10/7/25

Design and Analysis of Algorithm

* Different types of algorithm design
  (i) Divide & Conquer
  (ii) Greedy
  (iii) Dynamic Programming

* Time Complexity
  (i) Big-oh (O)
  (ii) Big- Omega (Ω)
  (iii) Big- Theta (Θ)

* Types of algorithm—
  (i) Iterative Algos.
  (ii) Recursive Algos.

  → Three types of technique to find recursive
    (i) Back Substitution
    (ii) Recursive Tree
    (iii) Master's Theorem

How to compute T.C of Iterative Algorithm

* Frequency Count Technique

e.g

```
A1()
{ int a,b;
  Printf(---);
  :
}
```

PRINT ~~A~~
↳ DISPLAY

$T(A1) = 1$

If it doesn't contain any loop (doesn't matter how many many statement or if-else are there the Time Complexity will be O(1)
↓
Constant Time Complexity

e.g

```
A2()
{ int i, n, Sum;          → 1
  Sum = 0                 → 1
  for(int i=0; i<=n; i++){   → 1+(n+1)+n
      Sum += i;           → n
  }
  Print sum;              → 1
}
```

$3n + 5$

ignore Constant

$T(A2) = O(n)$ → Linear T.C

Assymptotic Notation

e.g

largest n or n+1

```
A3(){
 ----        ---> 1
 ----        ---> 1
 for(i=1; i<=n; i++){        ---> 1+(n+1)+n ---> n
   for(j=1; j<=n; j++){      ---> (1+(n+1)+n)n ---> n^2
     Statment 1;             ---> 1*(n*n) ---> n^2
     Statment 2;             ---> (n*n) ---> n^2
   }                         ~~ n + 3n^2 + 2
 }
}
```

$T(A3) = O(n^2)$ → quadratic

17/7/25

Properties

(i) Input
(ii) Output
(iii) Effectiveness
(iv) Definiteness
(v) Finiteness

- O → Max time needed (worst case)
- Ω → Min time needed (Best case)
- Θ → Average time needed (Average case)

Time Complexity of Iterative Algorithm

*
```
A3(){
  del 1;                          ---> 1
  for(i=1; i<=n/2; i++)           ---> n/2 + 1 ~~ n
  {
    S1;                           ---> n/2
    S2;                           ---> n/2
  }
  S3;                             ---> 1
  S4;                             ---> 1
  S5;                             ---> 1
}
```

$T(n) = 3n/2 + 4$
$= 3n/2 + 4$
$= O(n)$

*
```
A5(){
  d1;                             ---> 1
  for(i=1; i<n; i++){             ---> 1 + (n/2+1) + n/2
    S1;                           ---> n/2
    S2;                           ---> n/2
  }
  S3;                             ---> 1
}
```

$T(n) = 4n + 4$ → $O(n)$

$T(A5) = O(n)$

* for (i=n; i>=1; i=i-2/i--) $T(n) = O(n)$

* AG()

```
{
    d1;                                  ---> 1
    for (i=1; i<=n; i=i*2)               ---> log2 n
    {
        S1;                              log2 n
        S2;                              log2 n
        S3;                              log2 n
        S4;                              log2 n
    }
    S5;                                  ---> 1
    S6;                                  ---> 1
}
```

i	P
1	2
2	4
4	8
⋮	⋮
n/2	n

k

At kth iteration

$n = 2^k$

$k = \log_2 n$

$T(n) = O(\log_2 n)$

$T(n) = 5\log_2 n + 3$

$T(n) = O(\log_2 n)$

Logarithmic

23|7|25

Iterative Algorithm

Eg 6

```
P=0;
for (i=1; P<=n; i++){
    P=P+1;
}
```

$T(AG) = O(\sqrt{n})$

n = 100;

i	P
1	0+1 = 1
2	1+2 = 3
3	1+2 +3 = 6
4	1+2+3 +4 = 10
⋮	⋮
	P > n

Eg 7

```
for (i=1; i*i<=n; i++){
    Statement;
}
```

$T(A7) = O(\sqrt{n})$

→ $1+2+3+4+\cdots k = P$

$\frac{k(k+1)}{2} = P$

$\frac{k^2+k}{2} = P$

$P = n$

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

$\therefore k^2 \approx n$

$\because k = \sqrt{n}$

i	P = i×i
1	1
2	4
3	9
⋮	⋮
	k

$k > n$

$k = n$

$k^2 \cong n$

$k = \sqrt{n}$

$i^2 > n$

$i^2 = n$

$i = \sqrt{n}$

Eg 8

```
for (i=1; i<=n; i++){          ⟶ 2n+2
    S1;                        ⟶ n
    S2;                        ⟶ n
}
for (k=1; k<=n+1; k++){        ⟶ 2n+2
    S3;                        ⟶ n
    S4;                        ⟶ n
}
```

$8n+4$

$T(Ag) = O(n)$

Eg 9

```
p=0;
for (i=1; i<n; i=i*2){         ⟶ log2 n
    p++;                       ⟶ p = log2 n
}
for (j=1; j<p; j=j*3){         ⟶ log3 p
    statement;                 (p = log2 n)
}
```

$\log_3 p = \log_3(\log_2 n)$

$T(Ag) = O(\log_3 \log_2 n)$

$T(Ag) = \max(\log_2 n, \log_3 \log_2 n)$

$T(Ag) = O(\log_2 n)$

Overall

```
for (i=1; i<=n; i++){
    S1;
}
```
$O(n)$

```
i=1;
do {
    S1;
    i++;
} while (i<=n);
```
$O(n)$

```
i=1;
while (i<=n) {
    S1;
    i++;
}
```
$O(n)$

```
a=1;
while(a<b) {
    S1;
    a=a*2;
}
```
$a>b$

$a=1$

$a=1*2=2^1$

$a=2*2=2^2$

$a=2^2*2=2^3$

$T \to O(\log_2 b)$

$2^k = b$

$\therefore k = \log_2 b$

$k \to a = 2^k$

LAB 1 23/7/25

Linux → Redhat, Fedora, Mandrae, Ubuntu, Cant OS, Debian

::: → Terminal → $

```
$ gedit p1.c
$ mkdir CSE234057
$ cd CSE234057
$ mkdir DAALAB
$ cd DAALAB
$ cc p1.c   [compile] gcc/cc
```

mkdir → make folder
touch → create file

```
$ pwd (present wd)
$ /home/alian/
$ ls -l or ls -a   [to see hidden contents]
$ cc p1.c -o p1.o
$ -o p1.c
$ ./p1.o
```

$ vim p2.c → :x Enter
ESC : wqd → quit

- mkdir CSE234057 (small)
- cd CSE234057
- mkdir DAALab
- cd DAALab
- gedit P1.c [write code]
- cc P1.c [compile]
- cc P1.c -o p1.o
- ./p1.o

[output displayed]

{ Bubble Sort
Selection Sort
Matrix

LAB 1

LAB 2 30/7/25

Divide & conquer

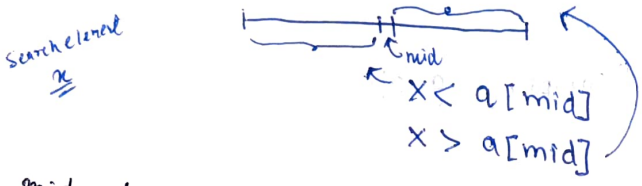
1. Binary Search (Recursive)
2. Quick Sort
3. Merge Sort

{ Linear Search
{ Binary Search

LAB 2

* Divide & conquer are a problem in paradise and it is subdivided and subdivided to form smaller problem & can be solved.

