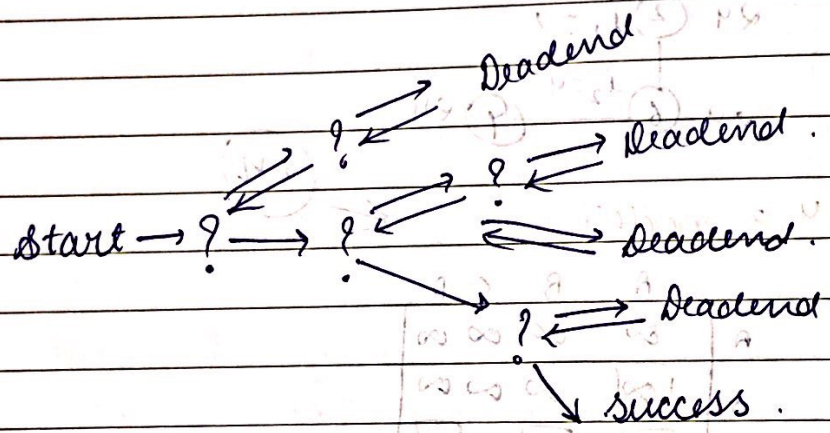


= 24.

Page No.:
Date:
YOUVA

* Backtracking :-

It is a general algorithm for finding all or some solutions to some computational problems that incrementally builds candidates to the sol^n, and abandons each partial candidates (backtrack) as soon as partial candidate sol^n leads to deadend. Algorithm backtracks to the point where candidate solution can be build again that can possibly reach to the valid solution.



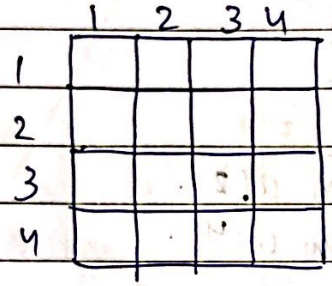
→ In the following figure
→ each non-leaf node in a tree is a parent of one or more other nodes.
→ Each node in the tree other than root has exactly one parent.
→ We explore each node as follows
→ to explore node 'n'
1.) if 'n' is a goal node return success.
2.) if 'n' is a leaf node return failure.
for each child c(n) explore c. if c was successful return success.
else
return failure.

→ Backtracking is a depth-first search (DFS) with any bounding function.

→ All solution using backtracking is needed to satisfy a complex set constraints.

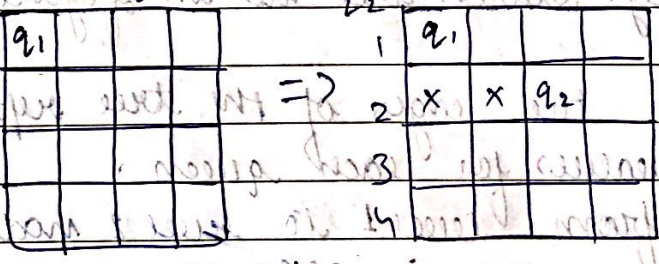
* 4- Queen Problem:-

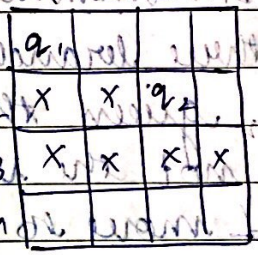
Given 4x4 chess board & we have to place 4 queens on this chess board such that they are in different rows, columns & diagonals.



4 queens q_1, q_2, q_3 & q_4. Let q_i placed in row r_i.

→ Starting from q_1, row 1, col 1 is first valid position –





So here we have at dead end this can't be valid soln & we need to backtrack last step & lets place q_2 in col 4.

q_3

	1	2	3	4
1	q_1			
2	x	x	x	q_2
3	x	q_3		
4				

$\Rightarrow$ q_4

	1	2	3	4
1	q_1			
2	x	x	x	q_2
3	x	q_3		
4	x	x	x	x

again we are at dead end and backtrack.

