

22/08/2027

### Algorithms :

#### ① Asymptotic Analysis

Recursive | Non recursive Algo

#### ② Asymptotic Notations

#### ③ Recurrence Relations [Methods to solve]

#### ④ Algorithm Design Techniques

Divide & conquer  
Greedy  
Dynamic

#### ⑤ Tree and graph representation & traversal

Priority Queues

#### ⑥ Sorting algo.

Books → Cormen

→ Sartaj Sahani

→ Mark Allen Weiss

ravi.82 peddappa@gmail.com

@facebook.com

08074172708

### # Introduction:

→ Step by step representation of computer program.

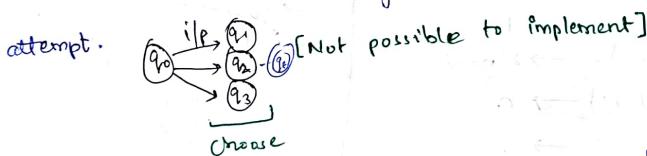
#### Criterias:

① Finiteness - Algo must terminate in finite amount of time.

② Definiteness - Each step of algorithm must have unique solution.

[Deterministic Algorithm],  $q_{10} \xrightarrow{\text{if } p} q_{11}$  [possible to implement in comp.]

Non Deterministic Algo: each step of algo consists of finite no. of solutions and algo should choose correct solution in first attempt.



③ Effectiveness - Each step of algo should be very basic.

### # Steps to design algorithm:

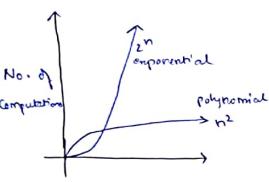
① Devise Algo: Design algo for given problem using best design technique.

② Validation of Algo: Test algo correctness.

③ Analysis of algo: Estimation of CPU execution time | main mem space reqd.  
to complete execution of algo.

Decidable Problem: Problem which can be solved in polynomial time using deterministic algo. [efficient algo.]  
Irreducible: No. of computations required  $\rightarrow n^2, n \log n, n^3, nk$  [polynomial time]  
 No. of computations required  $\rightarrow n^2, n \log n, n^3, nk$  [polynomial time]  
 No. of computations required  $\rightarrow n^2, n \log n, n^3, nk$  [polynomial time]

Undecidable Problem: Problems which take exponential time to solve  
Irreducible: using deterministic algo.  
 n/p size: # of computations  $\rightarrow 2^n, 3^n, n!, n^n$  [Exponential time]



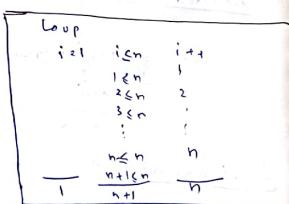
n	$n^2$ polynomial	$2^n$ exponential
10	100	1024
11	121	2048
12	144	4096
13	$169 = 13^2$	$2^{13}$ (halt)
50	$(50)^2$	$2^{50}$ (May not halt)

#### # Algorithm Analysis [Estimation of Time and space complexity]

① Time Complexity: Estimation of CPU time [CPU computations] required to terminate algo.

TC of algo = sum of frequency count of each instruction of algo.

Algo: sum(a,n)  
 $\{$  sum=0; ...  
 $\{$  for(i=1; i<=n; i++)  $\rightarrow 1$   
 $\{$  sum = sum + a[i]; }  $\rightarrow n$   
 $\{$  return sum;  $\rightarrow 1$   
 $\}$   
 $T(n) = 3n + 4$   
 $T(n) = \Theta(n)$



② Space Complexity: Estimation of main memory space required to execute algo.

sum(a,n)  $\Rightarrow$  a[] :  $\frac{n}{2} + 1$   
 $\{$  sum := 0;  
 $\{$  for(i=1; i<=n; i++)  
 $\{$  sum := sum + a[i]; }  
 $\{$  return sum;  $\rightarrow c(n)$   
 $\}$   
 $a[] : \frac{n}{2} + 1 \rightarrow c(n) = \text{space complexity } (S(n) = \Theta(n))$

- # TC [SC of loops]:
- ① for ( $i=1; i \leq n; i++$ )  $\rightarrow [n+1]$   
 $\{$  printf("Made Easy");  $\rightarrow \frac{n}{3n+2} = O(n)$   
 $\boxed{TC = \Theta(n)}$   
 $\boxed{SC = 2 \text{ (constant)} = O(1)}$
  - ② for ( $i=n; i \geq 1; i--$ )  
 $\{$  printf("Made Easy");  $\rightarrow n \text{ Times}$   $\boxed{TC = \Theta(n)}$   
 $\boxed{SC = \Theta(1)}$
  - ③ for ( $i=1; i \leq n; i=i+5$ )  
 $\{$  printf("Made Easy");  $\rightarrow \lceil \frac{n-1}{5} \rceil + 1 \text{ times}$   
 $\underbrace{i=1, 1+5, 1+2 \times 5, 1+3 \times 5, \dots, 1+k \times 5}_{k+1 \text{ prints}}$   
 $k+1 \leq n$   
 $k \leq \frac{n-1}{5} \Rightarrow k = \lfloor \frac{n-1}{5} \rfloor$   $\boxed{TC = \Theta(n)}$   
 $\boxed{SC = \Theta(1)}$
  - ④ for ( $i=n; i \geq 1; i = i-5$ )  
 $\{$  printf("Made Easy");  $\rightarrow \lceil \frac{n-1}{5} \rceil + 1 \text{ times}$   $\boxed{TC = \Theta(n)}$   
 $\boxed{SC = \Theta(1)}$
  - ⑤ for ( $i=1; i \leq n; i=i \times 2$ )  
 $\{$  printf("Made Easy");  $\rightarrow \lceil \log_2 n \rceil + 1 \text{ times}$   
 $\underbrace{i=1, 2, 2^2, 2^3, \dots, 2^k}_{k+1 \text{ times}} \leq n$   
 $2^k \leq n$   
 $k \log_2 \leq \log n$   
 $k \leq \log n \Rightarrow k = \lfloor \log_2 n \rfloor$   $\boxed{TC = \Theta(\log n)}$   
 $\boxed{SC = \Theta(1)}$
  - ⑥ for ( $i=n; i \geq 1; i = i/2$ )  
 $\{$  printf("Made Easy");  $\rightarrow \lceil \log_2 n \rceil + 1 \text{ times}$   $\boxed{TC = \Theta(\log n)}$   
 $\underbrace{i=n, \frac{n}{2}, \frac{n}{2^2}, \frac{n}{2^3}, \dots, \frac{n}{2^k}}_{k+1 \text{ times}} \geq 1$   
 $k+1 \geq \log_2 n \Rightarrow k = \lfloor \log_2 n \rfloor$   
 $\boxed{SC = \Theta(1)}$

⑦ for ( $i=1$ ;  $i \leq n$ ;  $i = i + 5$ )  
 $\quad \text{printf}(\text{" Made Easy"});$   $\lceil \log_5 n \rceil + 1$  times.

$$TC = \Theta(\log_5 n)$$

⑧ `for(i=2; i<n; i*=2)`  
`printf("Made Easy");` →  $\lfloor \log_2(\log n) \rfloor + 1$  times

$$TC = \log_2(\log_2 n)$$
$$SC = O(1)$$

$$\Rightarrow 2, 2^2, 2^4, 2^8, \dots$$

$$\underbrace{2, 2^{(k)}, 2^{(k^2)}, 2^{(k^3)}, \dots, 2^{(k^{\lfloor k \rfloor})}}_{k+1 \text{ times}} \leq n$$

$$\sum_{i=1}^{(k)} 2^{(k^i)} \leq n$$

$$2^{\lfloor k \rfloor} \log_2 2 \leq \log_2 n$$

$$2^k \leq \log_2 n$$

$$k \log_2 2 \leq \log_2 (\log_2 n) \Rightarrow k = \lfloor \log_2 (\log_2 n) \rfloor$$

⑨ for ( $i = n; i \geq 2; i = \lceil \frac{i}{2} \rceil$ )  
 $\text{printf}(\text{"Mode Easy"}, i); \rightarrow \lceil \log_2(\log_2 n) \rceil + 1 \text{ times}$

$$TC = \log_2(\log_2 n)$$

$$SC = \Theta(1)$$

\* for( $i=1$ ;  $i \leq n$ ;  $i = i + c$ )  
 {  
 (or)  
 for( $i=n$ ;  $i \geq 1$ ;  $i = i - c$ ) }  $Tc = \Theta(n)$

\*  $\text{for}(i=1; i \leq n; i = i + c)$   $T_C = O(\log_c(n))$   
 $\quad \quad \quad \text{for}(i=n; i \geq 1; i = i / c)$

\*  $\text{for } (i = \lceil \frac{n}{\sqrt{e}} \rceil; i^2 \leq n; i^2)$   $Tc = \Theta(\log_e \log_n n)$   
(or)

⑩ for ( $i=2$ ;  $i \leq 2^n$ ;  $i = i^2$ )  
 $\quad \text{printf}(\text{"Made [easy]"); \rightarrow [\log_2 n] + 1$   
 $\Rightarrow 2, 2^2, 2^4, \dots, 2^k \leq 2^n$   
 $2^{2^k} \leq 2^n$   
 $2^{k \log_2 2} \leq n \log_2 2$   
 ~~$k \leq \log_2 n$~~   $\Rightarrow k = \lceil \log_2 n \rceil$

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

```
(11) for (i = 1; i < n; i = i * 2)
    printf ("Made [ay]");
```

(12) for ( $i=1; i \leq 2^n; i=i*2$ )  
     printf ("Made Easy");  $TC = O(\log_2(2^n))$   
 $TC = O(n)$

→ # Nested Loops :

① Independent nested loop: Inner loop variable not depends on outer loop variable.

$\text{① } \text{Ex. } \left[ \begin{array}{l} \text{for } (i=1; i \leq n^2; i=i+10) \\ \quad \left[ \begin{array}{l} \text{for } (j=n; j \geq n_2; j=j/2) \\ \quad \left[ \begin{array}{l} \text{for } (k=n^2, k \geq n, k=k-10) \\ \quad \quad \text{print } f(\text{"mode easy"}); \end{array} \right] \\ \quad \end{array} \right] \\ \end{array} \right]$

$Tc = \Theta(n^2 \times (\log n - 1) \times n)$   
 $= n^2 \times 1 \times n^2$   
 $= \Theta(n^4)$

$K > n^2, n^2 - 10, n^2 - 2 \times 10, n^2 - 1 \times 10 > n$   
 $n^2 - k \times 10 \geq n$   
 $n^2$



$\text{for } (i=1; i \leq n; i++)$   
 for ( $j=1; j \leq n; j=j+i$ )  
 $x = x+1;$

$i \quad | \quad 1 \quad 2 \quad 3 \quad \dots \quad n$   
 $j \quad | \quad 1 \text{ to } n \quad \text{times} \quad 2 \text{ to } \frac{n}{2} \text{ times} \quad \dots \quad 1 \text{ time}$   
 $x = x+1 \quad | \quad n + \frac{n}{2} + \frac{n}{3} + \dots + \frac{n}{n}$

What is TC of loop?

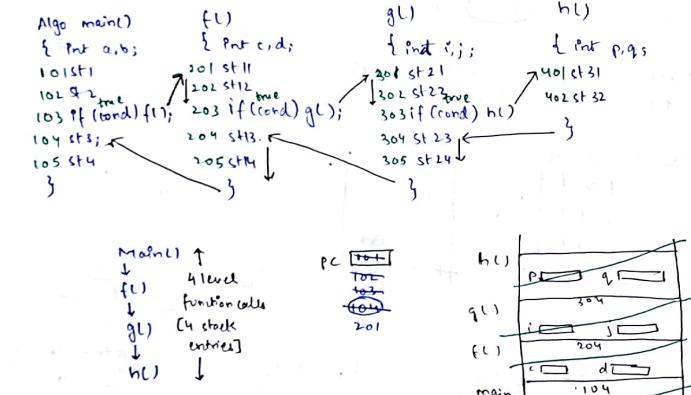
$1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} = \log n$   
 $\sum_{x=1}^n \frac{1}{x} dx \approx \log n$

$= n \left[ 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} \right]$   
 $= \Theta(n \log n)$

23/08/2017:

## Recursive Algorithm:

→ Stack DS required to run recursive algo and subroutine calls to store return address.



\* overheat of function call

- ~~① push [stack]  
② TOP [stack] pc  
③ PC ← New address [H]  
④~~

Overhead of function call: [Microprogram for function call]

- ①  $\text{Top}[\text{stack}] = \text{PC}$
  - ② Push [stack]
  - ③  $\text{PC} \leftarrow \text{New address}$  [if one exists]
  - ④ Pop [stack]
  - ⑤  $\text{PC} \leftarrow \text{Top}[\text{stack}]$

E.g. How much stack space required to run given program.

```

main()
{ int p,q;
  f();
  g();
}

```

7678440

Algo fact(n)

```
    { if (n < 1)  
        return 1;  
    else  
        return n *  
            fact(n - 1);  
}
```

$$Sc = \Theta(n)$$

```

graph TD
    factn[fact(n)] --> factn1[fact(n-1)]
    factn1 --> factn2[fact(n-2)]
    factn2 --> fact1[fact(1)]
    fact1 --> depth[Depth of recursion n levels]
    fact1 --> stack["[n stack entries to run]"]
  
```

A recursion tree diagram for the factorial function. The root node is  $\text{fact}(n)$ . An arrow points down to the child node  $\text{fact}(n-1)$ . Another arrow points down to the child node  $\text{fact}(n-2)$ . A third arrow points down to the child node  $\text{fact}(1)$ . To the right of the tree, there is an upward-pointing arrow above the text "Depth of recursion n levels". Below the tree, there is a downward-pointing arrow above the text "[n stack entries to run]".