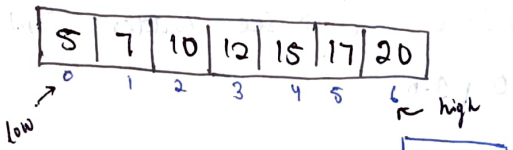
- Binary Search with Recursion



$mid = low + \frac{(high - low)}{2}$

$= 0 + \frac{6}{2}$

$= 3$



x = 15

- Binary Search with Recursion

```
BinSearchRec (a[], x, l, u){
    if (l == u){          a[u]
        if (x == a[l]){
            Print ("Search successful")
            return l;
        }
        else
            return -1;  // search unsuccessful
    }
    else {
        m = l + [(u-l)/2];
        if (x == a[m])
            return m; // search successful
    else if (x < a[m]){
        BinSearchRec (a[], x, l, m-1);
    }
    else {
        BinSearchRec (a[], x, m+1, u);
    }
}
```

```
BS{
  if (x == a[mid])
    return mid;
  else
    if (x < a[mid]
    BS (l, mid-1)
  else
    BS (mid+1, u)
  }.
```

E.g 10

```
Algo (){
    i=1, S=1;
    while (S<=n){
        i++;
        S = S+i;
        Pf("Hello");
    }
}
```

S>n
S≯n

S=1	3	6	10	15	21	...	n^2
i=1	2	3	4	5	6	....	k
Hello	1	2	3	4	5	....	k

$K(K+1)/2$

$S = n \geq \frac{k(k+1)}{2}$

$k = 2, 3, \ldots$

$S = 3, \ldots$

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

$k^2 \leq n$

$\therefore k \leq \sqrt{n}$

$T(A) = O(\sqrt{n})$

E.g 11

```
for(i=1; i<=n; i++){
    for(j=1; j<=i^2; j++){
        for(k=1; k<=n/2; k++)
        {
            stmt;
        }
    }
}
```

i=1	i=2	i=3
j=1	j=4=2^2	j=9=3^2
k=n/2	k=n/2	k=n/2
Stmt → $\frac{n}{2}*1$	$\frac{n}{2}*2^2$	$\frac{n}{2}*3^2$

$T = \frac{n}{2}*1^2 + \frac{n}{2}*2^2 + \frac{n}{2}\times 3^2 + \cdots \frac{n}{2}\times n^2$

$= \frac{n}{2} * (1^2+2^2+3^2+\cdots+n^2)$

$= \frac{n}{2} * \frac{n(n+1)(2n+1)}{6}$

$T(A) = O(n^4)$

n
j = n^2
k = n/2
$\frac{n}{2} * n^2$

E.g 12

$n = 2^{2^k}$

```
for(i=1; i<=n; i++){
    j=2;
    while(j<=n){
        j = j^2;
        Stmt;
    }
}
```

k=1	k=2	k=3
$n = 2^2 = 4$	$n = 2^{2^2} = 16$	$n = 2^{2^3} = 256$
j = 2, 4	j = 2, 4, 16	2, 4, 16, 256
Stmt → n*2	n*3	n*4

$n = 2^{2^k}$

$\therefore k = \log_2 \log_2 n$

k values

$n * (k+1)$ times

$n * (\log_2 \log_2 n + 1)$

$\therefore O(n \log_2 \log_2 n)$

Recursive Algorithm

(i) Back Substitution
(ii) Recursion tree
(iii) Masters Theorem

Eg 1

```
A(n){
  if(n>1)          ——— c
    return (A(n-1));
}
```

A(n-1) → T(n-1)

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

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

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

Put (2) in eq (1)

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

$= T(n-2) + 2C + C$ ---- (4)

Put (3) in (4)

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

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

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

```
if(n==1)
  return 1;
else
  return (A(n-1))
```

n = n-1

T(n-(n-1))

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

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

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

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

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

$T(n-2) = T(n-3) + C$ ---- (4)

Subs. eq 4 in eq 3

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

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

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

$= C + (n-1)C = n \cdot C$

EX-2

$T(n) = n + T(n-1)$, $n>1$ —(i)

$= 1$, $n=1$

$T(n-1) = (n-1) + T(n-1-1)$ [From (i)]

$= (n-1) + T(n-2)$ —(ii)

Substitute eq (ii) in (i)

$T(n) = n + (n-1) + T(n-2)$ —(iv)

$T(n-2) = (n-2) + T(n-3)$ [From (i)] —(iii)

Subs. eq (iii) in (iv)

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

Continuing this

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

$= n + (n-1) + (n-2) + (n-3) + \cdots + 3 + 2 + 1$

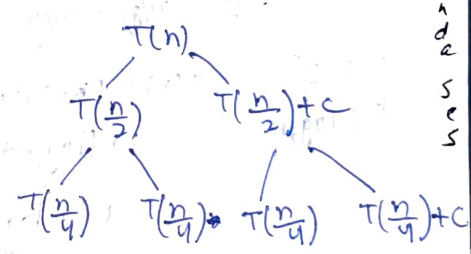
$= n(n+1)$ $\therefore T(n) = O(n^2)$

6/8/25

Time Complexity of Recursive Algorithm LAB 3

* Back Substitution Method

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



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

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

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

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

$= 8 \cdot T(n/8) + 4c + 2c + c$

$= 2^k \cdot T(n/2^k) + 2^{k-1} \cdot c + 2^{k-2} \cdot c + \cdots + 4c + 2c + c$

2^k $n = 2^k$ k^{th} term

$= 2^k \cdot T(1) + 2^{k-1} \cdot c + 2^{k-2} \cdot c + \cdots + 4c + 2c + c$

$= 2^k \cdot c + 2^{k-1} \cdot c + 2^{k-2} \cdot c + \cdots + 4c + 2c + c$

$= (2^k + 2^{k-1} + 2^{k-2} + \cdots + 4 + 2 + 1) \cdot c$ [G.P]

$= 2^k \cdot c \left(2^{k-1} + 2^{k-2} + \cdots + 4 + 2 + 1\right)$

$= \frac{2^k \cdot c (2^k - 1)}{(2-1)} \cdot c$ $= n \cdot c + (n-1)c$ $= 2nc - c$

$= nc + nc - c$ $= O(n)$

G.P

$a, ar, ar^2, \cdots ar^{n-1}$

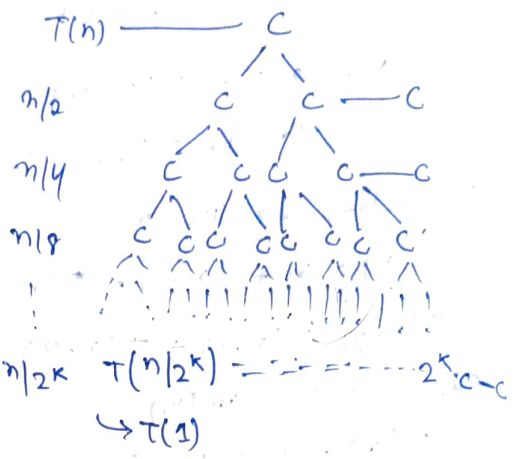
$\text{sum} = \frac{a(r^n - 1)}{r-1}$

Cormen (MIT) Sahani

bhanda ses

2. Recursion Tree Method

$$T(n) = \begin{cases} 2T(n/2) + c & ; n > 1 \\ c & ; \text{otherwise} \end{cases}$$



$T(n) = 2^k \cdot c + 2^{k-1} \cdot c + \cdots$

$8c + 4c + 2c + c$ (k terms)

$2^{k-1} = 1$, $\hookrightarrow n = 2^k$

$= 2^k \cdot c + (2^{k-1} + 2^{k-2} + \cdots 4 + 2 + 1)c$

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

$= n \cdot c + n \cdot c - c$

$= 2nc - c$

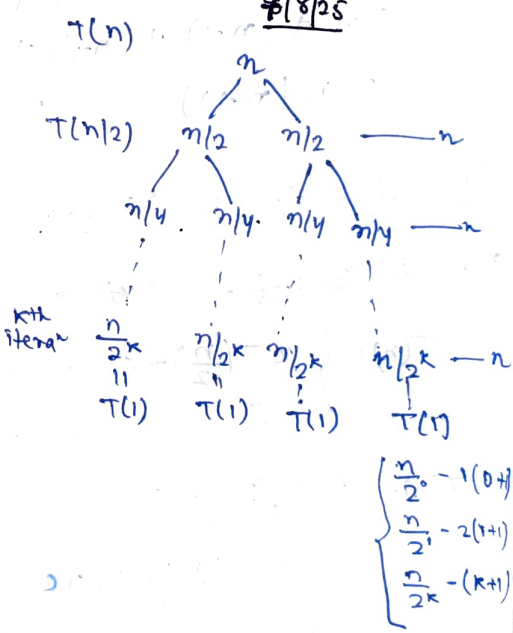
$= O(n)$

Recursion Tree Method

7/8/25

Eg 2

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



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

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

$n = 2^k$

$n \cong 2^k$ $\quad T(n) = (k+1) \cdot n$ $\quad$ (k+1) levels

$k = \log_2 n$ $\quad = n(\log_2 n + 1) = n\log_2 n + n$

$= O(n \log_2 n)$

$\frac{n}{2^0} - 1$ (0th)

$\frac{n}{2^1} - 2$ (1st)

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

* Masters Theorem

If $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function, then the time complexity of a recursive relation is given by

$T(n) = a \cdot T(n/b) + f(n)$

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

$a = 2$, $b = 2$, $f(n) = n$

1. If $f(n) \leq O(n^{\log_b a - \epsilon})$ then $T(n) = O(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} * \log_2 n)$
3. If $f(n) \geq \Omega(n^{\log_b a + \epsilon})$ then $T(n) = \Theta(f(n))$ $\quad \epsilon > 0$, positive constant.