	1	2	3	4
1	x	q_1		
2	x	x	x	q_2
3	q_3	x	x	x
4	x	x	q_4	x

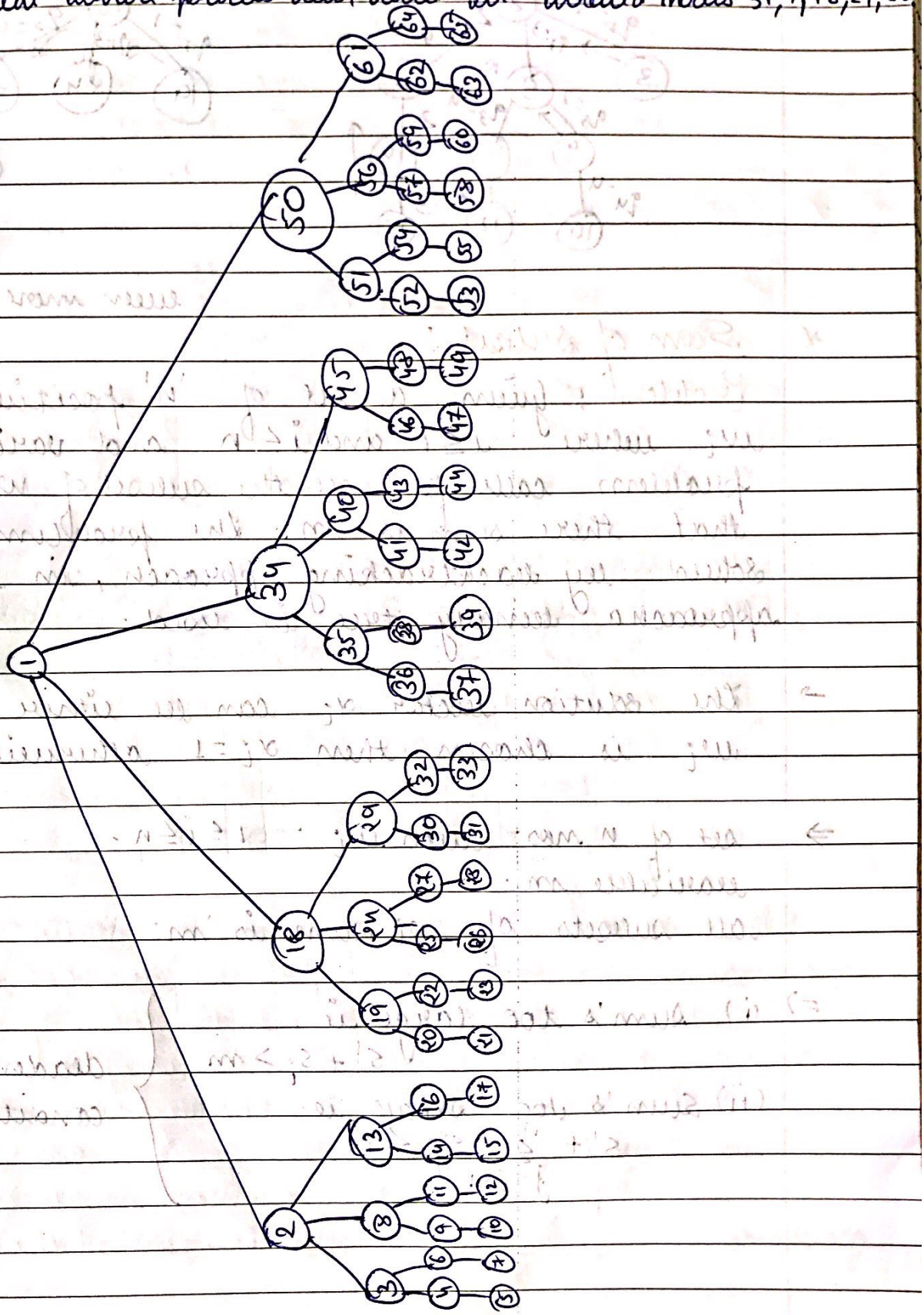
1.) q_3 placed only in col 2.
2.) q_2 placed only in col 4.

4 Queens Problem -

4 queens problem using backtracking can be done by search tree as in figure:-

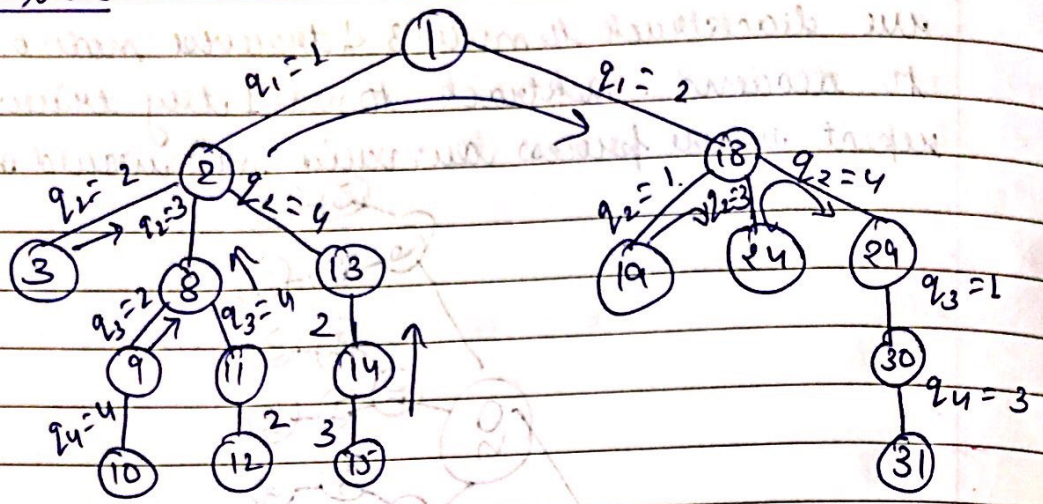
1.) Label on the edges of the tree represents possible column values for each queen.
2.) Edges from level 1 to level 2 nodes specify for q_1.
3.) Traverse the tree in DFS manner.
4.) Node 2 extends to three branches which gives values for q_2 given that $q_1 = 1$.
5.) As column 2 is not the valid position for q_2 backtrack to node 2 & move to node 8.
6.) Node 8 again extend to two branches, we are again at deadend. So backtrack to node 2 & place the queen q_2 at column 4.
7.) Move further to node 14 thus, placed q_2 at position 2
8.) The only branch left is towards node 15 which

places q_4 at column 3. which is not a valid position. If we backtrack to node 13 & traverse node 16 that would lead to deadend. Backtrack to node 1 & try traversing node 18 & repeat whole process our valid soln would nodes 31, 1, 18, 24, 80



Page No.:
Date:

Final tree:



**** more complicated

* **Sum of subset :-**

Problem : Given a set of 'n' positive number w_i where $i \geq 1$ and $i \leq n$ and variable 'm' problem calls for all the subset of w_i such that there sum is 'm'. The problem can be solved by backtracking approach, in this approach a binary tree is used.