◻  $Tc$  of  $rec(n)$ :  $T(n)$  is  $Tc$  of  $rec(n)$

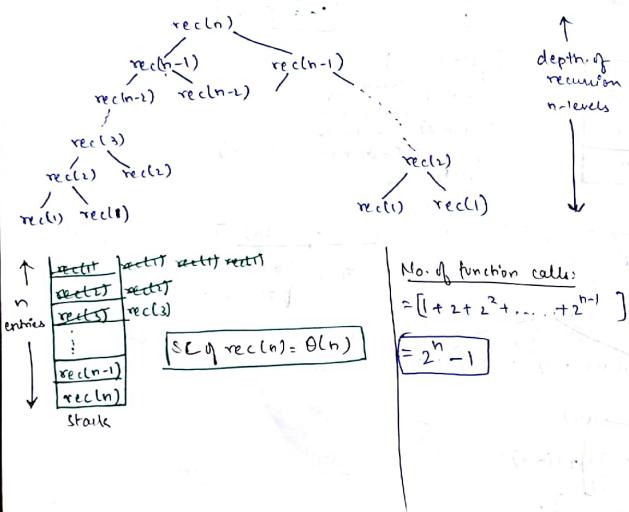
$$T(n) = \begin{cases} a = \Theta(1) & n \leq 1 \\ 2T(n-1) + c & n \geq 1 \end{cases}$$

Substitution method to solve recurrence relation

Q. Algo Rec(n)  
 { if ( $n \leq 1$ ) }  
 return(1) ] a  
 else  
 return (rec( $n-1$ ) + rec( $n-1$ ) + n);  
 }  
 rec(n) → T(n)  
 {  $x = rec(n-1)$  }  
 $y = rec(n-1)$  ]  $T(n-1)$   
 return ( $x+y+n$ ) ] c

$$\begin{aligned}
 T(n) &= 2T(n-1) + c \quad [T(n-1) = 2T(n-2) + c] \\
 &= 2[2T(n-2) + c] + c \\
 &= 2^2 T(n-2) + 2c + c \quad [T(n-2) = 2T(n-3) + c] \\
 &= 2^2 [2T(n-3) + c] + 2c + c \\
 &= 2^3 T(n-3) + c(2^2 + 2 + 1) \\
 &\quad \vdots k \text{ times} \\
 &= 2^k T(n-k) + c[2^{k-1} + \dots + 2 + 1] \\
 &= 2^k T(n-k) + c[2^k - 1] \xrightarrow{\text{sum}} s_n = \frac{a(r^n - 1)}{r - 1} ; r > 1 = \frac{1(2^k - 1)}{2 - 1} = 2^k - 1 \\
 n-k-1 &\rightarrow k-1 \\
 &= 2^{n-1} T(1) + c[2^{n-1}] \\
 \boxed{T(n) = a \cdot 2^{n-1} + c \cdot 2^{n-1} + c} \\
 T(n) &= \Theta(2^n)
 \end{aligned}$$

(b) SC and # of function calls of Rec(n):



**Note:**  
For recursive algorithm if each function call required const. time (except time for subroutine call), then  $TC$  of algo. =  $O(\# \text{ of function calls})$

④ Value returned by algorithm:  $\text{rec}(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 2 \cdot \text{rec}(n-1) + n & \text{if } n > 1 \end{cases}$   $\Theta(2^n)$

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

$$\begin{aligned}
 &= 2[T(n-2) + (n-1)] + n \\
 &= 2^2 T(n-2) + 2(n-1) + n \\
 &= 2^2 [2T(n-3) + (n-2)] + 2(n-1) + n \\
 &= 2^3 T(n-3) + 2^2(n-2) + 2(n-1) + n
 \end{aligned}$$

Q. Algo rec(n)

```

if (n ≤ 1)
    return (1);
else
    return (2 * rec(n-1) + n);
}

```

⑤ TC of rec(n)

$$\begin{aligned}
 T(n) &= \begin{cases} a & \text{if } n \leq 1 \\ T(n-1) + c & \text{if } n > 1 \end{cases} \\
 T(n) &= T(n-1) + c \\
 &= T(n-2) + 2c \\
 &= T(n-3) + 3c \\
 &\quad \vdots k \text{ times} \\
 &= T(n-k) + k \cdot c \\
 \text{Termination} &\quad \vdots \\
 &\quad \text{if } (n-k+1) \rightarrow k=n-1 \\
 &= T(1) + (n-1)c \\
 T(n) &= a + c \cdot (n-1) \Rightarrow T(n) = \Theta(n)
 \end{aligned}$$

⑥ SC & No. of function calls:

rec(n)  $\uparrow$   
 $\downarrow$  depth of recursion  
 $\downarrow$  n-levels  
 $\downarrow$  Rec(2)  
 $\downarrow$  Rec(1)

$\boxed{SC = \Theta(n)}$

No. of function calls =  $n$

⑦ Value returned by algorithm:

rec(n)  $\begin{cases} 1 & \text{if } n \leq 1 \\ 2 \cdot \text{rec}(n-1) + n & \text{if } n > 1 \end{cases}$

```

Q. Algo Rec(n)
{
    if(n≤1)
        return (1);
    else
        { rec(n-1)
            for (i=1;i<n;i++)
                printf("
}

```

$$\textcircled{6} \quad \underline{\text{TC of rec}(n)}: T(n) = \begin{cases} \alpha; & n \leq 1 \\ T(n-1) + n; & n > 1 \end{cases}$$

$$\begin{aligned} T(n) &= T(n-1) + n \quad [T(n-1) = T(n-2) + (n-1)] \\ &= T(n-2) + (n-1) + n \\ &= T(n-k) + (n-k) + (n-1) + n \\ &\quad \downarrow k \text{ times} \\ &= T(1) + (n-(k-1)) + \dots + (n-1) + n \end{aligned}$$

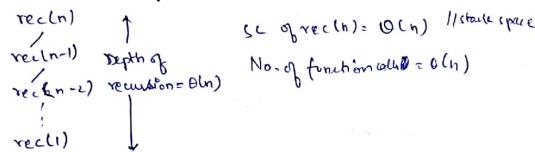
Termination

$$k = n - 1$$

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

$$T(n) = \frac{n(n+1)}{2} = O(n^2)$$

(b) SC and no. of function calls:



```

Q. Algo rec(n)
      { if (n≤1)
          return (1)
        else
          { return (rec(1
                    rec(n-1));
          for (i=1; i≤n;
                print (" ")
            }

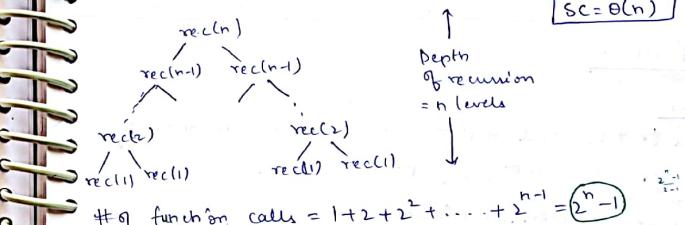
```

$$\begin{aligned}
 \text{a) } T(n) &= \underbrace{T(n-1)}_{2T(n-1)+n} + n = \begin{cases} a & ; n \leq 1 \\ 2T(n-1) + \cancel{n+n+n+n} + n & ; n > 1 \end{cases} \\
 T(n) &= 2T(n-1) + n \quad T(n-1) = 2T(n-2) + (n-1) \\
 &= 2(2T(n-2) + n-1) + n \quad T(n-2) = 2T(n-3) + (n-2) \\
 &= 2^2 T(n-2) + 2(n-1) + n \\
 &= 2^2 [2T(n-3) + (n-2)] + 2(n-1) + n \\
 &= 2^3 T(n-3) + 2^2(n-2) + 2(n-1) + n \\
 &\vdots k \text{ times} \\
 &= 2^k T(\underbrace{n-k}_{\downarrow}) + 2^{k-1}(n-(k-1)) + \dots + 2(n-1) + n
 \end{aligned}$$

$$\Rightarrow T(n) = 2^{n-1} \cdot 1 + 2^{n-2} \cdot (n - (n-2)) + 2^{n-3} \cdot (n - (n-3)) + \dots$$

$$\begin{aligned}
 T(n) &= 2^{n-1} + 2^{n-2} + 2^{n-3} + \dots + 2^{(n-1)} + n \\
 2T(n) &= 2^n + 2^{n-1} + 2^{n-2} + \dots + 2^{(n-1)} + 2n \\
 2T(n) - T(n) &= [2^n + 2^{n-1} + 2^{n-2} + \dots + 2^2 + 2] - n \\
 T(n) &= \left[ \frac{2(2^n - 1)}{2 - 1} \right] - n \Rightarrow T(n) = 2^{n+1} - 2 - n \\
 T(n) &= O(2^n)
 \end{aligned}$$

b) SC and no. of function calls:



## Simplification of A.P. & G.P. Series:

$$5 \cdot T(n) = 5^n \cdot n + 5^{n-1} \cdot (n-1) + \dots + 5^2 \cdot 2 + 5^1 \cdot 1$$

$$5T(n) = 5^{n+1} \cdot n + 5^n \cdot (n-1) + \dots + 5^3 \cdot 3 + 5^2 \cdot 2$$

$$5T(n) - T(n) = 5^{n+1} \cdot n - 5^n + 5^{n-1} + \dots + 5^3 + 5^2 - 5$$

$$4T(n) = 5^{n+1} \cdot n - [5^n + 5^{n-1} + \dots + 5^3 + 5^2 + 5]$$

$$4T(n) = 5^{n+1} \cdot n - \frac{5}{4}(5^n - 1)$$

$$T(n) = \frac{1}{4} \left[ 5^{n+1} \cdot n - \frac{5(5^n - 1)}{4} \right] \quad \text{∴ } T = \Theta(n \cdot 5^{n+1})$$

$$\therefore T(n) = \frac{n}{4^n} + \frac{n-1}{4^{n-1}} + \frac{n-2}{4^{n-2}} + \dots + \frac{2}{4^2} + \frac{1}{4}$$

$$\frac{T(n)}{4} = \frac{n}{4^{n+1}} + \frac{n-1}{4^n} + \frac{n-2}{4^{n-1}} + \dots + \frac{2}{4^3} + \frac{1}{4^2}$$

$$\frac{3T(n)}{4} = \frac{1}{4^n} + \frac{1}{4^{n-1}} + \frac{1}{4^{n-2}} + \dots + \frac{1}{4^3} + \frac{1}{4^2} + \frac{1}{4} - \frac{n}{4^{n+1}}$$

$$\left[ \frac{\frac{1}{4}}{\frac{3}{4}} \left( 1 - \frac{1}{4^n} \right) \right] = \frac{1}{3} \left[ \left( 1 - \frac{1}{4^n} \right) \right]$$

$$\frac{3T(n)}{4} = \frac{1}{3} \left[ \left( 1 - \frac{1}{4^n} \right) \right] - \frac{n}{4^{n+1}} \Rightarrow T(n) = \frac{4}{3} \left[ \frac{1}{3} \left( 1 - \frac{1}{4^n} \right) \right] - \frac{n}{4^{n+1}}$$

$$T(n) = \frac{4}{3} \left[ \frac{1}{3} \left( 1 - \frac{1}{4^n} \right) - \frac{n}{4^{n+1}} \right]$$

$\boxed{T(n) = \Theta(1)}$

leading term is constant

$$\therefore T(n) = \frac{1}{3^n} + \frac{2}{3^{n-1}} + \frac{3}{3^{n-2}} + \dots + \frac{n-1}{3^2} + \frac{n}{3} \quad \underline{\underline{\Theta(n)}}$$

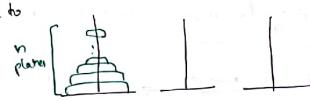
$\frac{T(n)}{3}$

### # Towers of Hanoi:

Problem statement:

Given 3 towers [X, Y, Z] and n plates in Tower X with descending order of diameter from bottom to top.

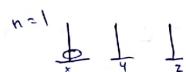
Compute no. of plate movements required to transfer n plates from X to Y:



Constraints:

- Only top plate allowed to move from tower
- At any time lower diameter plate not below upper diameter plate.

$$n=1 : \# \text{ of moves} = 1 (X \rightarrow Y)$$



$$n=2 :$$

$$\# \text{ of moves} = 3 [X \rightarrow Z, X \rightarrow Y, Z \rightarrow Y]$$



$$n=3 :$$

$$\begin{aligned} a &: X \rightarrow Y, X \rightarrow Z, Y \rightarrow Z \\ b &: X \rightarrow Y \\ c &: Z \rightarrow X, Z \rightarrow Y, X \rightarrow Y \end{aligned} \quad 7 \text{ moves}$$



- Procedure to transfer n plates from X to Y using 2 temp:
- ① Transfer Top (n-1) plates from X to Z using Y as temp.
- ② move one plate from X to Y
- ③ Trans (n-1) plates from Z to Y using X as temporary.

Algo TOH(X, Y, Z, n)

{ //TOH (source, dest, temp, # of plates)

if (n==1)

return (X → Y)

else

{ TOH(X, Z, Y, n-1)

move X → Y

TOH(Z, Y, X, n-1)

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

T(X, Y, Z, 4) (15 moves)

7 moves

T(X, Z, Y, 3)

7 moves

T(Y, Z, X, 2)

7 moves

T(Y, X, Z, 1)

7 moves

T(Z, Y, X, 1)

7 moves

T(X, Y, Z, 1)

7 moves

T(X, Z, Y, 1)

7 moves

T(Y, Z, X, 1)

7 moves

T(Y, X, Z, 1)

7 moves

T(Z, Y, X, 1)

7 moves

T(X, Y, Z, 1)

7 moves

T(X, Z, Y, 1)

7 moves

T(Y, Z, X, 1)

7 moves

T(Y, X, Z, 1)

7 moves

T(Z, Y, X, 1)

7 moves

T(X, Y, Z, 1)

7 moves

T(X, Z, Y, 1)

7 moves

T(Y, Z, X, 1)

7 moves

T(Y, X, Z, 1)

7 moves

T(Z, Y, X, 1)

7 moves

T(X, Y, Z, 1)

7 moves

T(X, Z, Y, 1)

7 moves

T(Y, Z, X, 1)

7 moves

T(Y, X, Z, 1)

7 moves

T(Z, Y, X, 1)

7 moves

T(X, Y, Z, 1)

7 moves

T(X, Z, Y, 1)

7 moves

T(Y, Z, X, 1)

7 moves

T(Y, X, Z, 1)

7 moves

T(Z, Y, X, 1)

7 moves

T(X, Y, Z, 1)

7 moves

T(X, Z, Y, 1)

7 moves

T(Y, Z, X, 1)

7 moves

T(Y, X, Z, 1)

7 moves

T(Z, Y, X, 1)</

Q. Algo rec(n)  
 { if (n ≤ 1)  
   return(1)  
 else  
   return (rec(n/2) + rec(n/2) + n)  
 }

$$\text{① } T(n) = 2T(n/2) + c \quad \because T(n/2) = 2T(n/2) + c  
= 2[2T(n/2) + c] + c  
= 2^2 T(n/2) + 2c + c  
= 2^2 [2T(n/2) + c] + 2c + c  
= 2^3 T(n/2) + 2^2 c + 2c + c  
\vdots k \text{ times}  
= 2^k T(n/2) + c[2^{k-1} + \dots + 2^0 + 1]$$