Eg1: $T(n) = 3T(n/2) + n^2$

$a = 3 \quad b = 2 \quad f(n) = n^2$

$f(n) : \ldots \quad n^{\log_b a}$

$n^2 >> n^{\log_2 3}$

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

$\boxed{T(n) = \Theta(n^2)}$

Eg2: $T(n) = 4T(n/2) + n^2$

$f(n) \quad n^{\log_b a}$

$n^2 \quad n^{\log_2 4} = n^2$

$T(n) = \Theta(n^{\log_b a} * \log_b n)$

$\boxed{T(n) = \Theta(n^2 * \log_2 n)}$

Eg3: $T(n) = 2T(n/2) + n\log_2 n$ ($a = 2$, $f(n) = n\log_2 n$)

$f(n) : \ldots \quad n^{\log_b a}$

$n\log_2 n \quad \ldots \quad n^{\log_2 2} = n^1 = n$

$\boxed{T(n) = \Theta(n\log_2 n)}$

Eg4: $T(n) = 2T(n/4) + n^{0.61}$ (a, b, $f(n)$)

$f(n) \quad n^{\log_b a}$

$n^{0.61} \quad n^{\log_4 2} = n^{1/2} = n^{0.5}$

$n^{0.61} > n^{0.5}$

$\boxed{T(n) = \Theta(n^{0.61})}$

Eg5: $T(n) = 16T(n/4) + n!$

$f(n) \quad n^{\log_b a}$

$n! \quad n^{\log_4 16} = n^2$

$\boxed{T(n) = \Theta(n!)}$

Eg6: $T(n) = \sqrt{2}\,T(n/2) + \log_2 n$

$f(n) \quad n^{\log_b a}$

$\log_2 n \quad n^{\log_2 2^{1/2}}$

$= n^{1/2} \cdot \log_2 2$

$= n^{1/2} = \sqrt{n}$

$\log_2 1024$

$\log_2 2^{1024} \quad (2^{1024})^{1/2}$

$= 1024 \quad 2^{512}$

$2^{10} \quad 2^{512}$

$\boxed{T(n) = \Theta(\sqrt{n})}$

Eg7: $T(n) = 2T(n/2) + n$

$f(n) \quad n^{\log_b a}$

$n \quad n^{\log_2 2} = n$

$T(n) = \Theta(n^{\log_b a} * \log_b n)$

$= \Theta(n * \log_2 n)$

$2^n \quad n^2 \quad n = 2^{128}$

$2^{2^{128}} \quad (2^{128})^2$

$\log_2 2^{2^{128}} \quad \log_2 (2^{128})^2$

$2^n \log_2 2 = 2 * \log_2 2^{128} = 2 * 128$

$= 256$

Eg8: $T(n) = 64T(n/8) - n^2\log_2 n$ ($f(n)$ negative)

$64T(n/0.95) + n$ ✗

Masters Theorem

13/8/25

Comparing two exp/values

- $\log_2$ on both sides
- Take a large value of n

$f(n)$	$n^{\log_b a}$
$\sqrt{\log n}$	$\log \log n$
$(\log n)^{1/2}$	$\log_2 \log_2 n$
$\frac{1}{2}\log_2(\log_2 n)$	$\log_2(\log_2 \log_2 n)$

E.g

$f(n)$	$n^{\log_b a}$
$n^{\log n}$	$n \log n$
$\log_2 n \times \log_2 n$	$\log_2(n \log_2 n)$
$\log_2 10^{10} \times \log_2 10^{10}$	$\log_2(10^{10} \times \log_2 10)$
$T(A) = O(n \log n)$	

Take $n = 10^{10}$

E.g

$f(n)$	$n^{\log_b a}$
$n^{\sqrt{n}}$	$n^{\log_2 n}$
$\sqrt{\log_2 n}$	$\log_2 n \times \log_2 n$
$\sqrt{n}$	$\log_2 n$
$\sqrt{2^{10}}$	$\log_2 10$
	10×1

$n^{\sqrt{n}} > n^{\log_2 n}$

Ex- $T(n) = 6T(n/3) + n^2 \log_2 n$ (a = 6, b = 3, $f(n)$)

$f(n)$	$n^{\log_b a}$
$n^2 \log_2 n$	$n^{\log_3 6} = n^{1.7}$

$T(Ex) = O(n^2 \log_2 n)$

Ex9- $T(n) = \sqrt{2}\,T(n/2) + \log_2 n$ (a = $\sqrt{2}$, b = 2, $f(n)$)

$a \geq 1 \quad b > 1$

$f(n)$	$n^{\log_b a}$
$\log_2 n$	$n^{\log_2 \sqrt{2}}$
2^{10} ✓	$(2^{10})^{1/2}$
$\log_2 2^{10}$	$= 2^5$
$= 10$	

$T(n) = O(n^{1/2}) = O(\sqrt{n})$

Ex10- $T(n) = 3T(n/2) + n$ (a = 3, b = 2)

$f(n)$	$n^{\log_b a}$
n	$n^{\log_2 3}$
	$<$

$T(n) = (n^{\log_2 3})$

Ex11- $T(n) = 4T(n/2) + \log_2 n$ (a = 4, b = 2)

$f(n)$	$n^{\log_b a}$
$\log_2 n$	$< n^{\log_2 4} = n^2$

$T(n) = O(n^2)$

Ex12- $2^n\,T(n/2) + n^n$ ($a = 2^n$, b = 2)

$f(n)$	$n^{\log_2 2^n}$
n^n	

Ex13- $T(n) = 2T(n/2) + n\log n$ (a = 2, b = 2, $f(n) = n\log n$)

$f(n)$	$n^{\log_b a}$
$n \log n$	$> n^{\log_2 2} = n$

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

Iterative

Space Complexity

```
A( )
{
  int a,i,j,n;
}
```
S.C = O(1)

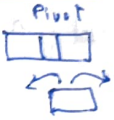
```
A()
{
  int a[10],i,n,j;
}
```
n=10
S.C = O(n)

```
A[ ][ ]
{
  int b,k,c;
}
```
S.C = O(1)

a[][] → O(n²)

1. WAP to implement quick sort using divide & conquer method.
2. WAP " " merge sort " " " " " "

Quick sort



14/8/25

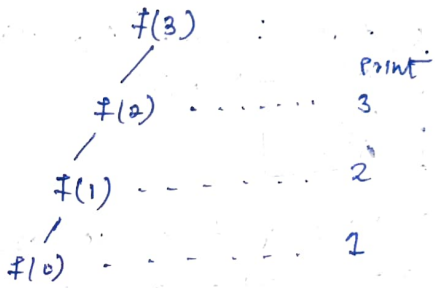
Space Complexity of Recursive Algorithms

Recursion -

```
f(n){
  if(n>=1)
  {
    f(n-1);
    Print(n);
  }
}
```

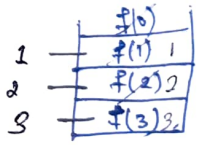
Output -
1, 2, 3

f(3)



∴ f(1) is the last function call
f(0) will return nothing

* use stack to keep track of the pre executed statement just before execution call.



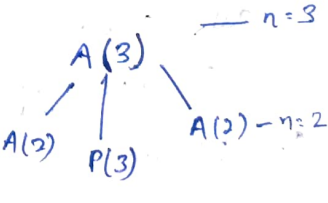
value of n = 3 ⇒ (n+1) ~~*c~~

Total space required for execution → (n+1)*c

S(f(n)) = O(n)

Recursion — Time Complexity

n=3



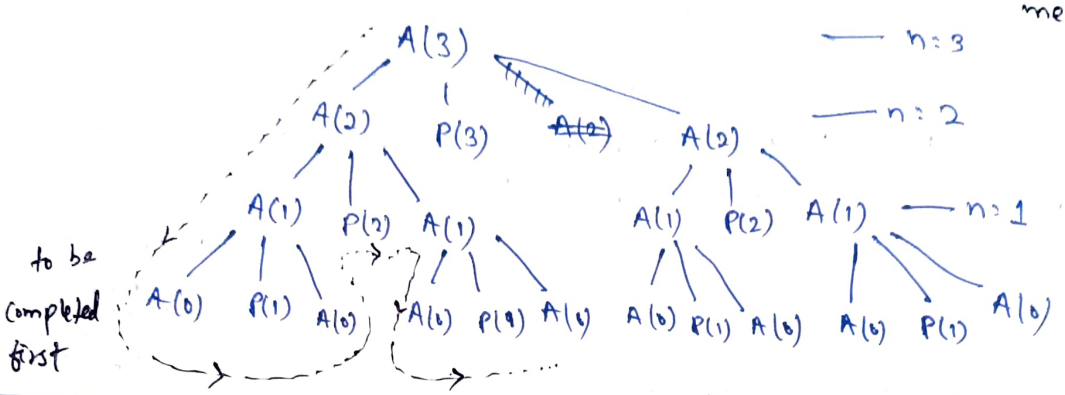
2)
```
A(n){
  if(n>=1)
  {
    A(n-1);   — T(n-1)
    Print(n); — c
    A(n-1);   — T(n-1)
  }
}
```
← T(n)

$$T(n) = 2T(n-1)+c; \quad n \geq 1$$
$$= c \text{ ; otherwise}$$

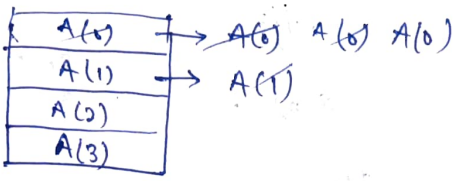
A(2)
all statement assignment with A2 + Further recursive calls

complete then → P(3) → complete then → A(2)

Assignment → Time complexity back substitution & Recursion Tree method.