→ The solution vector x_i can be either 0 or 1 if w_i is chosen then $x_i = 1$ otherwise $x_i = 0$.

⇒ set of n nos. with w_i : $1 \leq i \leq n$.
variable m.
all subsets of w_i sum is m.

⇒ (i) Sum's too large ie.
$$s' + s_i > m$$

(ii) Sum's too small ie.
$$s' + \sum_{j=i+1}^{n} s_j < M$$

} deadend condition

Page No.:
Date:

Example :-

$(w_1, w_2, w_3, w_4) = (11, 13, 24, 7)$, $m = 31$

Manually,

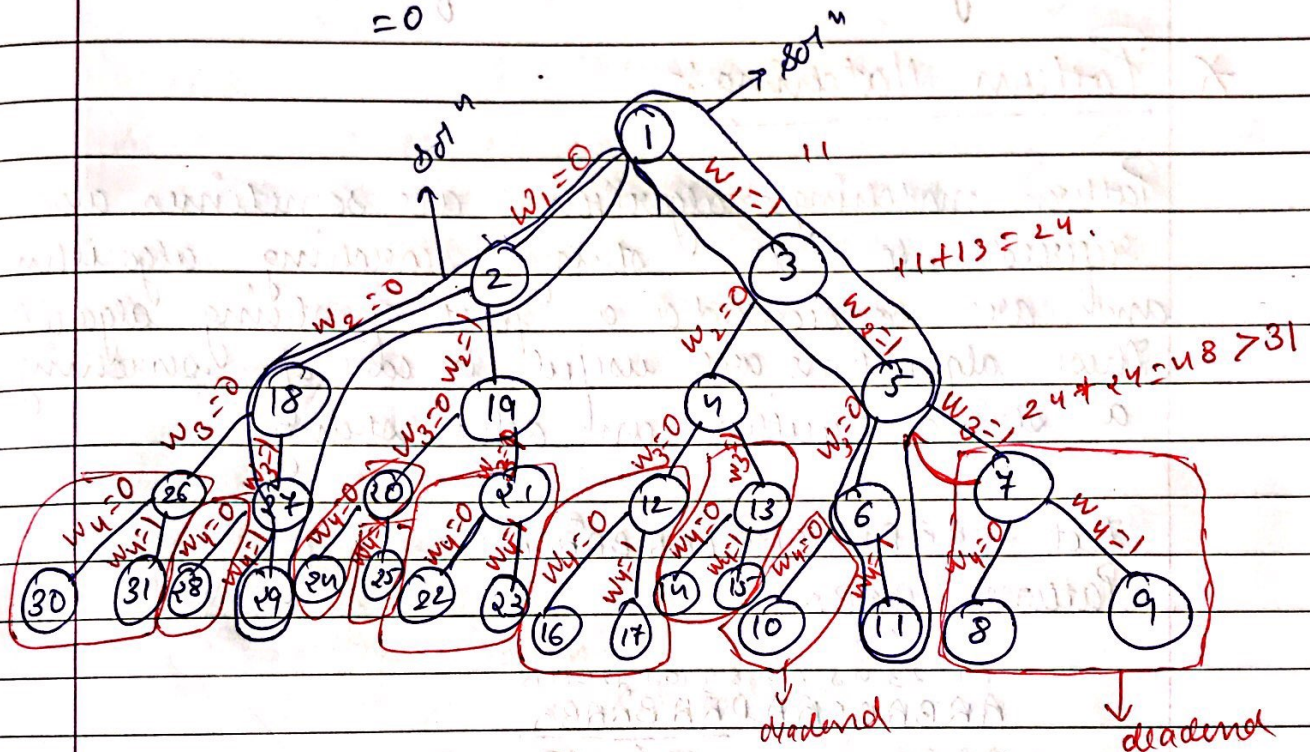
Subsets of w which makes sum = 31

(11, 13, 7) => sol^n vector (1, 2, 4)

(24, 7) (3, 4)

$x_i = 1$

$= 0$



$S_1 = 1101$ {11, 13, 7}

$S_2 = 0011$ {24, 7}.

→ Starting from node 1 expand it to node 2 & 3.

→ Selecting node 3, $w_1 = 1$, subset thus contains vector w_1 making sum = 11.

→ Node 3 is further expanded to node 4 & 5. to give a choice for w_2.

→ Using DFS we move forward to expand node 5 creating node 6 & 7.

→ Subset 7 now contains vectors w_1, w_2 making sum = 24.

→ If node 7 further selected for which $w_2 = 1$ sum becomes 48 i.e. $> m$ (31) this is an invalid soln & we need to backtrack to node 5. & choose node 6 where $w_3 = 0$.

→ Expanding 6 we get 10 & 11 choosing 11 $w_4 = 1$ making sum = 31 = m. This is valid solution similarly other soln can be found.

* Pattern Matching :-

Pattern matching algorithms are sometimes also referred to as string searching algorithm and are considered as a part of string algorithm. These algorithms are useful in case of searching a string within another string.

Text : AABAACAADAABAABA.

Pattern : AABA.

```
01 23 45 67 89 10 11 12 13 14 15
AABAACAADAABAABA.
AABA          ②   ③.
①
```

⇒ Naive Pattern Matching :

Algorithm :-

1) The naive algorithm finds all valid shifts using a loop that checks the condition $P[1 \ldots m] = T[s+1 \ldots s+m]$ for each of the ~~n-1+m~~ $n-m+1$ possible values of s. It can be interpreted graphically as sliding a template containing the pattern over the text uniform noting for which shifts all of the characters on the template equal the corresponding

characters in the text.

algo Naive-string-Matcher (T, P)

1.) $n \leftarrow$ length [T]

2.) $m \leftarrow$ length [P].