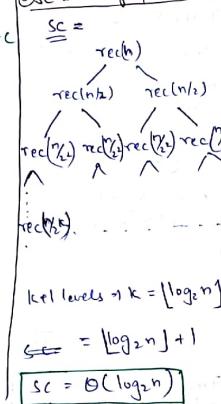
$$\text{Terminates } \frac{n}{2^k} = 1 \Rightarrow n = 2^k \quad 2^k - 1 = n - 1$$

$$T(n) = nT(1) + c[n-1]$$

$$T(n) = a.n + c(n-1) = \Theta(n) = T.C$$

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

⑥ SC & no. of function calls:



$$\text{SC} = \Theta(\log_2 n)$$

$$\text{No. of function calls: } \Rightarrow 1 + 2 + 2^2 + \dots + 2^k = 2^{k+1} - 1$$

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

② Value returned by rec(n):

$$\text{rec}(n) = \begin{cases} a & ; n \leq 1 \\ 2 \cdot \text{rec}(n/2) + n & ; n > 1 \end{cases}$$

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2[2T(n/2) + n/2] + n \\ &= 2^2 T(n/4) + 2n \\ &= 2^2 [2T(n/4) + n/4] + 2n \\ &= 2^3 T(n/8) + 3n \end{aligned}$$

$$\begin{aligned} &\text{k times:} \\ &= 2^k T(n/2^k) + k \cdot n \\ &\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \log_2 n \\ &= nT(1) + n \log_2 n \\ &= n! + n \log_2 n \Rightarrow n(1 + \log_2 n) \quad \Theta(n \log n) \end{aligned}$$

⑥ Algo rec(n)  
 { if (n ≤ 2)  
   return;  
 else  
   return (rec(n/2) + rec(n/2) + c)

$$\text{② } T(n) = 2T(n/2) + c$$

$$\begin{aligned} &= 2[2T(n/4) + c] + c \\ &= 2^2 T(n/8) + 2c + c \\ &= 2^3 T(n/16) + 2^2 c + 2c + c \\ &\vdots k \text{ times} \\ &= 2^k T(n/2^k) + c[2^{k-1} + \dots + 2^0 + 1] \\ &n/2^k = 2 \Rightarrow \frac{1}{2^k} \log_2 n = 1 \Rightarrow \log_2 n = 2^k \\ &n = 2^k \end{aligned}$$

$$= \log_2 n T(1) + c[2^{k-1} + \dots + 2^0 + 1]$$

$$= \log_2 n \cdot a + c[\log_2 n - 1]$$

$$T(n) = a \log_2 n + c[\log_2 n - 1] = \Theta(\log_2 n)$$

No. of function calls:

$$\Rightarrow 1 + 2 + 2^2 + \dots + 2^k = 2^{k+1} - 1$$

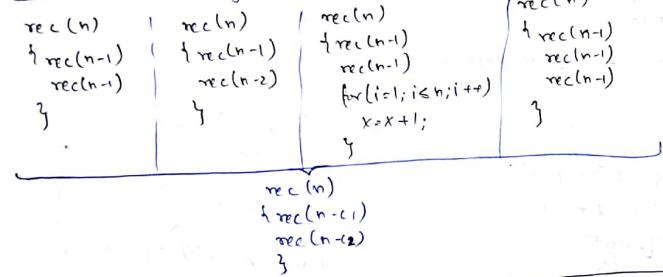
$$\Rightarrow 2 \log_2 n - 1 = \Theta(\log_2 n)$$

③ Value returned by rec(n):

rec function	SC of rec(n)
rec(n) = rec(n-1) + n	$\approx \frac{n}{2} = \Theta(n)$
rec(n) = rec(n/2) + n	$\approx \log_2 n = \Theta(\log_2 n)$
rec(n) = rec( $\sqrt{n}$ ) + n	$\approx \log_2 \log_2 n = \Theta(\log_2 \log_2 n)$

\* NOTE

### Exponential TC rec. Algo:



### # fast Exponential Algorithm:

Compute:  $x^n$

To compute  $x^n$

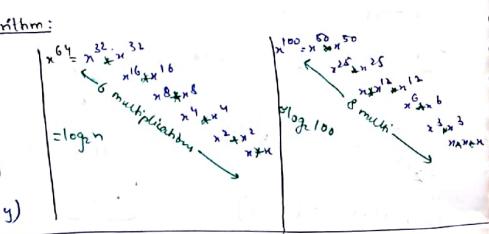
(i)  $y = x^{n/2}$

(ii) if  $n$  is even

return( $y, y$ )

else

return ( $n, y, y$ )



Mgo. FastExp( $x, n$ )

```
{
    // Compute  $x^n$ 
    if ( $n == 0$ ) return (1);
    else if ( $n == 1$ ) return ( $x$ );
    else

```

```

        if ( $y = \text{FastExp}(x, n/2)$ ;
            if ( $n \% 2 == 0$ )
                return ( $y * y$ );
            else
                return ( $x * y * y$ );
        }
    }
}
```

TC =  $\Theta(\log n)$

### TC of Fast Exp Algo:

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

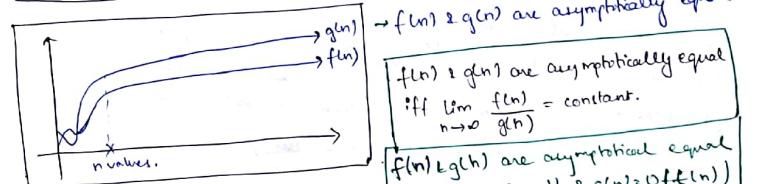
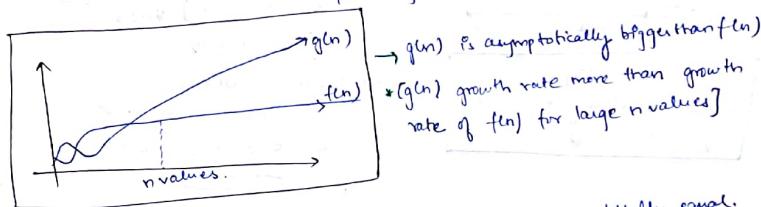
SC of fast Exp Algo(Recursive) =  $\Theta(\log n)$

No. of function calls =  $\Theta(\log n)$

### # ASYMPTOTIC NOTATIONS:

Asymptotic comparison of non-negative functions: Growth rate comparison for non-negative functions for large  $n$ -values.

$f(n), g(n)$  Asymptotic comparison: growth rate comparison of  $f(n) & g(n)$  for large  $n$  values.



$f(n), g(n)$  are asymptotically equal  
iff  $f(n) = O(g(n))$  &  $g(n) = O(f(n))$

$f(n) = 10n^2 + 20n$   
 $g(n) = 2n^2 + 100n + 100$   
 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{n^2[10+20/n]}{n^2[2+100n+100/n^2]} = \frac{\infty}{\infty} \therefore f(n) & g(n)$  are asympt. equal

$f(n)$  asymptotically bigger than  $g(n)$ : iff  
 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{5n^2 + 10}{10n + 20} = \frac{\infty}{\infty} = \infty$$

$\therefore f(n)$  asympt. bigger than  $g(n)$

$f(n)$  asymptotically smaller than  $g(n)$ : iff.  
 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$

$$\lim_{n \rightarrow \infty} \frac{2n^2}{n^3 + 10n^2} = \frac{2}{\infty} = 0$$

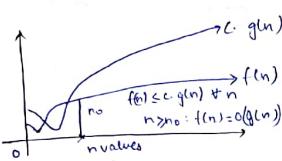
$\therefore f(n)$  asympt. smaller than  $g(n)$

## # Big-O Notation:

$\rightarrow f(n), g(n)$  are non-negative functions.

$\rightarrow f(n) = O(g(n)) \text{ iff}$

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



$f(n) = O(g(n)) \text{ iff}$   
 $g(n)$  asymptotically bigger or equal to  $f(n)$

E.g.  $2n+3 = O(n)$

$$\begin{aligned} f(n) &= 2n+3 & g(n) &= n \\ [2n+3] &\leq c[n] \text{ for all values } n \geq n_0 \\ [2n+3] &\leq 3[n] \quad \text{then } 2n+3 = O(n) \\ f(n) &\in O(g(n)) \end{aligned}$$

E.g.  $2n+3 = O(n^2)$

$$\begin{aligned} f(n) &= 2n+3 & g(n) &= n^2 \\ 2n+3 &\leq c[n^2] \quad \text{then } n \geq n_0 \\ [2n+3] &\leq O(n^2) \end{aligned}$$

E.g.  $2n+3 = O(\log_{10} n)$

$$\begin{array}{cccccc} 10 & 100 & 10^3 & 10^4 & 10^5 \\ \hline f(n) = 2n+3 & 23 & 203 & 2003 & 20003 \\ g(n) = 10000 \cdot \log_{10} n & 10000 & 20000 & 30000 & 40000 \\ 2n+3 \leq c \cdot \log_{10} n & \text{not true for all } n \text{ values} \end{array}$$

## # Asymptotic Comparisons:

\* Decrement function: (lesser)

$$f(n) = \frac{c}{n}, \frac{1}{n^2}, \frac{n^2}{2^n}$$

$$\left[ \frac{n^2}{2^n} < \frac{1}{n^2} < \frac{c}{n} \right]$$

\* Constant functions: (Bigger than decrement)

$$f_2(n) = \text{constant}$$

\* Log functions: (Bigger than constants)

$$\log n, (\log n)^k, (\log \log n), (\log \log n)^k$$

$$[\log n < (\log \log n)^k < \log n < (\log n)^k]$$

(smallest poly bigger than log largest)

\* Polynomial functions:

$$n^{0.1}, n^2, n^{0.5}, n \log n, n^3$$

$$[n^{0.1} < n^{0.5} < n \log n < n^2]$$

\* Exponential Functions: (Bigger than poly)

$$2^n, 3^n, n!, n^n, (1.01)^n$$

$$[(1.01)^n < 2^n < 3^n < n! < n^n]$$

## Asymptotic Comp.

dec < const < log < poly < exponential

Q1. Which is false?

①  $100 \log n = \frac{\log n}{100} = O(\log n)$

② if  $x < y$  then  $n^x = O(n^y)$

which is true?

③  $f(n) = O(g(n))$

④  $g(n) = O(f(n))$

⑤  $n^a = O(a^n) \quad (a > 1 \text{ const})$

⑥  $\sqrt{\log n} = O(\log \log_2 n)$

⑦  $f(n) = n \cdot \log_2 n$  ;  $g(n) = \log_2 n$

$f(n) \& g(n)$  asymp. equal

⑧  $f(n) = \begin{cases} n^3 & 0 \leq n \leq 10000 \\ n & n > 10000 \end{cases}$

⑨  $f(n) = O(g(n))$

⑩  $a \& b$

⑪  $g(n) = O(f(n))$

⑫ None

⑫  $g(n) = \begin{cases} n & 0 \leq n \leq 100 \\ n^2 & n > 100 \end{cases}$

Which is true?

⑬  $f(n) = \begin{cases} 2^n & \text{for even } n \\ n & \text{otherwise} \end{cases}$

⑭  $g(n) = \begin{cases} 2^n & \text{for odd } n \\ n & \text{otherwise} \end{cases}$

Which is true?

⑮  $f(n) = \sqrt{n}$        $g(n) = n^{1+\epsilon/n}$

Which is true?

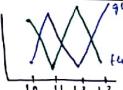
⑯  $f(n) = O(g(n))$

⑰  $a \& b$

⑱  $g(n) = O(f(n))$

⑲ None

$n$	$f(n) = n^{0.5}$	$g(n) = n^{1+\epsilon/n}$
90	$(90)^{0.5} < (90)^2$	
180	$(180)^{0.5} < (180)^1$	
270	$(270)^{0.5} > (270)^{-1}$	



$f(n) \& g(n)$  are asymp. non comparable  
 $f(n) \neq g(n) \quad g(n) \neq O(f(n))$

⑥  $f(n) = n^{1+\sin n}$      $g(n) = n^2$ ; which is true?

- ①  $f(n) = O(g(n))$   
 ②  $g(n) = O(f(n))$

③ a & b  
④ None

~~\*\*\*~~  $f(n) = n!$      $g(n) = n^n$

$f(n) = n(n-1)(n-2) \dots 2 \times 1$

$g(n) = n \cdot n \cdot n \cdot \dots \cdot n \cdot n$

$n^n$  asymptotically bigger than  $n!$   
 $n! = O(n^n)$   
 $n^n \neq O(n!)$

$\Rightarrow f(n) = n!$      $g(n) = (n-1)! \quad | \quad h(n) = (n+1)! + n$

$(n-1)! + n = (n-1)! < n! \Rightarrow g(n) = n! < f(n)$

$\Rightarrow f(n) = 2^n$      $g(n) = n^n$

solution:  $\log(f(n)) > \log(g(n))$   
 $= \log_2 2^n = n \log_2 2 = n \cdot \log_2 n = \sqrt{n} \cdot \log_2 n$   
 $n^n = O(2^n)$   
 $2^n + O(n^n) = 2^n + O(n \cdot 2^n) = 2^n + O(n \cdot \sqrt{n} \cdot \log_2 n)$

If  $\log f(n)$  asympt. bigger than  $\log g(n)$ , then  $f(n)$  asympt. bigger than  $g(n)$

$\rightarrow$  If  $\log f(n)$  asympt. bigger than  $\log g(n)$  then  $f(n)$  asympt. bigger than  $g(n)$ .

$\rightarrow$  If  $f(n)$  asympt. bigger than  $g(n)$  then:  $[\log f(n) \text{ asympt. bigger than } \log g(n)]$

e.g.  $f(n) > g(n)$   
 $2^n > n^2 \Rightarrow \log(2^n) > \log(n^2)$   
 $n^3 > n^2 \Rightarrow \log(n^3) = \log(n^2)$

or  
 $[\log f(n) \text{ asympt. equal to } \log g(n)]$

$\rightarrow$  If  $\log f(n)$  asympt. equal to  $\log g(n)$  then  $f(n) > g(n)$  ays.

⑤  $f(n) < g(n)$  ays.  
 ⑥  $f(n) = g(n)$  ays.

$\rightarrow$  If  $f(n)$  &  $g(n)$  ays. equal, then  $\log f(n), \log g(n)$  also ays. equal

~~\*\*\*~~  $f(n) = \log(n!)$   
 $g(n) = \log(n^n) = n \log n$

Stirling formula:  
 $n! \approx \sqrt{2\pi n e} \left(\frac{n}{e}\right)^n$

$\therefore \log(n!) \approx \log\left(\sqrt{2\pi n e} \left(\frac{n}{e}\right)^n\right)$

$\Rightarrow \frac{1}{2} \log 2\pi n e + n \log\left(\frac{n}{e}\right) = \left[\frac{1}{2} \log 2\pi e + \frac{1}{2} \log n + n \log n - n \log e\right]$

$\log(n!) \approx (n \log n)$      $\log n! = O(n \log n)$   
 $n \log n = O(\log n!)$

$\therefore f(n)$  and  $g(n)$  are asymptotically equal.