Print
1
2
1
3
1
2
1



At no point of time, we are exceeding these 4 cells

Total no. of cells required at any point of time = (n+1)

Total space required for execution = (n+1) * c

$S(A(n)) = O(n)$

DIVIDE AND CONQUER

* Dividing a problem to sub problems then sub sub problems till problems become small enough; when solution of all sub problems arrived we merge them to get solution of a bigger complex problem.

1. Divide
2. Conquer
3. Combine

1. Binary Search
2. quick sort
3. Merge sort

Divide and Conquer

1. Binary Search
2. Merge Sort
3. quick Sort
4. Heap Sort

1. Divide
2. Conquer
3. Combine

(*) Binary Search

0	1	2	3	4	5	6
3	5	7	8	9	11	13

low → up

elem = x
to be Selected

$mid = low + \left(\frac{up - low}{2}\right)$

$= 0 + \frac{6-0}{2} = 3$

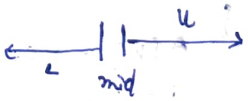
x = 3

a[mid] = 8 false

```
x == a[mid]  // Search successful at position mid.
if(x<a[mid])
   Search the lower half.        { low = low
                                  up = mid - 1 }
else
   Search the upper half
```

```
int BSR(A[], x, l, u)
 {
   if (l == u)
    {
      if (a[l] == x)
        return l; //search successful
      else
        return -1; //search unsuccessful
    }
   else
   {
     mid = l + (u-l)/2;
     if( A[mid] == x)
       return mid; //search successful
     else if (x< A[mid])
     {
       u = mid-1;
       BSR(A, x, l, u);
     }
   else
   {
     l = mid+1;
     BSR(A, x, l, u);
   }
 }
}
```

Base condition smallest during position



x > a[mid]

$T(n) = 1 \cdot T(n/2) + C$

a b f(n)

f(n) $n^{\log_b a}$

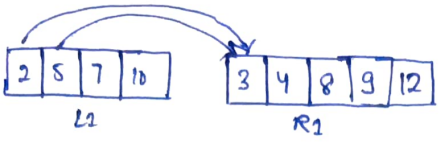
c = 1 $n^{\log_2 1} = n^0 = 1$

$T(n) = O(n^{\log_b a} * \log_b n)$

$= O(1 * \log_2 n)$

$= O(\log_2 n)$

(*) Merge Sort




```
if (ar[i] ≤ ar[j])
    al[k++] = al[i]
        i++
else
    al[i++] = ar[j]
        j++      k=0
```

al

2	3	4	5	7	8	9	10	12

```
Merge (A, P, q, r)
{
  n1 = q - P + 1;
  n2 = r - q;
create L[1, 2, ... n1+1], R[1, 2, ... n2+1]
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) k++
  if (L[i] ≤ R[j])
    {
      A[k] = L[i]
      i = i + 1
    }
  else
    {
      A[k] = R[j]
      j = j + 1
    }
}
}
```

```
Merge Sort (A, P, r)
{
  if (P < r)
  { q = ⌊(P+r)/2⌋
  Merge1s
  Mergesort (A, P, q)   — T(n/2)
  Merge Sort (A, q+1, r) — T(n/2)
  Merge Sort (A, P, q, r)
  }
}
```

O(n)

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

(a, b)

$f(n)$: n ; $n^{\log_b a}$; $n^{\log n_2 2}$

$T(n) = (n^{\log_b a} * \log_b n)$

$= (n * \log_2 n)$

Merge Sort

27/8/25

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

a b f(n)

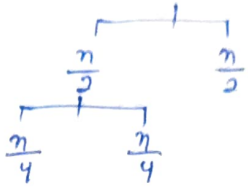
f(n)	$n^{\log_b a}$
n	$n^{\log_2 2} = n^1 = n$

$T(n) = O(n^{\log_b a} \times \log_b n)$

$= O(n \times \log_2 n)$

Space Complexity

$n \rightarrow$ Merge process



Quick Sort

$n\log_2 n$

Pivot element

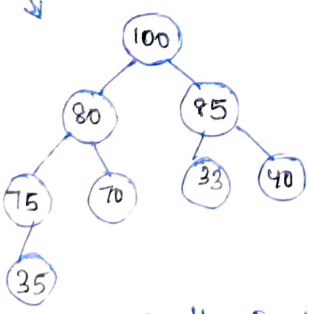
```
Partition (A, p, r)
{
   x = A[r]
   i = p - 1
  for (j = p to r - 1)
  {
    if (A[j] ≤ x)
    {
      i = i+1;
      Swap A[i] with A[j]
    }
  }
  Swap A[i+1] and A[r]
    return i+1
}
```

28/9/25

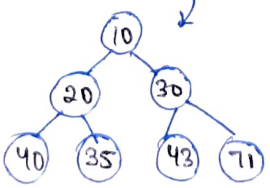
10/9/25

Max heap

Min heap



Min heap



1	2	3	4	5	6	7	8
100	80	85	75	70	33	40	35

1 - index

lc = 2xi

rc = 2xi + 1

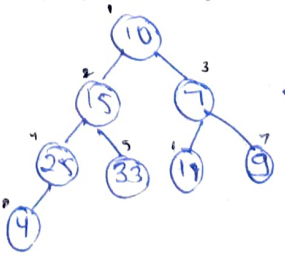
0-index

lc = 2i + 1

rc = 2i + 2

Ex

10, 15, 7, 25, 33, 18, 9, 4



Not maintaining Max heap Property

number → n

Floor $\lfloor \frac{n}{2} \rfloor + 1$ to n → leaves

1 to $\lfloor \frac{n}{2} \rfloor$ → non-leaves

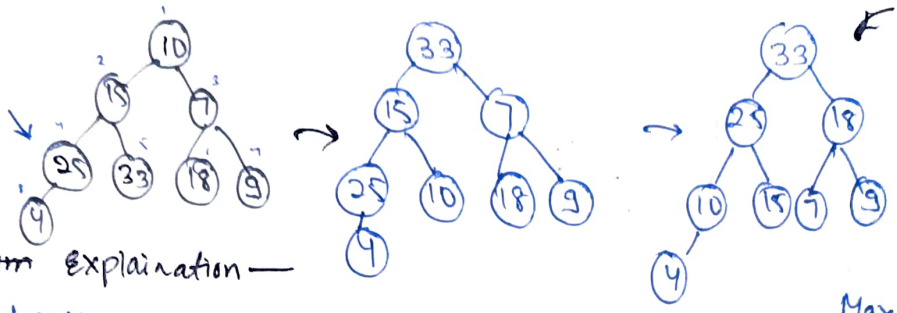
n = 8 $\lfloor \frac{8}{2} \rfloor + 1$ to 8

5 to 8 → leaves

1 to $\lfloor 8/2 \rfloor$ → 1 to 4

Max-Heapify() function

→ largest non-leaf node to root $\lfloor n/2 \rfloor$ → 4



Ans

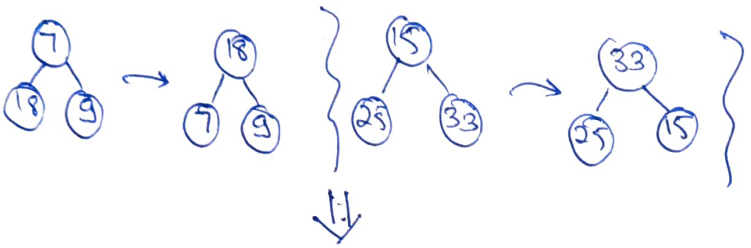
Exam Explaination —

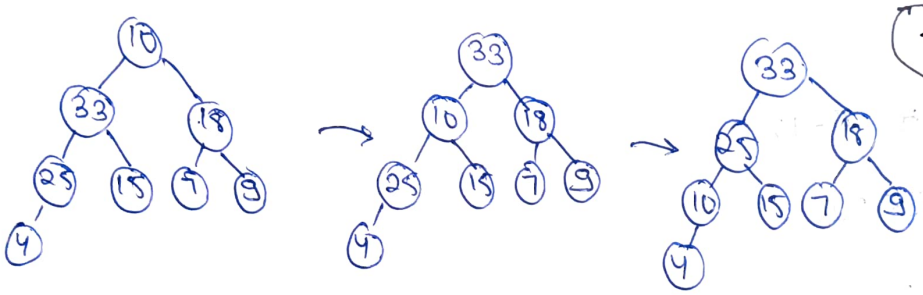
$\lfloor \frac{8}{2} \rfloor = 4$

Max-heapify (A,3)