3.) for $s \leftarrow 0$ to $n-m$.

4.) do if $P[1 \ldots m] = T[s+1 \ldots s+m]$

5.) then print "pattern occurs with shift s.

Example:-

Show that comparison the naive string matches makes for pattern $P = 0001$ in text.

$T = 0000100010100001$

1.) $n \leftarrow$ length [T]

length [T] = 15

$n \leftarrow 15$

2.) $m \leftarrow$ length [P]

length [P] = 4

$m \leftarrow 4$

3.) for $s \leftarrow 0$ to $n-m$.

$s \leftarrow 0$ to $15-4$

$s \leftarrow 0$ to 11

4.) $P[1 \ldots 4] = T[0+1 \ldots 0+4]$

$P[1 \ldots 4] = T[1 \ldots 4]$

T = | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |

P = $ $ $ $ | 0 | 0 | 0 | 1 |

pattern unmatch.

$s = 1$

$P[1 \ldots 4] = T[1+1 \ldots 1+4]$.

$P[1 \ldots 4] = T[2 \ldots 5]$

T = | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |

| 0 | 0 | 0 | 1 |

pattern matched.

$s = 1$.

Page No.:
Date:
YOUVA

* Rabin Karp algorithm :-

Preprocessing + Matching Algo

. T = text . P = pattern

modulo = q d = base .

1.) Find the length of text t & pattern p .

$n \leftarrow$ length(T)

$m \leftarrow$ length(p)

2.) calculate h

$h \leftarrow d^{m-1}$.

3.) initialise decimal value P & t_0 .

$P \leftarrow 0$

$t_0 \leftarrow 0$

4.) Before matching is performed, preprocessing is done for string of length m .

for $i \leftarrow 1$ to m .

do $p \leftarrow (dp + p[i]) \bmod q$.

$t_0 \leftarrow (dt_0 + T[i]) \bmod q$.

5.) Matching is performed for every substring of t with p .

for $s \leftarrow 0$ to $n-m$.

if $p = t_s$.

then if $P[1 \ldots m] = T[s+1 \ldots s+m]$

then print " pattern matched for s .

if $s < n-m$ then

$t_{s+1} \leftarrow (d(t_s - T[s+1]h) + T[s+m+1] \bmod q$.

6.) Exit

Page No.:
Date:

Example:

eg $T = \langle 2,3,5,9,0,2,3,1,4,1,5,2,6,7,3,9,9,2,1 \rangle$.
$P = \langle 3,1,4,1,5 \rangle$ & mod q =
$\Rightarrow M = \text{length}[P] = 5$.

Now $T[s+1 \ldots s+M] = P[1 \ldots M]$
$0 \le s \le n-m$.
$p = P \bmod q = 31415 \bmod 13$.
$= 7$.
$s = 0$, $t_{s+1} = ?$
$t_s = 23590 \bmod 13$.
$= 8 \neq 7$.
pattern unmatched & shift is applied.

T =	2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1
	3	1	4	1	5														

$t_{s+1} = (d(t_s - T[s+1]h) + T[s+M+1]) \bmod 13$.
T_{s+1} = ~~to~~ higher order digit for . $= 23590$
$T(s+M+1)$ = new lower order digit $= 35902$
$t_{s+1} = t_2 = (10(23590 - 2 \times 10000) + 2) \bmod 13$.
$= 35902$

Similarly,

2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1

3	1	4	1	5

$s=2 \Rightarrow 59023 \bmod 13 = 3 \neq 7$
$s=3 \Rightarrow 90231 \bmod 13 = 11 \neq 7$.
$s=4 \Rightarrow 02314 \bmod 13 = 0 \neq 7$.
$s=5 \Rightarrow 23141 \bmod 13 = 1 \neq 7$
$s=6 \Rightarrow 31415 \bmod 13 = 7 = 7$
Pattern matched.

Compare

3	1	4	1	5	-	-

3	1	4	1	5

pattern matched with shift = 6.
This is done till $s = n-m$
$= 19-5$
$= 14$

Date: YOUVA

* KMP Matcher :-

Knuth - Morris - pratt (KMP) algorithm.

→ Rabin Karp String Matching

It is a string matching algorithm that seeks a pattern ie. a substring within a text by using hashing. This algorithm calculates a hash value for the pattern & for each 'm' character subsequence of text to be compared. It also involve both the steps of pattern matching ie. pre-processing & matching. It assumes each character to be a digit in radix-d notation.

Let P[1..n] be a pattern then P denotes its corresponding decimal value. d=10 for decimal value & d=2 for binary number.

→ For the given pattern p[1..n] 'p' denotes the decimal value & for the given text $t_s[1..n]$ t_s → denotes the decimal value. For the given pattern s is valid shift if and only if $p = t_s$. where $s \geq 0$ & $s \leq n-m$. P & t_s are calculated as $p = p \bmod q$.

* KMP matcher :-

→ It is a linear time string matching algorithm the array π allows the transition function to be computed as needed.

→ The prefix-function π for a pattern encapsulates

knowledge about how the pattern matches against shift of itself.

→ This information can be used to avoid testing useless shifts in the naive pattern matching algorithm or to avoid pre-computation of transition function for a string matching automation.