25/08/2017

# Testing if  $f(n)$  is polynomially bounded or exponentially bounded:

$\rightarrow$  if  $\log f(n) = O(\log n)$   
 $\log f(n) \leq \log n$  [Asymptotically]

Then  $f(n)$  is polynomially bounded.

$\rightarrow$  if  $\log f(n) \neq O(\log n)$   
 $\log f(n) > \log n$  [Asymptotically]

Then  $f(n)$  is exponentially bounded.

E.g.  $f(n) = n^k$  // poly fun.  
 $\log f(n) = k \log n = O(\log n)$

E.g.  $f(n) = 2^n$   
 $\log f(n) = n \log 2 \neq O(\log n)$ .  
// Expo. fun.

~~Q.~~  $f(n) = n!$   
 $O(n \log n) \neq O(\log n)$   
Exponential

~~Q.~~  $f(n) = (\log n)!$   
 $\log(n) \approx (\log n) \log(\log n) \leq \log n$   
Polynomial      Exponential

③  $f(n) = (\log \log n)!$   
 [Polynomial]  
 $\log f(n) = \log \log n \log(\log \log n)$

④  $f(n) = \log(n!)$   
 $= n \log n$   
 $\log f(n) = \log(n \log n)$   
 $= \log n + \log \log n \leq \log n$   
 [Polynomial]

⑤  $f(n) = n^{\sqrt{n}}$   
 $\log f(n) = \sqrt{n} \log n + O(\log n)$   
 [Exponential]

⑥  $f(n) = n^{\log n}$   
 $\log f(n) = \log n \log n + \log \log n$   
 [Exponential]

⑦  $f(n) = (\log n)^{\log n}$   
 $\log f(n) = \log n \log(\log n) + O(\log n)$   
 [Exponential] [Exponential]

⑧  $f(n) = (\log n)^{\log \log n}$   
 $\log f(n) = (\log \log n)^{\frac{1}{\log n}} + O(\log n)$   
 [Polynomial]

⑨  $f(n) = (\log \log n)^{\log n}$   
 $\log f(n) = \log n \cdot \log(\log \log n) + O(\log n)$   
 [Exponential]

⑩  $f(n) = \log(\log n)!$   
 $\log f(n) = \log(\log(\log n))$   
 [Polynomial]

**# Omega Notation ( $\Omega$ ):**  
 $\Rightarrow f(n)$  and  $g(n)$  non negative functions.  
 $f(n) = \Omega(g(n))$  iff  $f(n) \geq c \cdot g(n) \quad \forall n \geq n_0$ .

$\rightarrow f(n) = \Omega(g(n))$  iff  $g(n) = O(f(n))$ .  
 E.g.  $2^n = \Omega(n^m)$   
 $n^m = O(2^n)$   
 $2^n = \Omega(n^m)$

**# Theta Notation ( $\Theta$ ):**  
 $\Rightarrow f(n)$  and  $g(n)$  non negative functions.  
 $f(n) = \Theta(g(n))$  iff  $c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall n \geq n_0$   
 $[c_1, c_2, n_0 \rightarrow \text{constants}]$

$\rightarrow f(n) = \Theta(g(n))$  iff  $g(n)$  asymptotically equal to  $f(n)$ .  
 $\rightarrow f(n) = \Theta(g(n))$ ; if  $f(n) = O(g(n))$  &  $f(n) = \Omega(g(n))$

**# Little-O Notation:**  
 $\Rightarrow f(n)$  and  $g(n)$  are non-negative functions.  
 $f(n) = o(g(n))$  iff  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$   
 $f(n) = o(g(n))$  iff  $g(n)$  asymptotically bigger than  $f(n)$ .

Test:  
 1.  $n! = o(n^n)$  ✓  
 2.  $2^n = o(n^n)$  ✓  
 3.  $\log n! = o(n \log n)$  X  
 4.  $\log n! = O((\log n)!)$  ✓  
 5.  $n^2 \log n = O(n^{2.0})$  ✓

### # Little Omega ( $\omega$ ):

$\rightarrow f(n) \geq g(n)$  non negative function.  
 $f(n) = \omega(g(n))$  iff  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$

$\rightarrow f(n) \geq \omega(g(n))$  iff  $g(n)$  asymptotically smaller than  $f(n)$ .

Q. Find given: true/false:

- ①  $2^{n+1} = \Theta(2^n)$  ✓
- ②  $2^{n+1} = \Omega(2^n)$  ✗
- ③  $x^{n+1} = \Theta(x^n)$  ✓
- ④  $x^{2n+1} = \Theta(x^n)$  ✗
- ⑤  $n! = \Theta(n^n)$  ✗
- ⑥  $\log n! = \Theta(n \log n)$  ✓

$\log n = (\log n)^m$

Q. Write given functions in asymptotically increasing order:

$$\begin{aligned} \log^k \log n &= (\log \log n)^k \\ \log \log^k n &= \log(\log n)^k \\ &= \log(k \cdot \log n) \\ &= \log k + \log \log n \\ \log \log \log^k n &= \log \log(k \cdot \log n) \\ &= \log \log k + \log \log \log n \end{aligned}$$

$$\log \log \log^k n = \log \log \log^k n = \log \log \log^k n \leq \log^k \log n$$

Q. Write in any. increasing order:

$$\begin{aligned} \log n! ; \sqrt{n} ; (\log n)! ; (\log n)! ; 2^n ; 2^{n/2} ; 2^n \\ 2^{n+1} ; (\log n)^{\log n} ; (\log n)^n ; n^{\log n} ; n \log n \end{aligned}$$

Polynomial:

$$\log n! ; \sqrt{n} ; (\log n)! ; n \log n ;$$

$(\log n)^n$

Exponential:

$$\begin{aligned} (\log n)! ; 2^n ; 2^{n/2} ; 2^n ; 2^{n+1} \\ (\log n)^n ; n^{\log n} ; (\log n)^{\log n} \\ (\log n)^n \end{aligned}$$

$\log \log n!$

$\log n! \approx \log n$

$\log n = \log n + \log \log n$

$\sqrt{n} = \frac{1}{2} \log n$

$\log n \approx n \log \log n$

$(\log n)! \approx \log \log n \log \log \log n$

$(\log \log n)! \leq \sqrt{n} \leq \log n = n \log n \leq (\log n)! \leq (\log n)^n \leq n^n \leq 2^{n/2} \leq 2^n \leq 2^{n+1} \leq 2^{2n} \leq (\log n)^2$

Q.  $O(n) + \Omega(n) + \Theta(n) =$

①  $O(n)$

②  $\Omega(n)$

③  $\Theta(n)$

$O(n)$

$\Omega(n)$

$\Theta(n)$

$O(n) \leq n + \Omega(n) \geq n + \Theta(n) = n = \Theta(n)$

Q.  $f(n) = O(n)$

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

$h(n) = \Theta(n)$

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

### # Armstrong Rules over notations:

① Reflexivity: [O,  $\Omega$ ,  $\Theta$  allows reflexivity]

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

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

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

② Transitivity: If  $f(n) = O(g(n))$  and  $g(n) = O(h(n))$ , then  $f(n) = O(h(n))$ .  
 Similarly,  $\Omega$ ,  $\Theta$ ,  $\omega$  also represent transitive property.

③ Asymmetry: ①  $O, \Omega$

$$f(n) = O(g(n)) \text{ iff } g(n) = \Omega(f(n))$$

②  $\Omega, \omega$

$$f(n) = \Omega(g(n)) \text{ iff } g(n) = \omega(f(n))$$

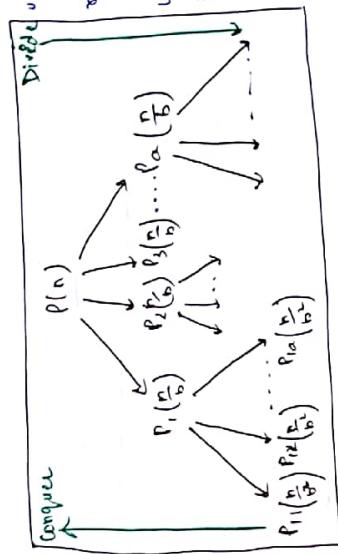
## # DIVIDE AND CONQUER DESIGN TECHNIQUE:

- Problem P with "n" inputs of not small problem.
- Divide P into "a" no. of subproblems  $[P_1, P_2, \dots, P_a]$  each with  $\frac{n}{b}$  inputs.

$$f(n) * c = \frac{f(n)}{c} = \mathcal{O}(f(n))$$

$$f(n) * c = \frac{f(n)}{c} = \mathcal{O}(f(n))$$

$$f(n) * c = \frac{f(n)}{c} = \mathcal{O}(f(n))$$



- If subproblem not small, divide into sub-subproblems and conquer solution (P).
- Until subproblem becomes small return solution of small problem.

Q. Find True/False:

$$f(n) + g(n) = \Theta(\max(f(n), g(n))) \quad \text{True}$$

$$f(n) + g(n) = \Omega(\max(f(n), g(n))) \quad \text{True}$$

$$f(n) + g(n) = \Theta(\max(f(n), g(n))) \quad \text{True}$$

$$f(n) + g(n) = \mathcal{O}(\max(f(n), g(n))) \quad \text{False}$$

$$f(n) + g(n) = \omega(\min(f(n), g(n))) \quad \text{False}$$

Q. Test  $f(n) = O((f(n))^2)$  False

$$f(n) = \frac{1}{n} \Rightarrow \mathcal{O}\left(\frac{1}{n^2}\right) \quad \text{False}$$

$$Q. Test \quad f(n) = O(g(n)) \quad \text{from } 2^{f(n)} = O(2^{g(n)}) \quad \text{False}$$

$$f(n) = 2^n \quad g(n) = n \quad 2^n = \mathcal{O}(2^n) \quad (\text{False})$$

$$2^n = \mathcal{O}(n) \Rightarrow 2^n = \mathcal{O}(2^n) \quad (\text{False})$$

Q.  $(x+a)^n = \mathcal{O}(x^n)$ . a is const.

$x, n$  not const.

$$(x+a)^n = \mathcal{O}(x^n + \mathcal{O}_1 x^{n-1} + \dots + \mathcal{O}_n x + \mathcal{O}_0) = \mathcal{O}(x^n)$$

$$\mathcal{O}(x^n + \mathcal{O}_1 x^{n-1} + \dots + \mathcal{O}_n x + \mathcal{O}_0) \leq [C_1 + C_2 + \dots + C_n] \cdot x^n$$

$$C_1 \cdot x^n \leq [C_1 x^n + C_2 x^{n-1} + \dots + C_n x + C_0] \leq [C_1 x^n + C_2 x^{n-1} + \dots + C_n x + C_0] \leq \text{const. } g(n)$$

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

$$(x+a)^n = \Theta(x^n)$$

Proof:  $c_1 \cdot x^n \leq [x+a]^n \leq [C_1 x^{n-1} + \dots + C_0] \cdot x^n$

$C_1 \cdot x^n \leq [C_1 x^n + C_2 x^{n-1} + \dots + C_n x + C_0] \leq [C_1 x^n + C_2 x^{n-1} + \dots + C_n x + C_0] \leq \text{const. } g(n)$

Recurrence Relation of D and L algo:

$$T(n) = \begin{cases} q(n) = \Theta(1) & ; \text{ if } n=1 \\ \mathcal{O}(T(\frac{n}{b}) + f(n)) & ; \text{ if } n>1 \end{cases}$$

add constants.

$$T(n) \rightarrow \mathcal{T}C \text{ of } D \text{ and } L(n)$$

$$q(n) \rightarrow \mathcal{T}C \text{ to solve small problem}$$

$$f(n) \rightarrow \mathcal{T}C \text{ to divide \& conquer}$$

## # Solving of Revenue Recd:

- ② Recursive tree method
  - ③ Masters method
  - ④ Change of variable method
  - ⑤ Generating function method

$$\text{# Substitution Method:}$$

$$\textcircled{1} \quad T(n) = \begin{cases} T(n/2) + n^2 & n > 1 \\ 0 & n = 1 \end{cases}$$