" " (A,2)

" " (A,1)





Final Max Heap

Heap Algo

(1).

```
Max-Heapify (A,i)
{
  lc = 2*i;
  rc = 2*i+1;
  if ( lc ≤ A.Heapsize && A[lc] > A[i])   // Heapsize → n (Array size)
    largest = lc;
  else
    largest = i;
  if (rc ≤ A.heapsize && A[rc] > A[largest])
    largest = rc;
  if (largest ≠ i)
  exchange A[i] and A[largest]
  Max-Heapify (A, largest)
}
```

left child

1. Max-Heapify (A,i)
2. Build-MaxHeap(A)

```
Build Max-Heap(A)
{
  A.Heapsize = A.length;
  for(i = ⌊A.length/2⌋ to 1)
      Max-Heapify(A,i);
}
```

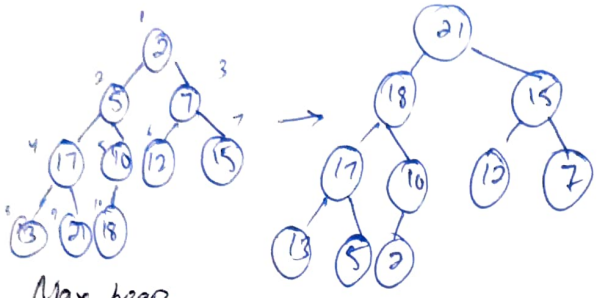
4 std func: max-heapify, build-heap, extract-max, heap-sort

29/10/25

Max heap
Min heap

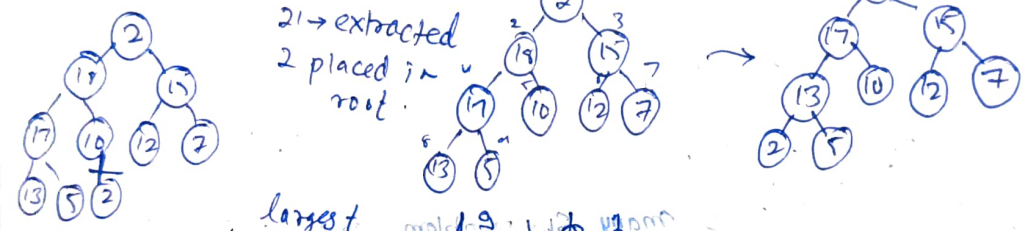
2,5,7,17,10,12,15,13,18

Max-heapify()



Max-heap

1. extract-max()



21 → extracted
2 placed in root.

21 printed

18 printed
5 will place in root.

largest non leaf node = $\lfloor \frac{9}{2} \rfloor$ to 1 = 4 to 1

Print → 21, 18, 17, 15, 13, 12, 10, 7, 5, 2.

Algo –

$\log_2 n$ → height of heap

$2 * \log_2 n$ - Time Complexity of max-heapify $O(\log_2 n)$

(5) build_heap(A){

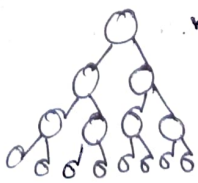
```
A.heap_size = A.length
for(i = ⌊A.length/2⌋ to 1)
    max-heapify(A,i)
}
```

$\lceil \frac{13}{2^{3+1}} \rceil = \lceil \frac{13}{2^{1+1}} \rceil = 4$

$\lceil \frac{13}{2^{2+1}} \rceil = 2$

$\lceil \frac{13}{2^{3+1}} \rceil = 1$

Time Complexity of build_heap



$n=3$
$n=2$
$n=1$
$n=0$

$\lceil \frac{n}{2^{n+1}} \rceil$

$\lceil \frac{13}{2^{1+1}} \rceil \lceil \frac{13}{4} \rceil = 4$

Total amount of work = $\lceil \frac{n}{2^{n+1}} \rceil * O(n)$

$T(BH) = \sum_{n=0}^{\log_2 n} \lceil \frac{n}{2^{n+1}} \rceil * O(n)$

$T(BH) = \sum_{n=0}^{\log_2 n} \lceil \frac{n}{2^{n+1}} \rceil (c \cdot n)$

$= \frac{c \cdot n}{2} \sum_{n=0}^{\log_2 n} \lceil \frac{n}{2^n} \rceil$

$= \frac{c \cdot n}{2} \sum_{n=0}^{\log_2 n} \left\lceil \frac{n}{2^n} \right\rceil$

$\leq \frac{c \cdot n}{2} \sum_{n=0}^{\infty} \frac{n}{2^n}$

$= \frac{c \cdot n}{2} 2$

$: c \cdot n : O(n)$

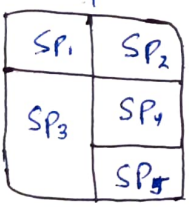
3. extract_max (A){
  if (A.heap_size < 1)
    error "heap underflow"
  max = A[1]
  A[1] = A[A.heap size]
  A.heap_size = A.heapsize - 1
  max_heapify (A, 1)
  return max;

4. Heap Sort(A){
  build_Heap (A)
  for(i = A.length down to 2)
    exchange A[1] with A[i]
    A.heap size = A.heap.size - 1
    max_heapify (A,1) ← root node max heapify
  }
T.C (heap_Sort) : $n * \log_2 n$

30/10/25

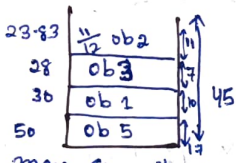
Greedy Algorithm

In Greedy algorithm we will solve many subproblem and to give with our Optimal solution.



Dynamic programming
→ 0/1 knapsack

1. Greedy knapsack (Fractional knapsack) problem → bag



Objects	1	2	3	4	5
weights	10	12	7	14	17
Profit	30	26	28	23	50

max capacity
⇒ C = 45

capacity of knapsack is 45 unit

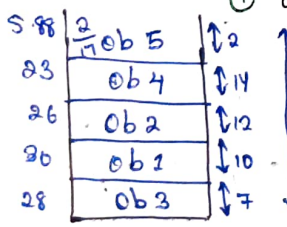
$\frac{11}{12} \times 26 = 23.83$

① Total profit = 131.83

① choose objects with higher profits.

② choose objects with lower weight.

③ choose objects with profit/weight ratio.



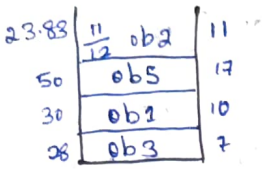
$\frac{2}{17} \times 50 = 5.88$

② Total profit = 112.88

For Greedy Algorithm find Profit/~~cost~~ weight ratio is the best approach among all 3.

obj	1	2	3	4	5
weights	10	12	7	14	17
profits	30	26	28	23	50
P/W	3	2.17	4	1.64	2.94

→ obj 3, obj 1, obj 5, obj 2, obj 4
Sorting by descending order P/W



Total profit = 131.88 ← we get the highest profit in the 3rd approach among all 3 approaches.

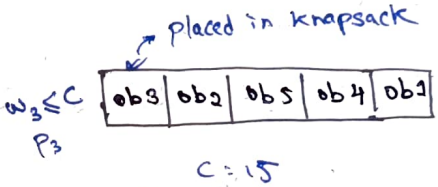
③

Algorithm

```
Greedy knapsack (){
    c=m, p=0;
    for i=1 to n
        compute Pi/Wi
```

n → no. of objects
m → c → Max capacity of the knapsack.
Store temporarily
P_i → profit of item i
W_i → weight of item i
p → total profit

```
    Sort objects in descending order of Pi/Wi
    for i=1 to n from sorted list {
        if (c>0 && Wi ≤ c)
            c = c - Wi
            p = p + Pi
        else
            break;
    }
    if (c>0){
        p = p + Pi * c/Wi
    }
}
```



c = 15
obj w_5 = 20

$3 * \frac{15}{20}$

Huffman Coding

1. Uniform length Coding
2. Variable length Coding

10,000
a, b, c, d, e → 5

uniform L.C
a → 000
b → 001
c → 010

~~a→0, c→10, e→11~~.
↖ No. of bit

5000 × 1
3000 × 2
600 × 3
850 × 2
550 × 3
16,150 ← variable

a → 0
c → 10
e → 11
b → 110
d → 111

5000 → a
3000 → c
600 → b
850 → e
550 → d

variable L.C
a → 0
b → 10
c → 111

ASCII → 8 × 10,000
= 80,000

uniform → 3 × 10,000 = 30,000
No. of bits using uniform length coding

variable No. of bits using variable length coding

Ex- n = 100 (100 characters)
a = 50
b = 40
c = 5
d = 5

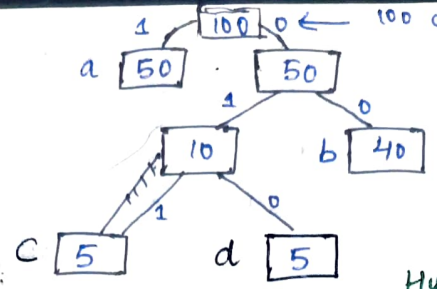
* we are transmitting 100 characters

uniform
↳ Minimum coding = 2 * 100 = 200

c = d = 5
b = 40
a = 50
↑ Sorted in descending

n=100

a=50	c: 5
b: 40	d: 5
c: 5	b: 40
d: 5	a: 50



Assumption
{ l → 1
r → 0

Huffman Tree

1*50 + 2*40 + 3*5 + 3*5
= 160

Huffman Codes - we have to traverse the tree root to leave node.