Knuth Morris - Pratt (KMP) Algorithm :-

```
KMP matcher [T, P)
1.) n ← length [T]
2.) m ← length [P]
3.) π ← compute-prefix-function (p)
4.) q ← 0
5.) for i ← 1 to n.
6.) do while q > 0 & p[q+1] ≠ T[i]
7.) do q ← π[q]
8.) if p[q+1] = T[i]
9.) then q ← q+1
10.) if q = m.
11.) then print " pattern occurs in shift i-m
12.) q ← π[q].
```

→ compute prefix function :-

```
1.) m ← length [P].
2.) π[1] ← 0
3.) k ← 0
4.) for q ← 2 to m.
5.) do while k > 0 & p[k+1] ≠ p[q]
6.) if p[k+1] = P[q]
7.) then. k ← k+1
8.) π[q] ← k.
9.) return π
```

Page No.:
Date:

Eg: compute prefix funct^n π for pattern.

ababbabbabbababbabb.

length[P] = 19 som = 19.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P(i)	a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π(i)	0	0	1	2	0	1	2	0	1	2	0	1	2	3	4	5	6	7	8

Initialise m ← 19
2.) $\pi[1] = 0$
3.) k ← 0
4.) q ← 2 to 19

k = 0
q = 2

do while k > 0 & p[k+1] ≠ P[q]
0 > 0 & p[0+1] ≠ P[2]
× a ≠ b true
for $\pi[q] \leftarrow k$
$\pi[2] \leftarrow 0$

Eg: P <aabab>.
T <aaababaabaababaab.>.

1.) n ← length[T].
length[T] = 17.
n ← 17.

2.) m ← length[P].
length[P] = 5
m ← 5.

3.) π

q	1	2	3	4	5
P[q]	a	a	b	a	b
π[i]	0	1	0	1	0

4.) $q \leftarrow 0$

5.) for $i \leftarrow 1$ to n

6.) $i \leftarrow 1$ to 17.

$i=1$ do while $q>0$ & $p[q+1] \neq T[i]$.

$q=0$ $0>0$ & $p[i] \neq T[i]$.

$0>0$ & $a \neq a$ F.

7.) if $P[q+1]=T[i]$.

$P[1] = T[1]$.

$a = a \cdot T$

$q \leftarrow q+1$

$q \leftarrow 0+1$

$q \leftarrow 1$.

$q = m$.

$1 \neq 5$ F.

$q=1$, $i=2$

$1>0$. $p[2] \neq T[2]$.

$1>0$ & $a \neq a$ F.

$P[q+1] = T[i]$

$a = aT$

$q \leftarrow 2$

$q = m$.

$2 \neq 5$ F.

$q=2$, $i=3$

$2>0$ & $p[3] \neq T[3]$.

$2>0$ & $b \neq a$ T.

$q \leftarrow \pi[q]$.

$q \leftarrow \pi[2]$

$q \leftarrow 1$.

inner loop condition

$q=1$, $i=3$

$1>0$ & $p[2] \neq T[3]$.

$a \neq a$ F.

$P[2] = T[3]$.

$a = aT$

Page No.:
Date:
YOUVA

$q \leftarrow 2$.
$q = m$
$2 \neq m$ F.

$i = 4$
$q = 2$

$2 > 0$ & $p[3] \neq T[4]$.
$2 > 0$ & $b \neq b$ F.
$P[3] = T[4]$
$b = bT$.
$q \leftarrow 3$.
$q = m$.
$3 \neq 5$ F.

$i = 5$
$q = 3$

$3 > 0$ $p[4] \neq T[5]$.
$3 > 0$ & $a \neq a$ F.
$p[4] = T[5]$.
$a = aT$.
$q \leftarrow 4$
$q = m$.
$4 \neq 5$ F.

$i = 6$
$q = 4$

$4 > 0$ & $p[5] \neq T[6]$
$4 > 0$ & $b \neq b$ F.
$P[5] = T[6]$
$b = bT$.
$q \leftarrow 5$
$q = m$.
$5 = 5$ T.

print pattern occurs with shift $i - m$.
print pattern occurs with shift 1.

$q \leftarrow \pi[q]$
$q \leftarrow \pi[5]$
$q \leftarrow 0$

i=7
q=0

0>0 ∧ P[i] ≠ T[7]

0>0 ∧ a ≠ a F.

P[L] = T[7]

a = a T.

q ← L

q = m. L ≠ 5 F.

i=8
q=L

L>0 ∧ P[2] ≠ T[8]

L>0 ∧ a ≠ a F.

P[2] = T[8].

a = a T.

q ← 2

q = m.

2 ≠ 5 F.

* Boyer's Moore method for pattern Matching:-

→ ① This is the most effecient string matching algorithm in usual applications.

→ ② The algorithm scans the characters of the pattern from right to left beginning with the rightmost character ③ During the testing of possible placement of pattern P against text T a mismatch of text character T[i] = c with the corresponding pattern character Pj is handled as follows:-

① If c is not contained anywhere in T then shift the pattern P completely past T_i.

② Otherwise shift P until an occurence of character c in P gets aligned with T_i

1.) Bad match table
2.) Pattern matching rightmost of pattern to left.
3.) shift according to bad match.