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

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

$$= \frac{1}{4} \left[ \frac{\pi^2}{3} + \frac{\pi^2}{2} + \frac{\pi^2}{2} + \frac{\pi^2}{2} \right]$$

$$= T^k T \left( \frac{v_1}{2} + \frac{v_2}{2^{k-1}} \right) + \dots + \frac{v_n}{2^k} = T^k \left[ \frac{v_1}{2} + \frac{v_2}{2} + \dots + \frac{v_n}{2} \right] = T^k \left[ \frac{v_1}{2} + \frac{v_2}{2} + \dots + \frac{v_n}{2} + 1 \right] - 1$$

$$\frac{n}{a} = \alpha \Rightarrow \frac{n}{\alpha} = a^k \Rightarrow k = \log_a\left(\frac{n}{\alpha}\right) \xrightarrow{a=1} k = \log n$$

$$= \frac{\log(\frac{1}{\alpha})}{\log(1-\alpha)} \cdot \alpha + \frac{\alpha^2}{3} \left[ \left( \frac{\alpha}{1-\alpha} \right)^{\log(\frac{1}{\alpha})} - 1 \right]$$

$$= \log_{\frac{1}{a}} T(1) + \frac{1}{\log_a b} \left[ \left(\frac{1}{a}\right)^{\log_a b} - 1 \right] =$$

$$= \Theta(n^{\log_2 7}) = \Theta(n^{2.81})$$

$$\begin{aligned}
T(n) &= 2^{\lceil \log_2 n \rceil} + n \log_2 n \\
&= 2^{\lceil \log_2 n \rceil} + \frac{n}{2} \left[ \log_2 \left( \frac{n}{2} \right) \right] + n \log_2 n \\
&= 2^{\lceil \log_2 n \rceil} + \frac{n}{2} \left[ \log_2 \left( \frac{n}{2} \right) + \frac{n}{2} \log_2 \left( \frac{n}{2} \right) \right] + n \log_2 n \\
&= 2^{\lceil \log_2 n \rceil} + \frac{n}{2} \left[ \log_2 \left( \frac{n}{2} \right) + n \log_2 \left( \frac{n}{2} \right) + n \log_2 n \right] + n \log_2 n \\
&= 2^{\lceil \log_2 n \rceil} + \frac{n}{2} \left[ \log_2 \left( \frac{n}{2} \right) + n \log_2 \left( \frac{n}{2} \right) + n \log_2 \left( \frac{n}{2} \right) \right] + n \log_2 n \\
&= 2^{\lceil \log_2 n \rceil} + \frac{n}{2} \left[ \log_2 \left( \frac{n}{2} \right) + n \log_2 \left( \frac{n}{2} \right) + n \log_2 \left( \frac{n}{2} \right) \right] + n \log_2 n \\
&= 2^{\lceil \log_2 n \rceil} + \frac{n}{2} \left[ \log_2 \left( \frac{n}{2} \right) + n \log_2 \left( \frac{n}{2} \right) + n \log_2 \left( \frac{n}{2} \right) \right] + n \log_2 n
\end{aligned}$$

$$T(n) = c \cdot n + n \lceil \log(n) \rceil = O(n \cdot \log(n))$$

$$T(n) \approx 2T\left(\frac{n}{2}\right) + n \log_2 n$$

一  
二  
三

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log_2 n$$

$$= 2^{2^T} \left( \frac{n}{2} \right) + \frac{n \log_2 n}{2} - \frac{n}{2} + n \log_2 n = 2^{2^T} \left( \frac{n}{2} \right) + \frac{n \log_2 \left( \frac{n}{2} \right)}{2} + n \log_2 \left( \frac{n}{2} \right)$$

$$= 2^2 \left[ 2^T \left( \frac{5}{2} \right) \right] + \frac{5}{2} \left[ \log_2 \left( \frac{2^3}{2} \right) \right] + \frac{5}{2} \left[ \log_2 \left( \frac{2^3}{2} \right) \right] + n \log_2 n$$

$$= 2^{2^3} T\left(\frac{n}{2}\right) + \frac{1}{2} n \log_2\left(\frac{n}{2}\right) + n \log_2\left(\frac{n}{2}\right) + n \log_2 n$$

$$= \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \dots$$

$$= \frac{1}{2} \left[ \log_2 \left( \frac{1}{\epsilon} \right) + \log_2 \left( \frac{1}{\delta} \right) \right] + \frac{1}{2} \left[ \log_2 \left( \frac{1}{\epsilon} \right) + \log_2 \left( \frac{1}{\delta} \right) \right]$$

$$= \frac{1}{2} \left( \log_{\frac{1}{2}} x + \log_{\frac{1}{2}} y \right)$$

1.  $\frac{1}{2} \times 10^3$   $\text{kg/m}^3$

$$\left[ u^{(2)} + \dots + u^{(n)} \right] = f$$

$$c_{n+1} = \frac{1}{\lambda} \left( u_n - \log(u_n) + 1 \right)$$

$$③ T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + \frac{n}{\log n} & ; n > 1 \\ c & ; n = 1 \end{cases} \quad \Theta(n \log \log n)$$

$$④ T(n) = \begin{cases} \sqrt{n} T(\sqrt{n}) + n & ; n > 2 \\ c & ; n \leq 2 \end{cases}$$

$$\begin{aligned} T(n) &= n^{1/2} T(n^{1/2}) + n \\ &= n^{1/2} [n^{1/2} T(n^{1/2}) + n^{1/2}] + n \\ &= n^{1/2} T(n^{1/2}) + 2n \\ &\geq n^{1/2} [n^{1/2} T(n^{1/2}) + n^{1/2}] + 2n \\ &= n^{1/2} T(n^{1/2}) + 3n \\ &= n^{1/2} T(n^{1/2}) + k \cdot n \\ T(n) &= n^{\frac{1}{2}k+1} \cdot T(n^{\frac{1}{2}k}) + k \cdot n \end{aligned}$$

$$n^{\frac{1}{2}k+1} = 2 \Rightarrow 2^k = \log_2 n \Rightarrow k = \log \log_2 n$$

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

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

$$T(n) = c \cdot \frac{n}{2} + n \log \log n = \Theta(n \log \log n)$$

# Master's Theorem:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad [a \neq b \text{ const.}]$$

Case I: if  $f(n) = O(n^{\log_b a - \epsilon})$  [ $\epsilon > 0$  real no.]  
then,  $T(n) = \Theta(n^{\log_b a})$

Case II: if  $f(n) = \Theta(n^{\log_b a})$   
then,  $T(n) = \Theta(n^{\log_b a} \cdot \log n)$

Case III: if  $f(n) = \Omega(n^{\log_b a + \epsilon})$  [ $\epsilon > 0$  real no.]  
then,  $T(n) = \Theta(f(n))$

E.g.:  $T(n) = 7T\left(\frac{n}{2}\right) + n^2$

Case I.  $f(n) = n^2 = O(n^{\log_2 7 - \epsilon}) \Rightarrow T(n) = \Theta(n^{\log_2 7})$   
 $n^2 \leq n^{2.01 - \epsilon}$

Case II.  $T(n) = 4T\left(\frac{n}{2}\right) + n^2$

Case I.  $n^2 = O(n^{\log_2 4 - \epsilon}) \quad \left| \begin{array}{l} \text{Case II. } n^2 = \Theta(n^{\log_2 4}) \\ n^2 \leq O(n^{2-\epsilon}) \times \end{array} \right. \quad T(n) = \Theta(n^2 \log_2 n)$

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

Case I.  $n^2 = O(n^{\log_2 3 - \epsilon}) \quad \left| \begin{array}{l} \text{Case 2: } n^2 = \Theta(n^{\log_2 3}) \\ n^2 \leq n^{1.69} \times \end{array} \right. \quad \left| \begin{array}{l} \text{Case 3: } n^2 = \Omega(n^{\log_2 3 + \epsilon}) \\ n^2 \geq n^{1.69+\epsilon} \checkmark \end{array} \right. \quad T(n) = \Theta(n^2)$

①  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \text{const.} \Leftrightarrow f(n) \approx g(n) \text{ asympt. equal.}$

②  $\frac{f(n)}{g(n)} = \text{const. or log func.} \Leftrightarrow f(n) \text{ & } g(n) \text{ are polynomially equal}$

$f(n)$	$\cdot g(n)$	Asy comp.	Poly. comp.
① $2n^2$	$n^2$	$f(n) = g(n)$	$f(n) = g(n)$
② $n^2$	$n^2 \log n$	$f(n) < g(n)$	$f(n) = g(n)$
③ $n^2$	$n^2/\log n$	$f(n) > g(n)$	$f(n) = g(n)$
④ $n^2$	$n^{2.1}$	$f(n) < g(n)$	$f(n) < g(n)$
⑤ $n^2(\log n)^0$	$n^{2.5}$	$f(n) < g(n)$	$f(n) < g(n)$
⑥ $n^2$	$2^n$	$f(n) < g(n)$	$f(n) < g(n)$

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

Case I: If  $n^{\log_b a}$  polynomially bigger than  $f(n)$ , then,  $T(n) = \Theta(n^{\log_b a})$

Case II: if  $n^{\log_b a}$  and  $f(n)$  asymptotically equal, then,  $T(n) = \Theta(n^{\log_b a} \cdot (\log n))$  depth recursion

Case III: if  $f(n)$  polynomially bigger than  $n^{\log_b a}$ , then,  $T(n) = \Theta(f(n))$

E.g.  $T(n) = 3T(n/2) + n^2$   
 $f(n) = n^2 \quad n^{\log_2 3} = n^{1.59}$

$\Theta(n^2)$ .

$T(n) = aT(n/b) + f(n)$	$n^{\log_b a}$	$f(n)$	$\Theta(\cdot)$
① $T(n) = 2T(n/2) + \sqrt{n}$	$n^{\log_2 2} = n^1$	$n^{1/2}$	$\Theta(n)$
② $T(n) = T(n/2) + \sqrt{n}$	$n^{\log_2 1} = 1$	$n^{1/2}$	$\Theta(\sqrt{n})$
③ $T(n) = 2T(n/3) + n^3$	$n^{\log_3 2} = n^{2/3}$	$n^3$	
④ $T(n) = 2T(n/3) + n^3$	$n^{\log_3 2} = n^3$	$n^3$	$\Theta(n^3 \cdot \log_3 n)$
⑤ $T(n) = T(n/3) + C$	$n^{\log_3 1} = 1$	$C$	$\Theta(\log_3 n)$
⑥ $T(n) = nT(n/2) + n \log n$	$n^{\log_2 4} = n^2$	$n \log n$	$\Theta(n^2)$
⑦ $T(n) = 4T(n/2) + n^3 \log n$	$n^{\log_2 4} = n^2$	$n^3 \log n$	$\Theta(n^3 \log n)$

Q.  $T(n) = 2T(n/2) + n \log n$   
 $f(n) = n \log n$  [  $f(n) & n^{\log_b a}$  polynomially equal but not asymptotically equal.]  
 $n^{\log_2 2} = n$  [  $n^{\log_2 2} = n$  asymptotically equal.]  
 $T(n) = n(\log n)^2$  [Master Theorem failed]

Q.  $T(n) = 2T(n/2) + \frac{n}{\log n}$   
 $T(n) = 4T(n/2) + n^2 \log \log n$  [Master Theorem failed]

$T(n) = 4T(n/2) + \frac{n^2}{(\log n)^{10}}$   
 $\checkmark T(n) = 16T(n/2) + n^4(\log n)^{10} = n^4(\log n)^{10}$   
 $\checkmark T(n) = T(n/2) + \log n = n(\log n)^2$

\* Modified Master's Theorem:

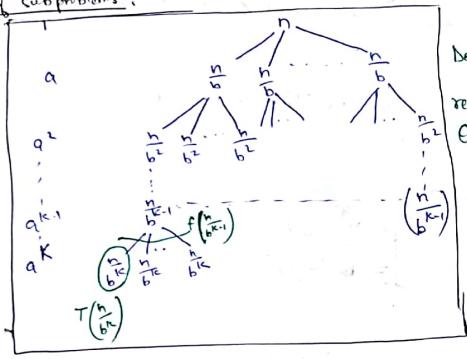
Case 2:  $f(n) = \Theta(n^{\log_b a} \cdot (\log n)^k)$  constant

$$T(n) = \Theta(n^{\log_b a} \cdot (\log n)^{k+1})$$

# Recursive Tree Method: 26/08/2017

$$T(n) = \alpha T\left(\frac{n}{b}\right) + f(n)$$

# of subproblems:

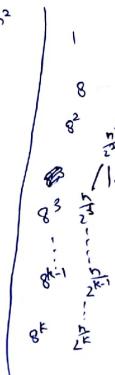


$$T(n) = \alpha^k T\left(\frac{n}{b^k}\right) + \sum_{i=0}^{k-1} \alpha^i f\left(\frac{n}{b^i}\right)$$

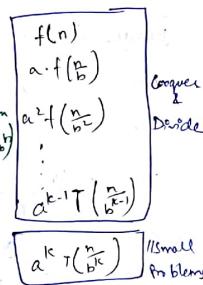
$$\begin{aligned} \because \frac{n}{b^k} = 1 \Rightarrow n = b^k \Rightarrow k = \log_b n \\ T(n) = \alpha^{\log_b n} T(1) + \sum_{i=0}^{\log_b n - 1} \alpha^i f\left(\frac{n}{b^i}\right) \end{aligned}$$

$$T(n) = n^{\log_b \alpha} \cdot c + \sum_{i=0}^{\log_b n - 1} \alpha^i f\left(\frac{n}{b^i}\right)$$

$$\begin{aligned} \text{e.g. } T(n) &= 8T\left(\frac{n}{2}\right) + n^2 \quad a=8; b=2; f(n)=n^2 \\ T(n) &= 8T\left(\frac{n}{2}\right) + n^2 [1+2+2^2+\dots+2^{k-1}] \\ \frac{n}{2} = 1 \Rightarrow n = 2^k \Rightarrow k &= \log_2 n \\ &= 8^{\log_2 n} T(1) + n^2 [n-1] \\ &= n^{\log_2 8} \cdot c + n^2 \cdot n^2 \\ T(n) &= cn^3 + n^3 - n^2 \end{aligned}$$

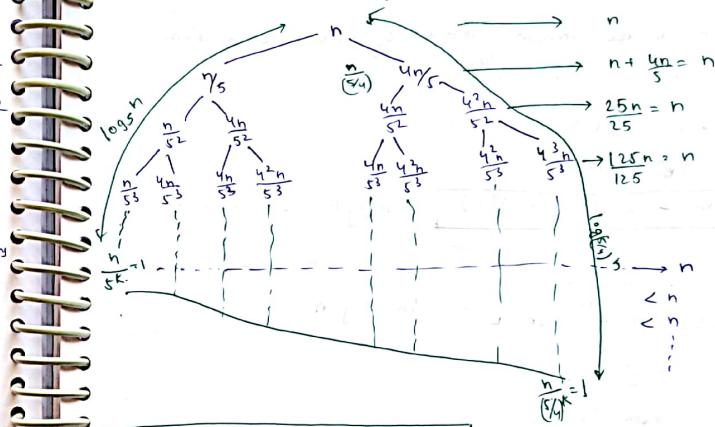


D & Time



$$\text{e.g. } T(n) = \begin{cases} T\left(\frac{n}{b}\right) + T\left(\frac{4n}{5}\right) + n & n > 1 \\ a & n = 1 \end{cases}$$

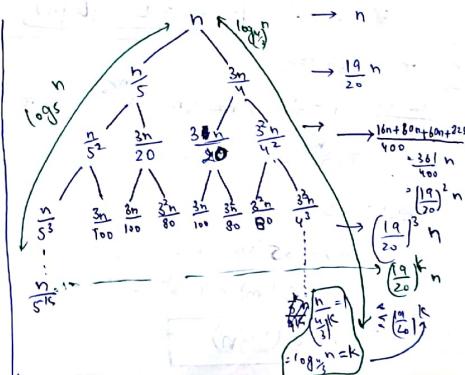
Too complex to solve using substitution method.



$$T(n) = (\log_{3/4} n) \cdot n = \Theta(n \log n)$$

$$\text{e.g. } T(n) = \begin{cases} T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right) + n & n > 1 \\ a & n = 1 \end{cases}$$

$$\begin{aligned} T(n) &= n \left[ 1 + \frac{19}{20} + \left(\frac{19}{20}\right)^2 + \dots + \left(\frac{19}{20}\right)^k \right] \\ &= n \left[ \frac{1 - \left(\frac{19}{20}\right)^{k+1}}{1 - \frac{19}{20}} \right] \\ &= 20n \left[ 1 - \left(\frac{19}{20}\right)^{\log_{19/20} n + 1} \right] \\ &= 20n \left[ n - n \left(\frac{19}{20}\right)^{\log_{19/20} n + 1} \right] \\ &\approx 20n \\ T(n) &= \Theta(n) \end{aligned}$$