a → 1
b → 00
c → 011
d → 010

Huffman Codes

6|11|25

Greedy Algorithm

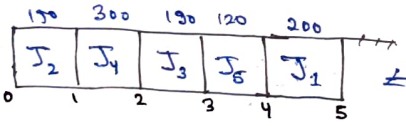
1. Greedy knapsack ✓
2. Huffman Coding ✓
3. Job sequencing with deadlines ↓

Jobs / Deadlines

	J_1	J_2	J_3	J_4	J_5	J_6
D:	5	3	3	2	4	2
P:	200	150	190	300	120	100

Job1 should be completed in 5 hrs like that

* Select the job in such a way so you can make maximum profit.



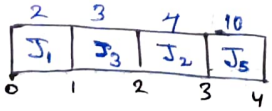
← Gang chart

Select job will maximum profit → J_4

Total Profit = 150+300+190+120+200 = 960 unit

Ex2.

	J_1	J_2	J_3	J_4	J_5
P:	2	4	3	1	10
D:	3	3	3	4	4



Maximum profit → J_5
Dealine → 4 hrs

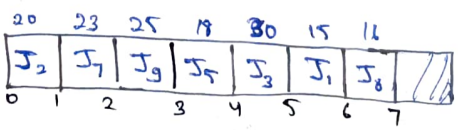
* If slot is full search for other best suited

Total profit = 2+3+4+10
= 19 unit

Ex3.

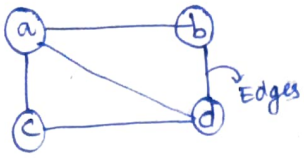
Discarded because deadline

	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
P:	15	20	30	18	18	10	23	16	25
D:	7	2	5	3	4	5	2	7	3



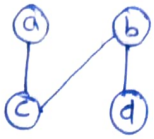
Total profit = 20+23+25+18+30+15+16
= 147 unit

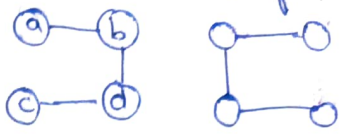
4. Minimum Cost Spanning tree



4 Nodes
4 vertices

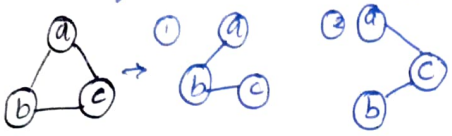
Spanning tree is a tree which is constructed graph containing all the node with minimum no. of edges.





n vertices/nodes

Spanning tree : n-1 edges



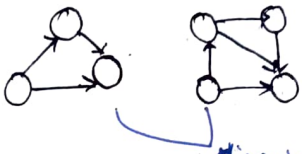
n^{n-2}

n : vertices

For n = 3 : $3^{3-2} = 3^1 = 3$

For n = 4 : $4^{4-2} = 4^2 = 16$

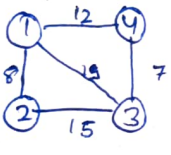
* Directed graph
Undirected graph



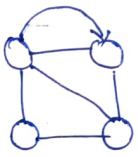
directed graph → which have directions.

① Un-weighted Graph → No weight assigned to edges

② weighted graph



weighted Graph



→ possible

graph will n nodes means n-1 edges

n^{n-2} spanning trees

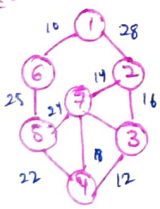
Minimum Cost Spanning Tree

2 Algorithms —

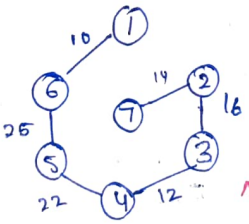
(a) Prim's Algorithm

(b) Kruskal Algorithm

Ex.



using algorithm prism's



MCST

Total cost : 10+25+22+12+16+14

= 99

① 10 is lowest edge

② find the neighbours of 1 and 6

1 → 6, 2 ↑ taken

6 → 1, 5 ↑ ✓

① — 28 — ②

⑤ — 25 — ⑤ ✓

5 → ~~6~~, 7, 4

4 → ~~5~~, 7, 3

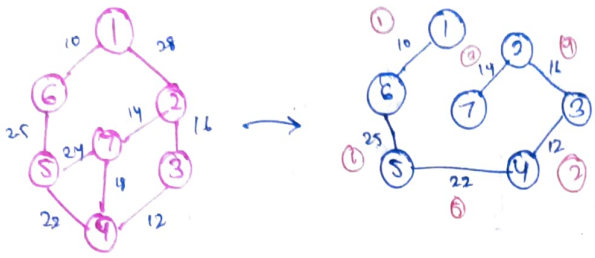
3 → 4, 2

2 → 1, 7, 3 ↑

got a cycle where all nodes is taken!!

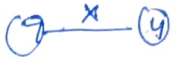
Km → Complete graph

Using Kruskal Algorithm–



$n = 7$ nodes

$n-1 = 6$ edges

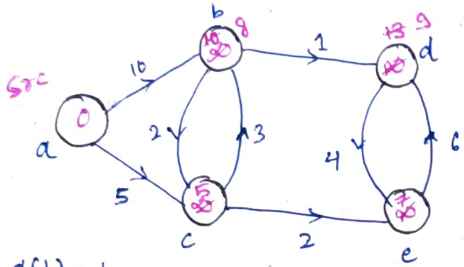


we are getting complete cycle so don't do it.

12/11/25

Dijkstra Algorithm

Single Source shortest path Algorithm



Relax (u, v, c)
If $d(u) > d(v) + c(u,v)$ (cost)
Then
$d(u) = d(v) + c(u,v)$

Relaxing a node

$d(b) = \infty$

$d(a) + c(a,b) = 0 + 10$

$d(b) > d(a) + c(a,b)$ ↑ True

So, $d(b) = d(a) + c(a,b)$

$u = b$, $v = a$ $\Rightarrow d(b) = 10$

$d(c) = \infty$

$d(a) + c(a,c) = 0 + 5$

$d(c) > d(a) + c(a,c)$

So, $d(c) = d(a) + c(a,c)$

$d(c) = 5$

∴ Src a & src c (node a, node c) has been taken so, node has ~~wat~~ whatever ~~th~~ edges that will be fully ignored.

∴ Next ~~closed~~ closest distance node ~~Src~~ = c

Consider only outgoing edge

$d(b) = 10$,

$d(c) + c(c,b) = 5 + 3 = 8$

$d(b) > d(c) + c(c,b)$

So, $d(b) = 8$

$d(e) = \infty$

$d(c) + c(c,e) = 5 + 2 = 7$

$d(e) > d(c) + c(c,e)$

So, $d(e) = 7$

~~Src~~ node = e

$d(d) = \infty$

$d(e) + d(e,d) = 7 + 6 = 13$

$d(d) > d(e) + d(e,d)$

So, $d(d) = 13$

∴ node b outgoing edge is node d ignored.