1.) Bad match table:-

Value = length - index - 1 (every remaining letter = length)

Ex:-

pattern = 'team mast'

text = " WELCOME TO TEAMMAST"

TEAMMAST

index 0 1 2 3 4 5 6 7

length = 8

letter	T	E	A	M	S	*
value	7	6	5	4	1	8
	8		2	3		

T = 8-0-1

E = 8-1-1

A = 8-2-1

M = 8-3-1

M = 8-4-1

A = 8-5-1

S = 8-6-1

T = 8

Last letter = length

T: WELOME TO TEAMMAST

TEAM MAST

T = 8 = Match value

S=8 right space towards right.

T: WELCOME TO TEAMMAST.

P: TEAMMAST

S=2

see the value of S. if first letter is not match

mismatch value then corresponding value in text.

T: WELCOME TO TEAMMAST
TEAMMAST
S = 9.

Q.) T: TRUST HARD TOOTH BRUSHES
P: TOOTH.
01234

1.) Bad match table.

letter	T	O	H	*
value	4	3	5	5
	1	2		

T = 5-0-1
O = 5-1-1
O = 5-2-1
T = 5-3-1
H = 5

T: TRUST HARDTOOTH BRUSHES
P: TOOTH

S=1

T: TRUST HARD TOOTH BRUSHES
P: TOOTH

T: TRUST HARD TOOTH BRUSHES
P: TOOTH

write the no. shifts value of only the first character matched.

S=5

T: TRUST HARD TOOTH BRUSHES
P: TOOTH

S=1

T: TRUST HARD TOOTH BRUSHES
P: TOOTH

S=2

T: TRUST HARD TOOTH
P: TOOTH

S=1

T: TRUST HARDTOOTH BRUSHES
P: TOOTH

S=10

Page No.:
Date:

Unit-4

* Assignment Problems :-

Problem statement :- (Linear Assignment)
There are 'n' no. of agents & 'n' no. of tasks any agent can be assigned to perform incurring some cost that may vary depending on the agent task assignment it is required to perform all task by assigning exactly one agent to each task in such a way that total cost of assignment is minimised. If the no. of agents and task are equal & the total cost of the assignment for all task is equal to the sum of cost for each agent then the problem is called linear assignment problem. The most effecient algorithm to find optimal sol^n to an assignment problem is Hungarians method or Hungarian algo.

* Hungarian algorithm for Assignment Problem.

Step 1: Find the minimum value from each row subtract that minimum value from corresponding row in order to obtain new matrix

Step :2 Repeat step 1 for each column.

Step :3 Now some zeroes might be appearing in newly

constructed matrix. Draw minimum no. of lines that are required to cover all the zeroes in newly reduced matrix. These lines might be horizontal or vertical. If there are less than 'n' no. of lines drawn then goto step 4. otherwise optimal solution is present in the covered zeroes of matrix then goto step 5.

Step 4: Obtain smallest non-zero number from the reduced matrix say 'm' now subtract 'm' from each uncovered no. and add 'm' to each no. which is covered by two lines then repeat step 3.

Step 5: The total cost of assignment is calculated & it must be a minimized cost value. Each task must be assigned to only one agent.

Linear assignment problem (Hungarian algo)

Eg: Given cost matrix job.

	1	2	3	4
1	33	40	43	32
2	45	28	31	23
3	42	29	36	29
4	27	42	44	38

1) Subtract lowest value from each row.

	1	2	3	4	
1	1	8	11	0	-32
2	22	5	8	0	-23
3	13	0	7	0	29
4	0	15	17	11	-22

2.) Subtract lowest value from column which does not contain zero.

1	8	4	0
22	5	1	0
13	0	0	0
0	15	10	11

3.) Mark lines which contains max no. of zeros in rows.

1	8	4	0
22	5	1	0
13	0	0	0
0	15	10	11

4.) total no. of person = 4 lines.

Select minimum element from uncovered nos & subtract from remaining uncovered number & add to intersection.

Min no = 1

1	7	3	0
22	4	0	0
14	0	0	1
0	14	9	11

assign zero according to only single zero column.

	1	2	3	4
1	1	7	3	0
2	22	4	0	0
3	14	0	0	1
4	0	14	9	11

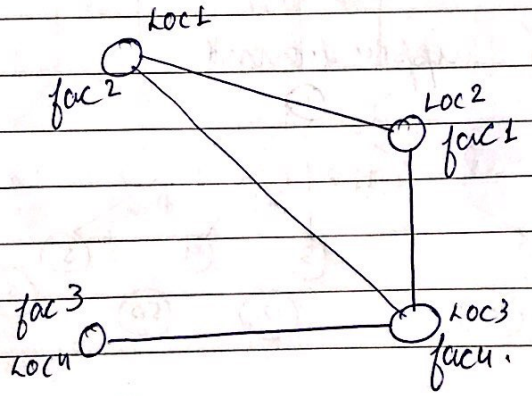
1-4 => 32
2-3 => 31
3-2 => 29
4-1 => 27
total = 119

* Quadratic Assignment Problem :- (QAP)

→ In QAP we are given a set of 'n' locations and 'n' facilities and told to assign each facility to a location.

→ There are n! possible assignments to measure the cost of each possible assignment we multiply the prescribed flow b/w each pair of facilities by the distance b/w there assigned locations and sum overall of the pair.

→ our aim is to find the assignment that minimises this cost.



=> Branch and bound method

	1	2	3	4
A	90	12	50	51
B	70	10	58	80
C	16	85	8	70
D	11	37	80	21

1.) find lower bound & upper bound for problem

Lowerbound = min (R, C)

R = sum value of min ~~row~~ from each row.

C = sum value of min from each column.