### # Change of Variables:

If recurrence relation

$$T(n) = aT(\sqrt{b}n) + f(n)$$

Convert into master theorem format of recurrence relation.

$$\text{Assume: } n = b^m \rightarrow \sqrt{b}n = b^{m/2}$$

$$T(b^m) = aT(b^{m/2}) + g(m)$$

$$\text{Assume: } T(b^m) = S(m)$$

$$S(m) = aS(m/2) + g(m)$$

Apply Master's theorem.

$$\text{Eq. } T(n) = T(\sqrt{n}) + c$$

$$\text{Assume: } n = 2^m \Rightarrow \sqrt{n} = 2^{m/2} = m = \log_2 n$$

$$T(2^m) = T(2^{m/2}) + c$$

$$\text{Assume } T(2^m) = S(m)$$

$$S(m) = S(m/2) + c$$

$$\text{Assume: } n^{\log_b a} = n^{\log_2 3} = 2^m = 1$$

$$S(m) = \Theta(\log_2 m)$$

$$S(m) = T(2^m) = T(n) = \Theta(\log \log n)$$

$$\text{Q. } T(n) = 3T(\sqrt[3]{n}) + c$$

$$\text{Assume: } n = 3^m \Rightarrow \sqrt[3]{n} = 3^{m/3} \Rightarrow m = \log_3 n$$

$$T(3^m) = 3T(3^{m/3}) + c$$

$$\text{Assume: } S(m) = T(3^m)$$

$$S(m) = 3S(m/3) + c$$

$$S(m) = \Theta(m) = T(3^m) = T(n)$$

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

$$\text{Q. } T(n) = 5T(\sqrt[5]{n}) + \log_5 n$$

$$\text{Assume: } n = 5^m \Rightarrow \sqrt[5]{n} = 5^{m/5} \Rightarrow m = \log_5 n$$

$$T(5^m) = 5T(5^{m/5}) + m$$

$$S(m) = T(5^m)$$

$$\Rightarrow 5S(m/5) + m = \boxed{m \cdot \log_5 m}$$

$$T(n) = \Theta(\log_5 n \log_5 \log_5 n)$$

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

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

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

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

Should we use generating function method  
[Discrete Maths]

n<sup>th</sup> Fib. Num.

$$\text{fib}(n) = \begin{cases} 1, & n=0 \\ 1, & n=1 \\ \text{fib}(n-1) + \text{fib}(n-2), & n>1 \end{cases}$$

Solve this rec. rel<sup>n</sup> using  
generating function method.

$$T(n) = 2T(n-1) + \boxed{T(1)=1}$$

$$T(n) = 2T(n-1) + 1 \quad \boxed{T(0)=1}$$

$$T(n) = T(n-1) + n \quad \boxed{T(1)=1}$$

$$T(n) = 2T(n-1) + n \quad \boxed{T(0)=1}$$

using substitution.



## # Recursive Binary Search :

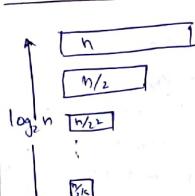
```

Algo RBsearch (l, h)
{ // a[l...h] array elements
  if (l == h) llsmall (p)
  { if (a[l] == a [h])
    return (l)
  else
    return (-1)
  }
  else
  {
    m = (l+h)/2
    if (x < a[m]) T( $\frac{n}{2}$ )
      RBsearch(l, m-1)
    else if (x > a[m]) T( $\frac{n}{2}$ )
      RBsearch(m+1, h);
    else
      return (m);
  }
}
  
```

# SC. of B.S. :  $\Theta(\log n)$  // stack space.  
// excluded input array space.

\* Because TCs of recursive and non recursive binary searches are same, non-recursive binary search is preferred because no stack space required and no overhead of function calls.

### worst case TC:



WC TC Recurrence relation of B.S.

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

Avg case:  $= \Theta(\log n)$

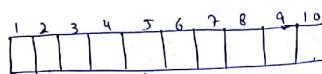
Best case:  $x$  in middle.  
 $T(n) = O(1)$

### # Problems:

① If array of 10 elements in sorted order.

Average # of element comp. to search element using binary search.

- Ⓐ 2.9 Ⓑ 3 Ⓒ 4 Ⓓ 2.



$$\text{Avg.} = \frac{29}{10} = 2.9$$

index	i	no. of comp.
1	3	3
2	2	2
3	3	3
4	4	4
5	1	1
6	4	4
7	3	3
8	2	2
9	3	3
10	4	4
Total		29

② Modify binary search algorithm such that algorithm should return position of first occurrence of searching element whose worst case TC should not exceed "logn".

low = 1; high = n

while (low ≤ high)

{ mid = (low + high) / 2

if (x < a[mid])

high = mid - 1

else if (x > a[mid])

low = mid + 1

else if (a[mid] == a[mid - 1])

high = mid - 1;

else

return (mid)

$$\text{WC AC TC} = \Theta(\log n)$$

③ Leading element of array is element which repeats more than  $n/2$  times.

(i) What is TC to find first leading element exists or not in array of  $n$  sorted elements.

- Ⓐ  $\Theta(\log n)$  Ⓑ  $\Theta(n)$  Ⓒ  $\Theta(n \log n)$  Ⓓ  $\Theta(n^2)$

(i) What is TC to find leading element exists or not in array of n elements whose values range [0 ... n]

- Ⓐ  $\Theta(n \log n)$  Ⓑ  $\Theta(n)$  Ⓒ  $\Theta(n)$  Ⓓ  $\Theta(n^2)$

(ii) What is TC to find leading element exists or not in array of n elements.

- Ⓐ  $\Theta(n \log n)$  Ⓑ  $\Theta(n)$  Ⓒ  $\Theta(n \log n)$  Ⓓ  $\Theta(n^2)$

(i) Sorted array:



Middle element must be leading element if leading element exists.

$x = a[mid]$   
i = call firstbinsearch (low, high, x)

If ( $a[i + \frac{y}{2}] = x$ )  
Then leading element present  
else  
Then leading element not exists.

TC =  $\Theta(\log n)$

(ii) Unsorted array of n elements.  
Range of elements [0 ... n]

for ( $i=0$ ;  $i < n$ ;  $i++$ )  
count [ $i$ ] = 0;

for ( $i=1$ ;  $i < n$ ;  $i++$ )  
count [ $a[i]$ ] = count [ $a[i]$ ] + 1;

for ( $i=0$ ;  $i < n$ ;  $i++$ )  
{ if ( $count[i] \geq \frac{n}{2}$ )  
then return (true)}

}

a	1	2	3	4	5	6	7	8	9	10
	3	9	4	10	3	9	9	3	9	
count	0	1	2	3	4	5	6	7	8	9

TC =  $\Theta(n)$

(iii) Array of n elements:

① sort array using best sorting algo. ]  $\Theta(n \log n)$

② find leading element in sorted array ]  $\Theta(n \log n)$

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

(4) Assume size of array unknown [Infinite]. First few elements in sorted order remaining elements are " $\infty$ ". Write binary search to search " $x$ " in array or to find last position of defined number.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	- - -
2	3	6	8	10	12	14	16	18	20	22	24	26	28	30	$\infty$	$\infty$	$\infty$	- - -

```
i=1
while (a[i] != 0)
    i = i + 1
    low = i // 2 ; high = i
    call BS (low, high)
    // for n elements
```

TC =  $\Theta(\log n)$

(5) Array of ~~n~~ distinct sorted elements which can be -ve or +ve. Find any element " $x$ " whose index " $x$ " present in array or not?

i = a[0] Index(x) = x

1	2	3	4	5
-5	1	3	2	1

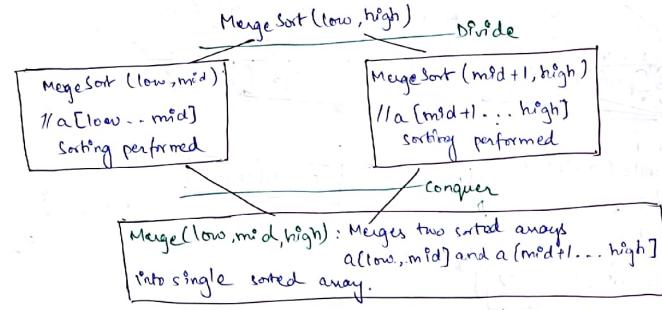
5	6	7
8	10	15

worst case TC =  $\Theta(n)$

1	2	3	4	5	6	7	8	9
-5	3	-2	0	2	3	5	8	10

```
low = 1 high = n
while (low <= high)
    mid = (low + high) / 2
    --if (mid < a[mid])
        high = mid - 1;
    else if (mid > a[mid])
        low = mid + 1;
    else
        return (-1);
```

Merge Sort Algo [Fully D and C Approach]:  
a [low.... high] array of n elements.

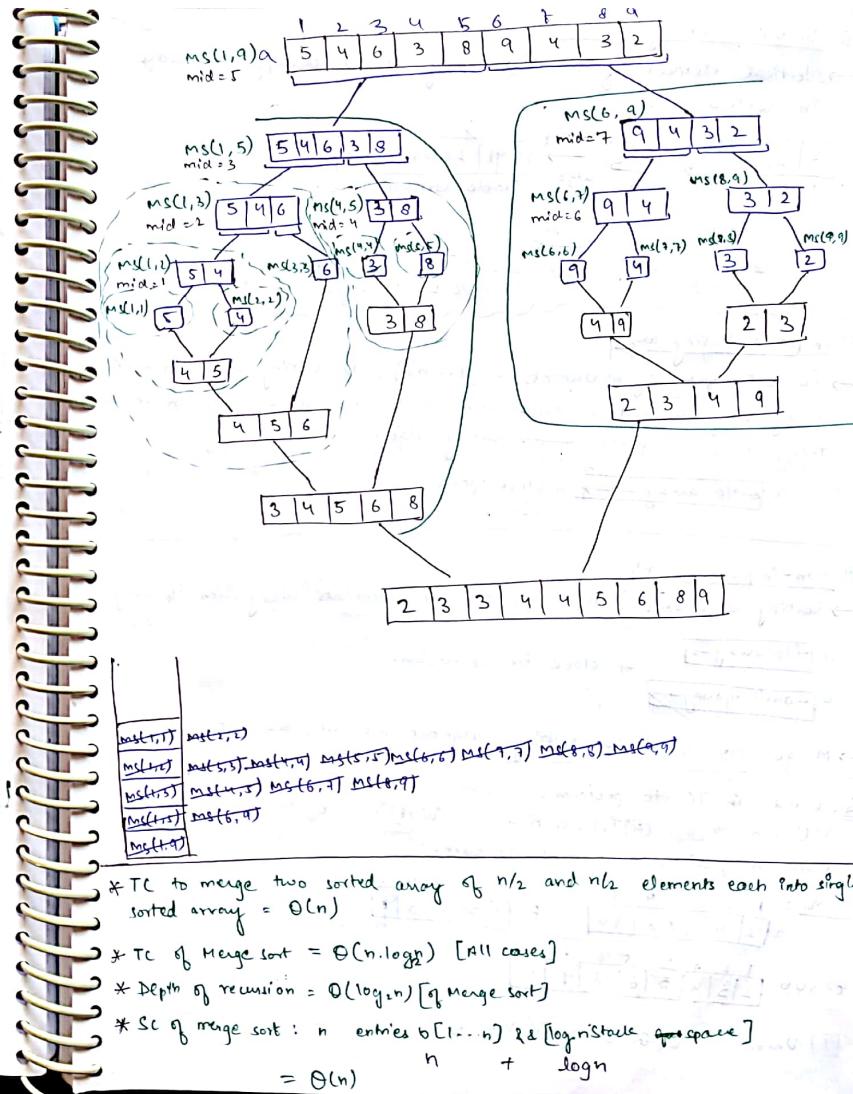


Algo MergeSort ( $low, high \rightarrow T(n)$ )  
 { If  $a[low \dots high]$  are  $n$  elements.  
   if ( $low < high$ )  
     {  $mid = (low + high)/2$   
       MergeSort ( $low, mid$ );  $\rightarrow T(n/2)$   
       MergeSort ( $mid+1, high$ );  $\rightarrow T(n/2)$   
       //conquer  
       Merge ( $low, mid, high$ );  $\Theta(n)$   
     }
 }

```

Algo Merge (low, mid, high)
{ // To merge two sorted arrays a[low..mid] and a[low..high]
// and a[mid+1 .. high] into single sorted array
    i = low; j = mid + 1; k = low
    while (i <= mid && j <= high)
        if (a[i] <= a[j])
            { b[k] = a[i]; i++; }
        else
            { b[k] = a[j]; j++; }
        k++;
}
if (i > mid)
    for (x = i; x <= high; x++)
        { b[x] = a[x]; k++; }
else
    for (x = i; x <= mid; x++)
        { b[x] = a[x]; k++; }
for (x = low; x <= high; x++)
    { a[x] = b[x]; }
}

```



### # Stable Sorting Algo:

→ Identical elements of unsorted array should occupy same order in sorted array.

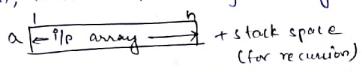
**Sorting algo**:  $\begin{bmatrix} 3 & 2 & 0 & 2 & 1 & 2 \end{bmatrix}$   
 $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$

**Stable sorting algo**:  $\begin{bmatrix} 0 & 1 & 2a & 2b & 2c & 3 \end{bmatrix}$

**Unstable sorting algo**:  $\begin{bmatrix} 0 & 1 & 2b & 2c & 2a & 3 \end{bmatrix}$

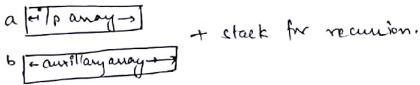
### # In place sorting algo:

→ To sort array of  $n$  elements ( $a[1...n]$ ); If sorting algo uses input array only, to sort elements and may use stack space for recursion then, it is in place algo. sorting algo.



### # Non-in place sorting :

→ Sorting algo uses extra array to sort elements along with  $\theta(n)$  array.



→ Merge sort is stable sorting algo but not in place sorting algo.

Q. What is TC to perform:

- (i) Union      (ii) Intersection      (iii) Minus of two sorted arrays,  $n$  and  $m$ , distinct elements in each.