$d(d) = 13$

$d(b) + c(b,d)$, $8 + 1 = 9$ ~~5+3=8~~

$d(d) > d(b) + c(b,d)$

$d(d) =$ ~~13~~ 9

$d(a,b) = 8$
$d(a,c) = 5$
$d(a,d) = 9$
$d(a,e) = 7$

↓ distance

Algorithm Using Min Heap

Dijkstra ~~Algorithm~~ (G, w, s) — Graph, weight matrix, Source node

{

1. Initialize Single Source (G, s) // all node initialize to ∞
2. $S = \phi$ ← set
3. $Q = G.V$ // Build min heap
4. while ($Q \neq \phi$)
   - u = Extract Min(Q) ← Src node i.e a
   - $S = S \cup \{u\}$
   - For each vertex $v \in G.Adj[u]$
     - Relax (u, v, w) // decrease key operation in min heap ↑ $\log(V)$

13/11/25

Matrix Multiplication (From Divide and Conquer)

$C_{m \times p} = A_{m \times n} \times B_{n \times p}$

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = C \begin{bmatrix} c_{11} = a_{11}*b_{11} + a_{12}*b_{21} & c_{12} = a_{11}*b_{12} + a_{12}*b_{22} \\ c_{21} = a_{21}*b_{11} + a_{22}*b_{21} & c_{22} = a_{21}*b_{12} + a_{22}*b_{22} \end{bmatrix}$$

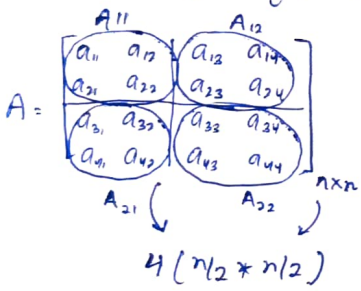
No. of Scaler multiplication = $m * n * p$

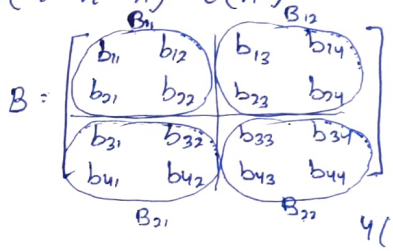
Hence it is $2*2*2 = 8$

```
for(i=0; i<m; i++){
    for(j=0; j<p; j++){
        c[i][j] = 0;
        for(k=0; k<n; k++){
            c[i][j] += a[i][k] * b[k][j]
        }
    }
}
```

Time Complexity

For 3 nested for loop = $O(n*n*n) = O(n^3)$





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

Algo:-

```
MM(A, B, n)
{
  if(n<=2){
    c11 = a11 * b11 + a12 * b21
    c12 = a11 * b21 + a12 * b22
    c21 = a21 * b11 + a22 * b21
    c22 = a21 * b12 + a22 * b22
  }else{
    mid = n/2
    MM(A11, B11, n/2) + MM(A12, B21, n/2)
    MM(A11, B12, n/2) + MM(A12, B22, n/2)
    MM(A21, B11, n/2) + MM(A22, B21, n/2)
    MM(A21, B12, n/2) + MM(A22, B22, n/2)
  }
}
```

Time Complexity —

$$T(n) = \begin{cases} 1, & n \le 2 \\ 8T(n/2) + n^2, & \text{otherwise} \end{cases}$$

Apply Master's Theorem

~~a(T)~~ $aT(n/b) + f(n)$

$a = 8;\ b = 2;\ f(n)$

$\Rightarrow n^{\log_b a} + n^2$

$n^{\log_2 8} + n^2 \Rightarrow n^3 + n^2$

Then, $T_C = O(n^3)$

(Even if after applying Divide & conquer approach)

Strassen's Matrix Multiplication —

1. $P = (A_{11} + A_{12})\ (B_{11} + B_{22})$
2. $Q = (A_{21} + A_{22}) \cdot B_{11}$
3. $R = A_{11} \cdot (B_{12} - B_{22})$
4. $S = A_{22} \cdot (B_{21} - B_{11})$
5. $T = (A_{11} + A_{12}) \cdot B_{22}$
6. $U = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$
7. $V = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$

Resultant Matrix elements:

$C_{11} = P + S - T + V$

$C_{12} = R + T$

$C_{21} = Q + S$

$C_{22} = P + R - Q + U$

TC $\Rightarrow$

$$T(n) = \begin{cases} 1, & n \le 2 \\ 7\,T(n/2) + k \cdot n^2 & \end{cases}$$ (masters Theorem)

$a\,T(n/b) + f(n)$ | $\Rightarrow n^{\log_b a} + kn^2$

$a = 7,\ b = 2$ | $\Rightarrow n^{\log_2 7} + n^2$

$\Rightarrow n^{2.81}$

$T.C = O(n^{2.81})$

Dynamic Programming

Matrix chain Multiplication

$A_{6\times 50} \times B_{50\times 200} \times C_{200\times 300}$

Dynamic Programming

19/11/25

Matrix chain Multiplication

D = A B C

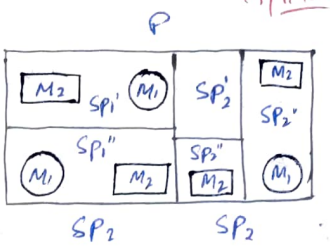
1. $(A \times B) \times C = D \rightarrow S_1 + S_2 = 60,000$
2. $A \times (B \times C) = D \rightarrow S_1 + S_2 = 1,20,000$

$A_{10 \times 20}$ $B_{20 \times 50}$ $C_{50 \times 100}$

$S_1 = 20 \times 50 \times 100 = 1,00,000$

$A_{10 \times 20}$ $B = [20 \times 100]$

$S_2 = 10 \times 20 \times 100 = 20,000$



$A_{10 \times 20}$ $B_{20 \times 50}$ $C_{50 \times 100}$

$S_1 = 10 \times 20 \times 50 = 10,000$

$A_1' = 10 \times 50$ $C_{50 \times 100}$

$S_2 = 10 \times 50 \times 100 = 50,000$

1. (A B) (C D)
2. ((A (BC)) D)
3. (((A B) C) D)
4. (A (B (C D)))
5. (A ((B C) D))

$A_{10 \times 100}$ $B_{100 \times 30}$ $C_{30 \times 5}$ $D_{5 \times 80}$

(A)' B C D

1. (A) ((B C) D)
2. (A) (B (C D))

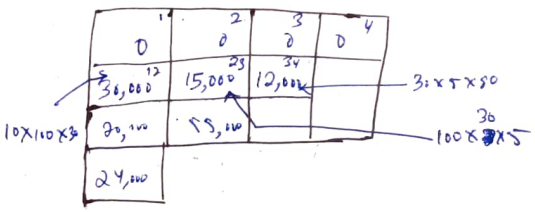
A B' C D

3. (AB) (CD)

A B C' D

4. ((AB) C) D
5. (A (BC)) D

1. A ((BC) D)
2. A (B (C D))
3. (AB) (CD)
4. ((AB) C) D
5. (A (BC)) D



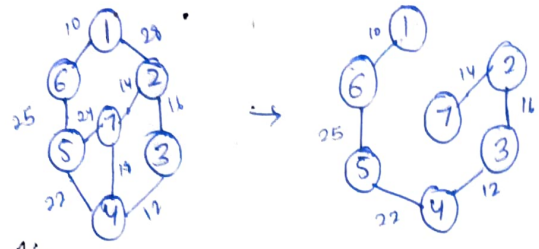
$m_{123} = \begin{cases} m_{12} \times m_3 = 30,000 \\ m_1 \times m_{23} \end{cases}$ ~~$10 \times 3 \times 5 = 31$~~

$m_{123} = \min \begin{cases} m_{12} \times m_3 = 30,000 + 10 \times 30 \times 5 \\ \quad = 30,000 + 1500 = 31000 \\ m_1 \times m_{23} = 10 \times 100 \times 5 + 15,000 \\ \quad = 5000 + 15000 = 20,000 \end{cases}$

$m_{234} = \min \begin{cases} m_{23} \times m_4 = 15000 + 100 \times 5 \times 80 = 15,000 + 40,000 = 55,000 \\ m_2 \times m_{34} = 100 \times 30 \times 80 + 12,000 = 2,40,000 + 12,000 = 2,51,000 \end{cases}$

$m_{1234} = \min \begin{cases} m_1 \times m_{234} = 10 \times 100 \times 80 + 55,000 = 1,35,000 \\ m_{12} \times m_{34} = 30,000 + 12,000 + 10 \times 30 \times 80 = 66,000 \\ m_{23} \times m_4 = 20,000 + 10 \times 5 \times 80 = 24,000 \end{cases}$

Prism's Algorithm



Algo-

```
import numpy as np
INFINITY = 99999
G = [[ 0, 28, 0, 0, 0, 10, 0],
     [28, 0, 16, 0, 0, 0, 14],
     [0, 16, 0, 12, 0, 0, 0],
     [0, 0, 12, 0, 22, 0, 18],
     [0, 0, 0, 22, 0, 25, 24],
     [10, 0, 0, 0, 25, 0, 0],
     [0, 14, 0, 18, 24, 0, 0 ]]
visited = [0, 0, 0, 0, 0, 0, 0]
no_edges = 0
visited[0] = True
min_cost = 0
N = 7
while no_edges < N - 1:
    min = INFINITY
    u = 0
    v = 0
    for i in range(N):
        if visited[i]:
            for j in range(N):
                if (not visited[j] and G[i][j]
                    if min > G[i][j]
                        min = G[i][j]
                        u = i
                        v = j
    Print(str(u+1)+ "->" +str(v+1)+ ":"
                        str(G[u][v]
    visited[v] = True
    min_cost = min_cost + G[u][v]
    no_edges = no_edges + 1
Print("Min cost = ", min_cost)
```