R = 12 + 10 + 18 + 11 = 41

C = 11 + 10 + 8 + 21 = 50

min (41, 50) = 41

upperbound = min(sum of diagonal element d_1, d_2)

D_1 = 90 + 10 + 18 + 21

= 129

D_2 = 51 + 58 + 85 + 11

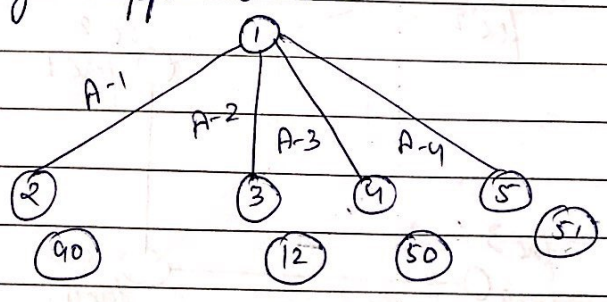
= 205

U.B. = min (D_1, D_2) = min(129, 205)

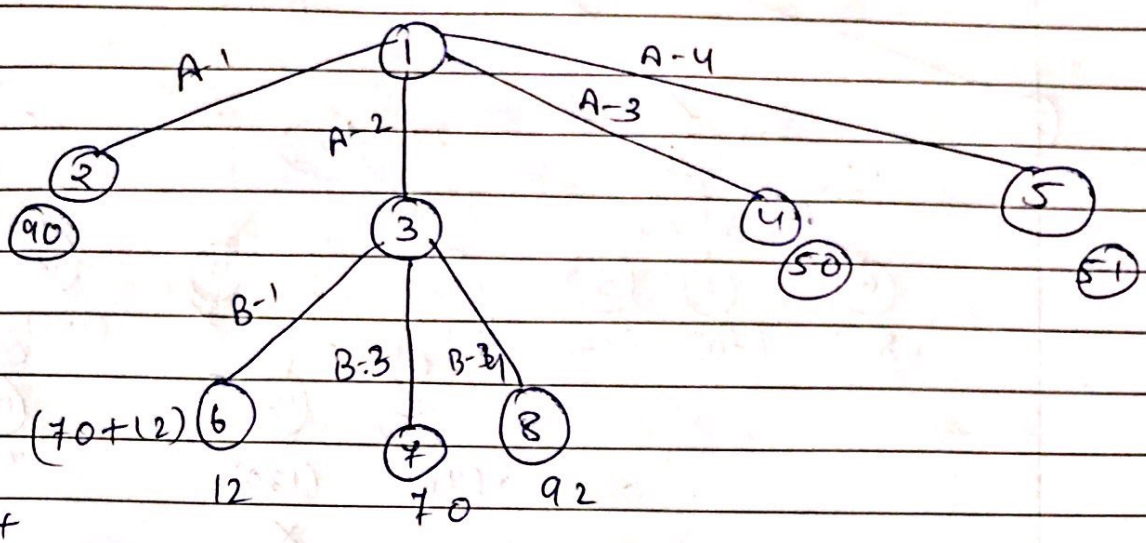
= 129

=> Sol^n using upper bound.

1.)