Solution:  $a = [1, 2, 3, 4, 5]$        $b = [1, 2, 3, 4, 5]$

$c = a \cup b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$

(i) Union  $\Rightarrow$  ~~TC~~  $TC = \Theta(n+m)$

i=1      j=1      k=1

while ( $i \leq n$  &  $j \leq m$ )

  if ( $a[i] < b[j]$ ) {  $c[k] = a[i]; i++; k++;$  }

  else if ( $a[i] > b[j]$ ) {  $c[k] = b[j]; j++; k++;$  }

  else {  $c[k] = a[i]; i++; j++; k++;$  }

  k++;

  if ( $i > n$ ) for ( $x=j$ ;  $x \leq m$ ;  $x++$ )

    {  $c[k] = b[x]; k++;$  }

  else for ( $x=i$ ;  $x \leq n$ ;  $x++$ )

    {  $c[x] = a[x]; k++;$  }

(ii) Intersection:

i=1      j=1      k=1

while ( $i \leq n$  &  $j \leq m$ )

  if ( $a[i] < b[j]$ ) {  $i++; k++;$  }

  else if ( $a[i] > b[j]$ ) {  $j++; k++;$  }

  else {  $c[k] = a[i]; i++; j++; k++;$  }

  k++;

TC =  $\Theta(n+m)$

(iii) Minus:  $TC = \Theta(n+m)$

Q. How many max<sup>m</sup> element comp. required to merge two sorted arrays  $n$  &  $m$  elements each into single sorted array.

- (a)  $n + m$       (b)  $n + m + 1$       (c)  $n + m - 1$       (d)  $n + m$

$2 | 4 | 6 | 8 | 10$        $3 | 5 | 7 | 9$

$2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10$

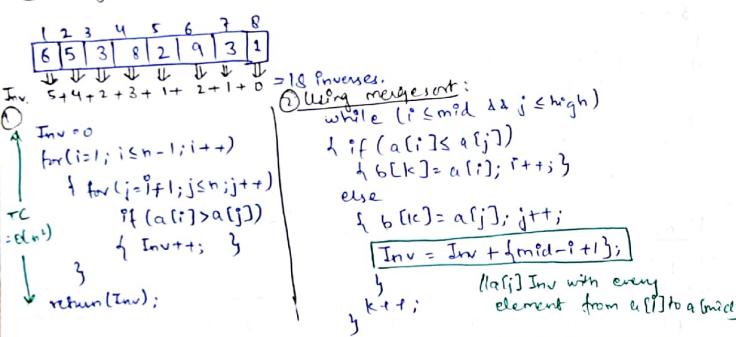
$k = n + m - 1$  placed

One comp. for each element.

=  $n + m - 1$  comparisons.

Q. Given  $a[1..n]$  array of  $n$  elements. Inverse defined as two consecutive elements of array such that  $i < j \text{ and } a[i] > a[j]$ . What is TC to find no. of inverses of array of  $n$ -elements.

- Ⓐ  $\Theta(n)$
- Ⓑ  $\Theta(n\log n)$
- Ⓒ  $\Theta(n^2)$

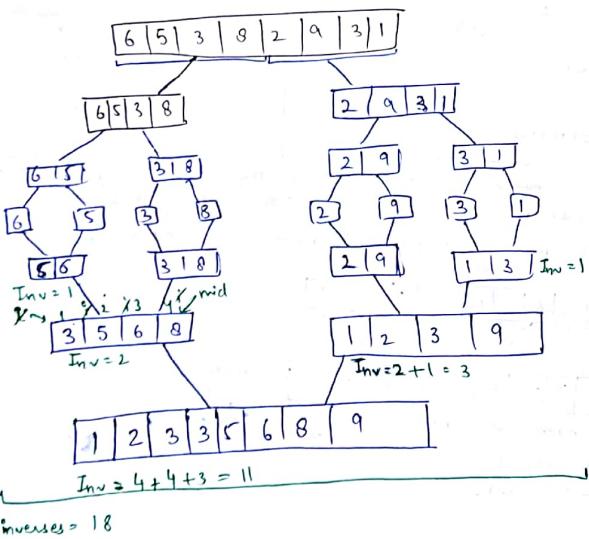


② Using mergesort:  
 while ( $i \leq \text{mid}$  &  $j \leq \text{high}$ )
 

- ↳ if ( $a[i] \leq a[j]$ )
- ↳  $b[k] = a[i]$ ;  $i++$
- else
  - ↳  $b[k] = a[j]$ ;  $j++$
  - ↳  $k++$

 $\text{Inv} = \text{Inv} + (\text{mid}-i+1)$

$\text{Inv with every element from } a[i] \text{ to } a[\text{mid}]$



$$\begin{aligned} Q. f_0 &= 1 & k=0 \\ f_1 &= 1 & k=1 \\ f_k &= f_{k-1} + f_{k-2}; & k \geq 2; \end{aligned}$$

$$\begin{array}{|c|c|c|c|c|c|c|c|} \hline k & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline f_k & 1 & 1 & 2 & 3 & 5 & 8 & 13 \\ \hline \end{array}$$

Algo MergeSort ( $f_k$   $\uparrow$ ps)

{ if ( $k \geq 1$ )

{ //Divide

MergeSort ( $f_{k-1}$   $\uparrow$ ps)

MergeSort ( $f_{k-2}$   $\uparrow$ ps)

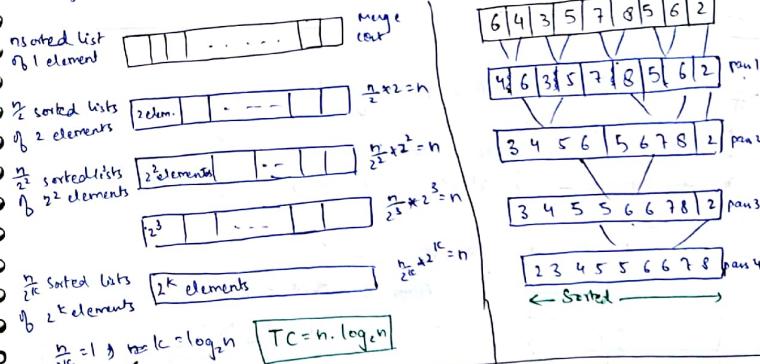
//Conquer

(Merge ( $f_{k-1}, f_{k-2}$ )) cost of merge fk.

}

#2 way Merge Sort: [straight Merge sort]

→ Initially  $a[1..n]$  consider as  $n$  sorted lists each list one element. Merge two lists until  $n$  sorted lists become single sorted list.



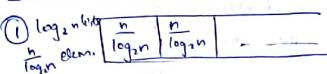
Q. Array consists  $\log_2 n$  sorted lists each list  $\frac{n}{2^k}$  elements. What is TC to merge many into single sorted list?

$$T(n) = T\left(\frac{n}{\log_2 n}\right) + n$$

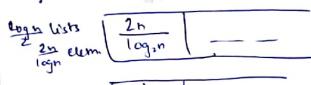
$$\log_2 n = k \Rightarrow$$

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

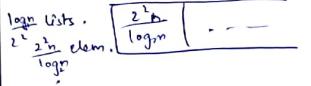
Q. Array consists of  $n$  strings each string of  $n$  characters. What's TC to sort  $n$  strings in lexicographic order?



merge count



$$\log_2 n \times \frac{n}{2} = \frac{n}{2} \log_2 n$$



$$\log_2 n \times 2^{n-k} = \log_2 n \times 2^{\log_2 n - k}$$

$$TC = n \log_2 n$$

# Quick Sort Algo : [Also partially D and C]

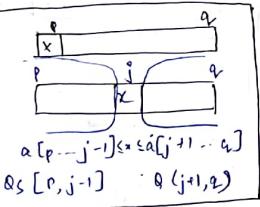
\* widely used sorting algo

\* Inplace and not stable sorting algo

\* Tony George [1970] {Turing Award 1985}

Basic Idea :

Choose pivot element and place in correct position.



Algo 'QuickSort(p,q)'

```
{
    // a[p...q] array of n elements
    if (p < q)
        {
            j = partition(a, p, q);
            QuickSort (p, j-1);
            QuickSort (j+1, q);
        }
}
```

Partition Algo Procedure:

Pivot  $\rightarrow p$        $q \leftarrow j$   $\rightarrow$  To away out of board accusif

1. Increment  $i$  until  $a[i] > x$  pivot element
2. Decrement  $j$  until  $a[j] \leq x$  bigger than all other element.
3. If  $i < j$  swap ( $a[i], a[j]$ )
4. Repeat ①②③ until  $i \geq j$
5. swap ( $a[p], a[j]$ )
6. return  $j$

1	2	3	4	5	6	7	8
35	20	40	50	30	60	15	75

1	2	3	4	5	6	7	8
30	20	15	35	50	60	40	75

Algo partition (a, p, q)  $O(n)$

```
{
    a[q+1] = +∞;
    i = p; j = q+1; x = a[p];
    do
        {
            i = i+1;
            while (a[i] < x)
                i = i+1;
            j = j-1;
            while (a[j] > x)
                j = j-1;
            if (i < j) swap (a[i], a[j]);
            swap (a[p], a[j]);
        }
    return j;
}
```

1	2	3	4	5	6	7	8	9	10
2	4	5	6	10	12	14	15	16	18

$a[i] < x$  : 1 comp  
 $a[j] > x$  :  $\frac{n}{2}$  comp.  
htl comp.

1	2	3	4	5	6	7	8	9	10
10	8	6	5	4	3	2	1	9	7

$a[i] < x$  :  $n$  comp  
 $a[j] > x$  : 1 comp  
htl comp.

unsorted

$a[i] < x$  :  $n/2$   
 $a[j] > x$  :  $\frac{n}{2} + 1$   
htl comp.

TC of partition algo increases:  $O(n^2)$

Example Quick Sort:

65	30	40	50	15	90	+∞
50	40	30	60	+∞		

65	30	40	50	15	90	+∞
40	30	+∞				

1	2	3	4	5	6	7	8
50	40	30	10	55	60	75	90

1	2	3	4	5	6	7	8
40	30	10	55	60	75	90	+∞

1	2	3	4	5	6	7	8
40	30	10	55	60	75	90	+∞

1	2	3	4	5	6	7	8
40	30	10	55	60	75	90	+∞

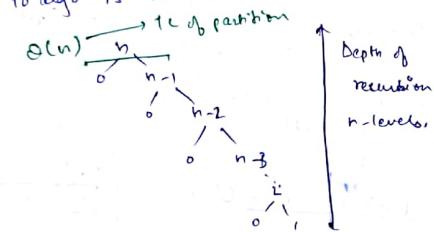
1	2	3	4	5	6	7	8
40	30	10	55	60	75	90	+∞

1	2	3	4	5	6	7	8
40	30	10	55	60	75	90	+∞

## # TC of Quicksort :

### ① Worst case TC of QS:

QS behaves as worst case if given i/p array to algo is in sorted order.

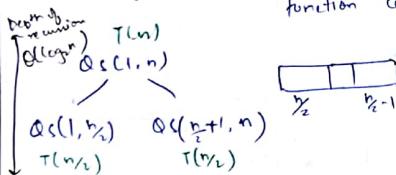


$$TC = \begin{cases} T(n-1) + n & ; n > 1 \\ 1 & ; n \leq 1 \end{cases}$$

$$TC = \Theta(n^2)$$

### ② Best case TC of Quicksort:

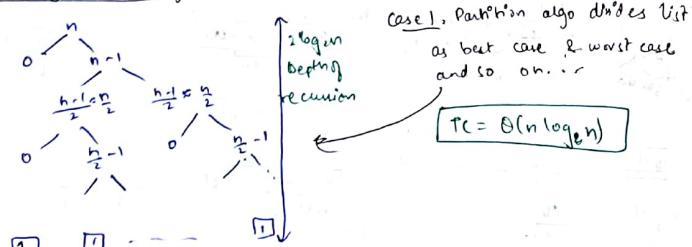
If pivot element places middle of list in every function call, then QS behaves as best case.



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

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

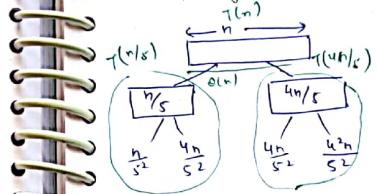
### ③ Average case TC of Quicksort: $\Theta(n \log n)$



case 1. Partition algo divides list as best case & worst case and so on... i.e.

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

(Q12) : Every time partition divides list into  $\frac{n}{5}$ s elements &  $\frac{4n}{5}$ s elements for given  $n$  elements.



Recurrence relation:

$$T(n) = \begin{cases} T\left(\frac{n}{5}\right) + T\left(\frac{4n}{5}\right) + n & ; n > 1 \\ T(n) = \Theta(n \log n) & ; n \leq 1 \end{cases}$$

### Average case recurrence relation of QS:

$$T(n) = \begin{cases} T(\alpha n) + T((1-\alpha)n) + n & ; n > 1 \\ 1 & ; n \leq 1 \end{cases}$$

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

$$TC(n) = T\left(\frac{n}{2}\right) + T\left(\frac{19n}{20}\right) + n$$

$$TC(n) = T\left(\frac{n}{1000}\right) + T\left(\frac{999n}{1000}\right) + n$$

### # Pivot Selection Methods:

① First / last element of array is pivot: Worst case  $TC = \Theta(n^2)$   
Pf: If array is in sorted order.

② Pivot is avg. of min and max of array:

8	4	6	10	12	14	Min=2 Max=14	$\Theta(n)$
8	4	6	10	12	14	Min=2 Max=14 Avg=8	

Algo QS(p,q)

if ( $p < q$ )  
 { k = pivotSet (a, p, q)  $\rightarrow \Theta(n)$   
 swap (a[k], a[q])  
 j = partition (a, p, q)  $\rightarrow \Theta(n)$   
 QS (p, j-1)  $\rightarrow T(n-2)$   
 QS (j+1, q)  $\rightarrow T(n)$

1	2	3	4	5	6	7	8	9	10
7	125	62	15	31	1000	250	50	31	
Min=1 Max=1000 Avg=500									

$$TC(n) = T(n-1) + cn = \Theta(n^2)$$

② Median element\*: Median element of array "x" such that:  
 $\frac{n}{2}$  elements  $\leq x \leq \frac{n}{2}$  elements

If median element of array is pivot element:  
 then worst case TC of QS :=  $\Omega(n\log n)$  [more than  $n\log n$ ]  
 which depends on TC to find median.