Largest Common Sequence (LCS)

20/11/25

Subsequence → character's extracting from a string in ascending order

String → It is a sequence of characters taken in consequtive order

University
1 2 3 4 5 6 7 8 9 10

Univ
1 2 3 4

Vers
4 5 6 7

X = AAB Y = ACA

		X	A	A	B
	V	0	0	0	0
A	0	1	1	1
C	0	1	1	1
A	0	1	2	2

if $x_i = y_i$ then prev+1

if $x_i \neq y_i$ then Max

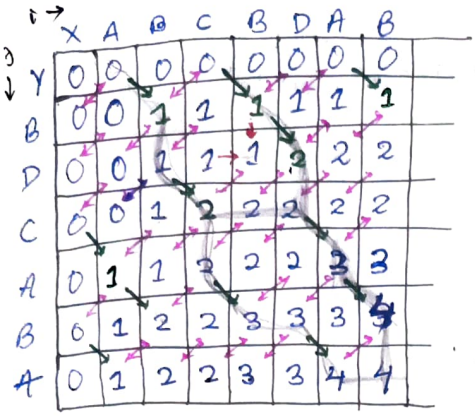
Length of the longest common subsequence = 2

AA

X = A B C B D A B |X| = 7

Y = B D C A B A |Y| = 6

∴ 1'st col & 1'st Row all will be in Zero



if $x_i = y_i$, diagonal +1

if $x_i \neq y_i$, compare x_i & y_i

|LCS| = 4

1. B D A B
2. B C A B
3. B C B A

∴ No. of longest common Subsequence = 3

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

→ diagonal element

LCS Algorithm :-

```
LCS(X, Y) {
m = x.length
n = y.length
let c[0...n, 0...m] be a new table
for j = 0 to n
    c[j, 0] = 0
for i = 0 to m
    c[0, i] = 0
for j = 1 to n
  for i = 1 to m
    if xi == yj
      c[i, j] = 1 + c[i-1, j-1]
    else
      c[i, j] = max(c[i-1, j], c[i, j-1])
}
```

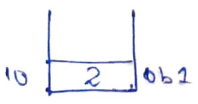
26/11/25

0/1 Knapsack

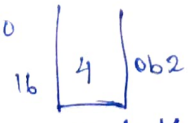
Fractional Knapsack ≠ Greedy Knapsack

	ob1	ob2
W	2	4
P	10	16
P/W	5	4

C = 4



Total = 10



Total = 16

Greedy knapsack ✗

C = W

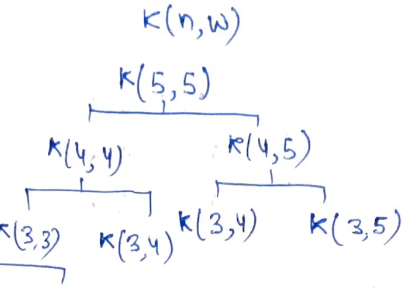
$K(i,w) = \begin{cases} \max(P_i + K(i-1, w-w_i),\ K(i-1,w)) & \\ 0 : \ i=0 \text{ or } w=0 & \\ K(i-1,w); \ w < w_i & \end{cases}$

$i \rightarrow w_i$, P_i



$O(2^n)$





P \ W	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	15	15	15	15	15	15
2	0	15	20	35	35	35	35
3	0	15	20	35	35	45	50

$K(2,2) = \max(20 + K(1,0),\ K^{15}(1,2))$

$K(2,3) = \max(20 + K^{20+15}(1,1),\ K^{15}(1,3))$

$K(2,4) = \max(20 + K^{20+15}(1,2),\ K^{15}(1,4))$

$K(3,1) = K(2,1)$

$w_3 = 4$, $w_3 = 1$

$K(3,2) = K(2,2)$

$K(3,3) = K(2,3)$

$K(3,4) = \max(30 + K^{30+0}(2,0),\ K^{35}(2,4))$

$K(3,3) = K(2,3) =$

$K(3,5) = \max(30 + K^{30+15}(2,1),\ K^{35}(2,5))$

$K(3,6) = \max(30 + K(2,2),\ K(2,6))$

$30+20,\ 35$

~~Splo~~ Space Complexity

27/11/25

Iterative Algorithms

```
A1()
{
  int i, n;
  i = 10;
  n = i*i + 3*i;
  Pf(n);
}
```

4 Bytes
↓
c
2*c

$S(A1) = O(c) = O(1)$

```
A2()
{
  int i, n, a[100];
  Stm 1;
  Stm 2;
  Stm n;
}
```

2*c
n*c

$S(A_2) = 2*c + n*c$
$= O(n)$

A[n]
n = 100

```
A3()
{
  int i, n = 1000;
  int a[n], b[n][n];
  Stm 1;
  Stm n;
}
```

$S(A_3) = 2*c + n*c + n + n*c$
$= O(n^2)$

```
A4 (a[], n)
{
  int i, j;
  char c1, c2;
  stmt 1;
  stmt n;
}
```

S(A4) = 2*c + 2*n = O(1)

```
A5 ( a[], n)
  int i, j, m[n][n];
  char c1, c2;
  stmt 1,
  stmt n,
}
```

S(A5): 2*c + 2*k + n*n*c, = O(n²)

```
A6 (n){
    int s = 0;
     if (n==1)
      return 1;
     else
      return (n * A6(n-1))
  }
```

n = 4

A6(4)
A6(3)
A6(2)

n

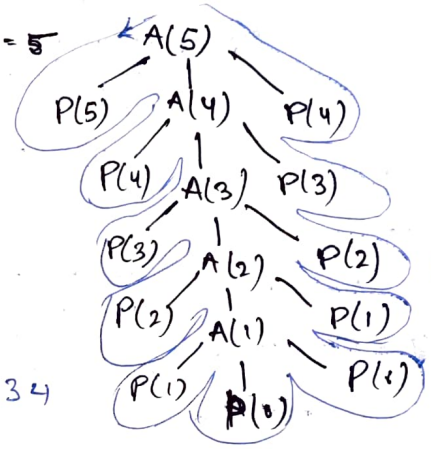


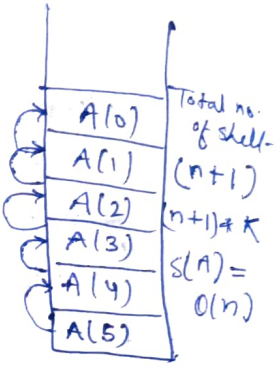
Recursive Algorithm

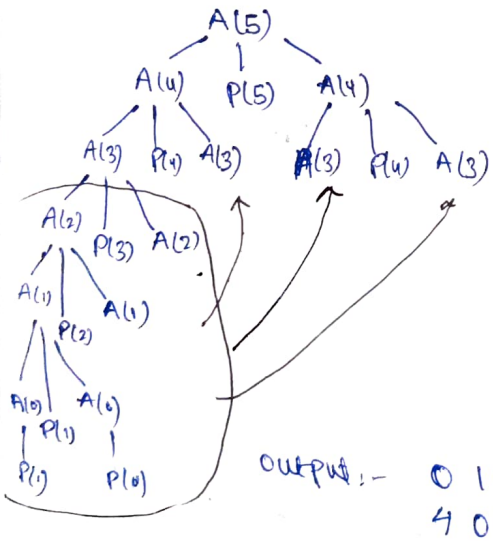
```
A(n){
 if (n==0)
   P(0)
 else
   P(n)
   A(n-1)
   P(n-1)
}
```

→ 5 4 3 2 1 0 0 1 2 3 4

n = 5



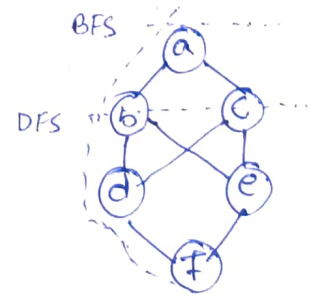




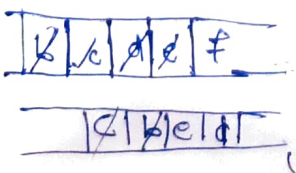
Output:- 0 1 0 2 0 1 0 3 0 1 0 2 0 1 0
4 0 1 0 2 0 1 0 3 0 1 0 2 0 1 0
5

Graph Traversal Algorithm

(1) Breadth First Search (BFS) - Queue
(11) Depth First Search (DFS) - Stack



(i) a b c d e f



Visited	explored
a - true	false
false	false
true	true
true	

(ii) a c b e d f