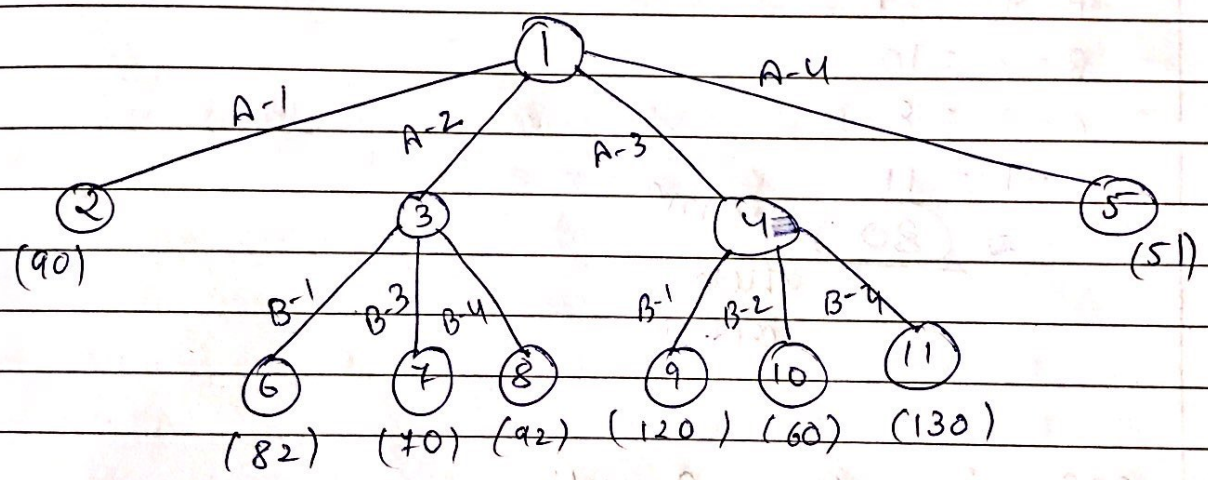


2.) Select min & explore

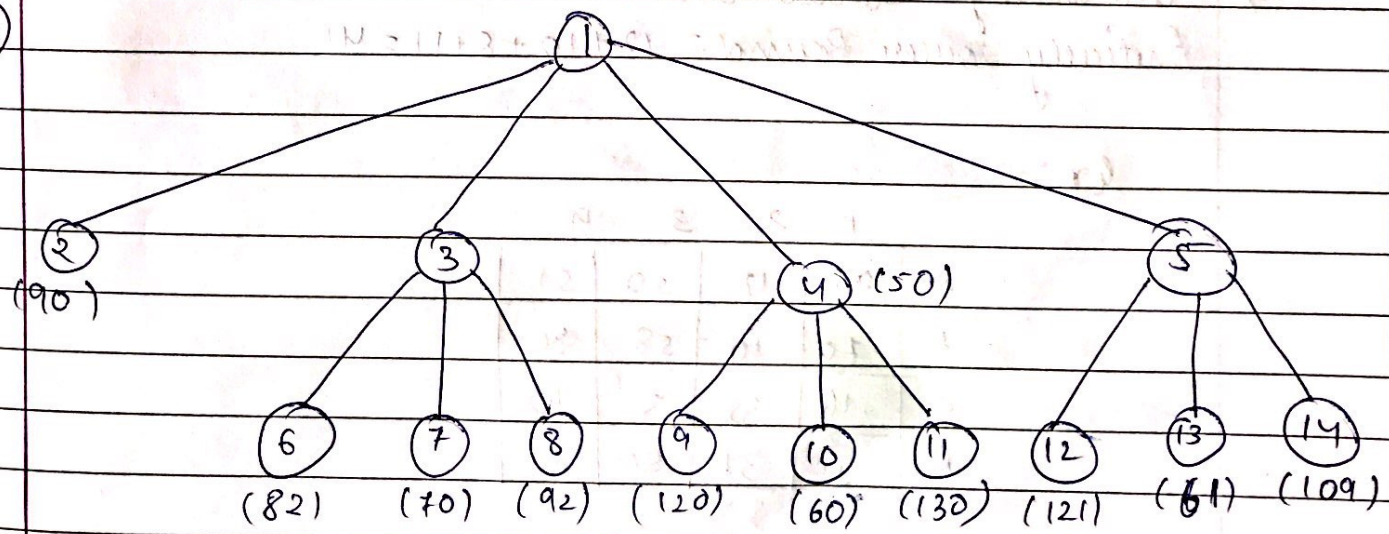


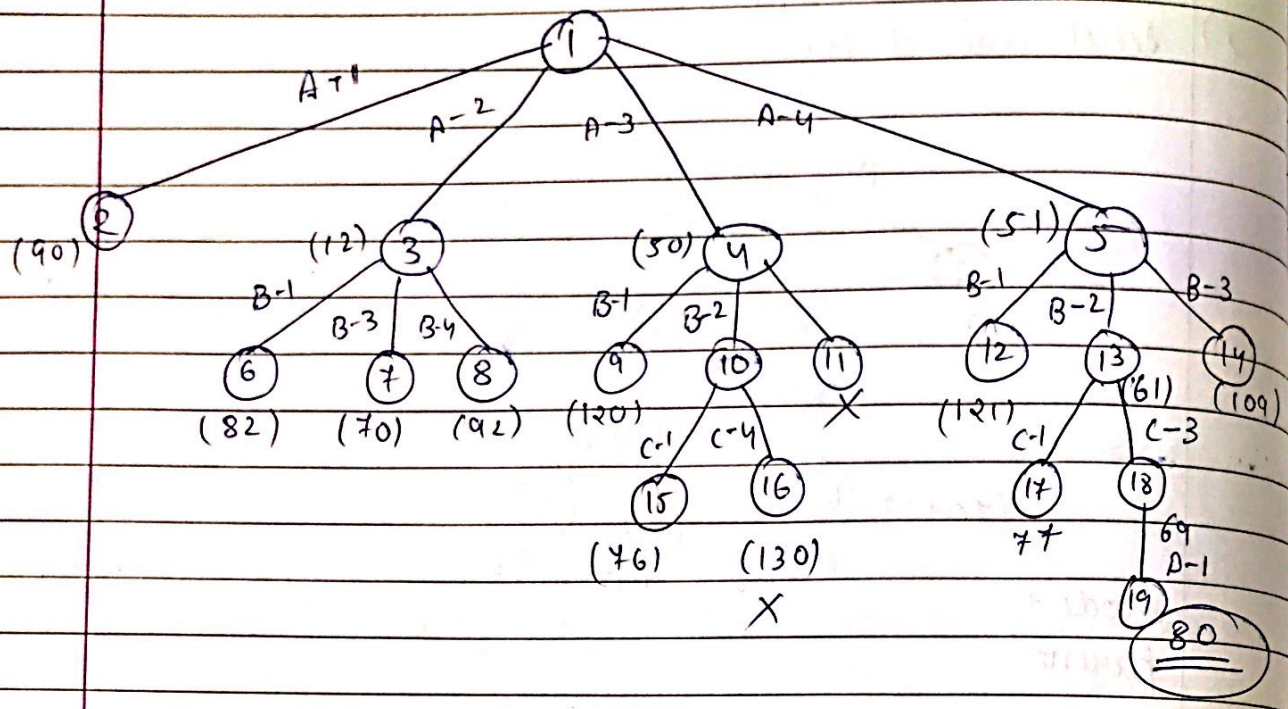
node + parent

3.)



4.)





A-4 = 51

B-2 = 10

C-3 = 8

D-1 = 11

80 same then correct

* BAP using lower Bound:-

Initially lower Bound = 12+10+8+11 = 41

Ex:-

	1	2	3	4
A	90	12	50	51
B	70	10	58	80
C	16	85	8	70
D	11	37	80	21