Also QS(p, q)  
 if ( $p < q$ )  
 k = median(a[p], a[q])  $\rightarrow \Omega(n)$   
 swap(a[k], a[p])  
 j = Partition(a, p, q)  $\rightarrow \Theta(n)$   
 QS(p, j-1)  $\rightarrow T(n/2)$   
 QS(j+1, q)  $\rightarrow T(n/2)$

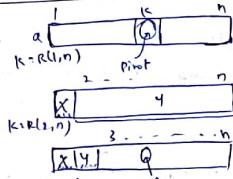
$$TC = \Omega(n\log n) \\ (\text{All cases})$$

Q. WC TC of Quick sort using median pivot selection?

- (A)  $\Omega(n^2)$  (B)  $\Omega(n)$  (C)  $\Theta(n\log n)$  (D)  $\Omega(n\log n)$

④ Random Pivot Selection:

Algo QS(p, q)  
 if ( $p < q$ )  
 k = Random(p, q)  
 swap(a[k], a[p])  
 j = position(a, p, q)  
 QS(p, j-1)  
 QS(j+1, q)



WC TC of QS

$= \Theta(n^2)$   
 // Every time randomly chooses min/max element as pivot w.r.t. unknown.

Probability of min/max element selected by Random Pivot selection is almost 0.

Q. What is WC TC of QS using Random Pivot selection?

- (A)  $\Omega(n^2)$  (B)  $\Omega(n\log n)$  (C)  $\Theta(n\log n)$  (D)  $\Theta(n)$

In pivot selection algo,  $\Theta(n)$  time to choose which is  $(\frac{n}{2})^{th}$  biggest element of array. Then what is WC TC of QS.

- (A)  $\Omega(n^2)$  (B)  $\Theta(n\log n)$  (C)  $\Theta(n)$  (D)  $\Omega(n)$

Q. If all elements of array is identical. What is TC of QS?

- (A)  $\Theta(n\log n)$  (B)  $\Theta(n^2)$  (C)  $\Theta(n)$  (D)  $\Theta(n)$

a [1 2 3 4 5 6 7]  
 [9 9 9 9 9 9 9]  
 // TC =  $\Theta(n\log n)$  or  $\Theta(n^2)$   
 Depends on implementation of partition.  
 Our algo =  $\Theta(n\log n)$

Q. List<sub>1</sub> = 1, 2, 3, ..., n

List<sub>2</sub> = n, n-1, n-2, ..., 1

if c<sub>1</sub> & c<sub>2</sub> comp. required to sort <sub>1</sub> & <sub>2</sub>, & list<sub>1</sub> & list<sub>2</sub> in according order using quick sort algo. Which is true?

- (A) c<sub>1</sub> < c<sub>2</sub> (B) c<sub>1</sub> > c<sub>2</sub> (C) c<sub>1</sub> = c<sub>2</sub> (D) c<sub>1</sub> = c<sub>2</sub> + n log n

Q. Write algo to find k<sup>th</sup> smallest element in array of n distinct element.

Method 1.

① Sort array using sorting algo  $\Theta(n\log n)$  All cases  
 ② return a[k]

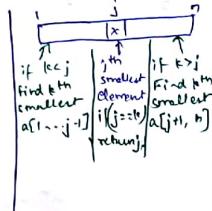
Method 2.

① Find pos of min from a[1] to a[n] & swap with a[1]  
 TC =  $\Theta(n)$   
 ② Find pos of min(a[2] to a[n])  
 swap with a[2] & k times  
 TC =  $\Theta(n^2)$   
 ALG  $\Theta(n^2)$

Method 3.

Best solution: using partition algo.

Algo k<sup>th</sup> smallest (p, q, k)  
 if (p = q)  
 if (k = p)  
 return (a[p])  
 else return (false)  
 else  
 partition(a, p, q)  
 if (k < j)  
 k<sup>th</sup> smallest (p, j-1, k)  
 else if (k > j)  
 k<sup>th</sup> smallest (j+1, q, k)  
 else return (a[j])



best case TC:  
 $\Theta(n)$   
 One function call  
 to terminate algo  
 if  $j \geq k$  after one  
 partition.

Average Case TC:  
 $T(n) = T(\frac{n}{2}) + n.c$   
 $T(n) = T(\frac{n}{2}) + n.c$   
 $T(n) = T(\frac{n}{2}) + c.n$   
 Assume "d.n",  $(1-d)n$   
 $d, 1-d$  is partition sizes

Worst Case TC:  
 $T(n) = T(n-c) + n$

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

\* Every time partition algo divides  
 $c.d(n-c)$  for slip away  
 with smallest element at ~~n~~  $n$ .

# Max-Min Algo:  
 →  $a[1...n]$  array of  $n$  elements. Find max & min element of array.

straight Max Min Algo:

```
Algo Max-Min(a,n)
{ Max = min = a[1];
for (i=2; i<n; i++)
    if (a[i] > max)
        max = a[i];
    else if (a[i] < min)
        min = a[i];
}
return (max, min);
```

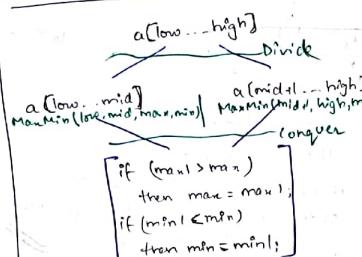
$\alpha [1\ 2\ 3\ 4\ 5]$   
 $[4\ 5\ 2\ 3\ 6]$

\* min comp. required to find  
 Max & Min =  $(n-1)$  comp.

\* Max comp. required to find  
 Max & Min =  $2(n-1) - 2n-2$  comp.

$$\frac{(n-1)}{2} + \frac{(n-1)}{2} \times 2 = \frac{3n-3}{2}$$

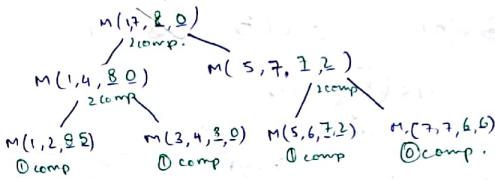
D And C Max-Min Algo:



Algo Max-Min (low, high, max, min)

```
{ if (low == high)
    n=1
    max = min = a[low];
else if (low == high - 1)
    { if (a[low] > a[high])
        { max = a[high];
        min = a[low];
        }
    else
        { max = a[low];
        min = a[high];
        }
    }
else
    { mid = (low + high) / 2
    max = min = Max-Min (low, mid, max, min)
    min = Max-Min (mid+1, high, max, min)
    if (max1 > max) max = max1;
    if (min1 < min) min = min1;
    }
}
return (max, min)
```

$\alpha [1\ 2\ 3\ 4\ 5\ 6\ 7]$   
 $[8\ 5\ 3\ 0\ 2\ 7\ 6]$



$T(n)$ : # of comparisons to find max-min using D And C.

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

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + 2 \\ &= 2\{2T(\frac{n}{4}) + 2\} + 2 \\ &= 2^2T(\frac{n}{8}) + 2^2 + 2 \\ &= 2^3T(\frac{n}{16}) + 2^3 + 2^2 + 2 \\ &\vdots \\ &= 2^{k-1}\left(\frac{n}{2^k}\right) + \left[2^k + \dots + 2^2 + 2\right] \\ &= 2^kT\left(\frac{n}{2^k}\right) + \left[2^k - 1\right] \\ &\downarrow \\ &n = 2^{k+1} \Rightarrow 2^k = \frac{n}{2} \Rightarrow k = \log_2\left(\frac{n}{2}\right) \\ &= \frac{n}{2}T(2) + 2\left(\frac{n}{2} - 1\right) \\ &\Rightarrow \frac{n}{2} \cdot 1 + n - 2 = \boxed{\frac{3n-2}{2} = T(n)} \end{aligned}$$

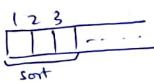
Max-Min Algo	Min Comp.	Avg. Comp.	Max Comp.
straight	$\frac{3n-3}{2}$	$\frac{3n-15}{2}$	$2n-2$
D & C	$\frac{3n-2}{2}$	$\frac{3n-2}{2}$	$\frac{3n-2}{2}$

Q. Min element comp. to find max-min of array of  $n$  elements.  
 $\Theta(n-1)$   $\Theta(2n-2)$   $\Theta(\frac{3n-2}{2})$   $\Theta(n)$

Q. Max element comp. to find max and min of array of  $n$  elements.  
 $\Theta(n-1)$   $\Theta(2n-2)$   $\Theta(\frac{3n-2}{2})$   $\Theta(n)$

Q. What is TC reqd. to return element from array of n distinct elements, which is neither max nor min?

(A)  $\Theta(\text{length})$    (B)  $\Theta(n)$    (C)  $\Theta(n^2)$    (D)  $\Theta(1)$

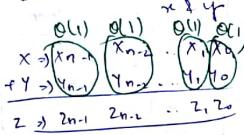


$a[1] < a[2] < a[3]$   
neither max nor min

# Integer Multiplication Algo:

- (A) Straight integer multiplication
- (B) D&C integer multiplication
- (C) Strassen's D&C int mult.

\* Integer Addition: x, y are two n-bit integers. TC required to add x + y :



(A) straight integer multiplication:

$$x=25 \quad y=5 \quad z=0$$

① if x is odd, then  $z = z+y = 5$

$$x = \left\lfloor \frac{x}{2} \right\rfloor = 12 \quad y = 2^4 = 16$$

② if x is even, then no add

$$x = \left\lfloor \frac{x}{2} \right\rfloor = 6 \quad y = 2^4 = 16$$

③ if x is even, then no add

$$x = \left\lfloor \frac{x}{2} \right\rfloor = 3 \quad y = 2^4 = 16$$

④ if x is odd,  $z = z+y = 45$

$$x = \left\lfloor \frac{x}{2} \right\rfloor = 1 \quad y = 2^4 = 16$$

⑤ If x is odd,  $z = z+y = 125$

$$x = \left\lfloor \frac{x}{2} \right\rfloor = 0 \quad y = 2^4 = 16$$

Algo IntMul( $X, Y$ )  
{ // X, Y two n-bit integers

$$Z = 0$$

while ( $X > 0$ ) O(1)

{ if ( $(X \& 1) == 1$ )

$$\{ Z = Z + Y; \text{left shift } O(1)$$

  X =  $\left\lfloor \frac{X}{2} \right\rfloor$ ; // Right shift O(1)

  Y =  $Y \ll 1$ ; // Left shift O(1)

return (Z);

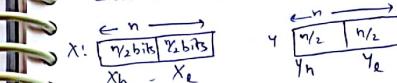
}

# of bits required to represent int x is  $\lceil \log_2 x \rceil$  bits.

TC =  $\Theta(n^2)$  # of bit operations.

F = # D&C int multiplication:

→  $X, Y$  are n bit numbers. If  $n > 1$  divide  $x \times y$  into two subint each with  $n/2$  bits.



$$x \cdot z = (X_h \cdot 2^{n/2} + X_e) * (Y_h \cdot 2^{n/2} + Y_e)$$

$$x \cdot z = X_h \cdot Y_h \cdot 2^n + (X_h \cdot Y_L + X_L \cdot Y_h) \cdot 2^{n/2} + X_L \cdot Y_L$$

add ← left shift      left shift

$$\begin{array}{rcl} \overbrace{\hspace{1cm}}^n & & \\ (01)_2 & (010)_2 & = 26 \\ X_h & X_e & \\ (01)_2 & X_e = 010 & \\ & & = 2 \end{array}$$

$$\begin{array}{rcl} X & = & X_h \cdot 2^3 + X_L \\ & = & 3 \cdot 2^3 + 2 \\ & = & 24 + 2 = 26 \end{array}$$

To multiply two n-bit int.  ~~$x \cdot y$~~

4 subint mul of  $n/2$  bits

3  $\frac{n}{2}$  left shift opr.

3 int additions

Algo IntMul ( $X, Y, n$ )  $\Rightarrow T(n)$

{ if ( $n \leq 1$ ) // small (P) ] a  
  return ( $X \times Y$ )

else

+ // Divide

$X_h, X_e$  are two int sub int of  $X$

$Y_h, Y_e$  are two sub int of  $Y$

P = IntMul ( $X_h, Y_h, n/2$ )

Q = IntMul ( $X_h, Y_e, n/2$ )

R = IntMul ( $X_e, Y_h, n/2$ )

S = IntMul ( $X_e, Y_e, n/2$ )

// Conquer

return ( $P \cdot 2^n + (Q+R) \cdot 2^{n/2} + S$ )  $\Rightarrow C \cdot n$

D&C recurrence relation:

$$T(n) = \begin{cases} 4T(n/2) + C \cdot n & n > 1 \\ a & n \leq 1 \end{cases}$$

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

### # Strassen's D&C:

$$x \begin{bmatrix} & \\ X_h & X_L \end{bmatrix} \quad y \begin{bmatrix} & \\ Y_h & Y_L \end{bmatrix}$$

$$X = X_h \cdot 2^{\frac{n}{2}} + X_L$$

$$Y = Y_h \cdot 2^{\frac{n}{2}} + Y_L$$

$$\begin{aligned} P &= X_h \cdot Y_h \\ Q &= X_L \cdot Y_L \\ R &= (X_h + X_L) \cdot (Y_h + Y_L) \\ X \cdot Y &= P \cdot 2^n + [R - P - Q] \cdot 2^{\frac{n}{2}} + Q \end{aligned}$$

Strassen's Int Mult. D&C Recurrence relation:

$$\begin{aligned} T(n) &= \begin{cases} 3T(n/2) + c \cdot n & ; n > 1 \\ c & ; n \leq 1 \end{cases} \\ &= \Theta(n^{\log_2 3}) \\ &= \Theta(n^{1.59}) \end{aligned}$$

### # Matrix Multiplication Algo:

- (a) Straight Matrix Multiplication
- (b) D&C Matrix Multiplication
- (c) Strassen's D&C Matrix multiplication

#### \* Matrix Addition:

$$C_{n \times n} = A_{n \times n} + B_{n \times n}$$

$$\begin{aligned} C_{11} &= A_{11} + B_{11} \\ C_{12} &= A_{11} + B_{12} \\ \vdots &\vdots \\ C_{n1} &= A_{11} + B_{n1} \\ C_{n2} &= A_{12} + B_{12} + \dots + B_{n2} \end{aligned}$$

$$\Theta(n^2) \text{ for } (i,1; i \leq n; i++) \\ \text{for } (j=1; j \leq n; j++) \\ C[i,j] = A[i,j] + B[i,j]$$

$$\# \text{Straight Matrix Multiplication:}$$

$$\begin{bmatrix} a_{11} & \dots & a_{1n} \\ a_{21} & & \\ \vdots & & \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & \dots & b_{1p} \\ b_{21} & & \\ \vdots & & \\ b_{n1} & \dots & b_{np} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{21} & \dots & c_{n1} \\ c_{12} & & & \\ \vdots & & & \\ c_{1p} & c_{2p} & \dots & c_{np} \end{bmatrix}$$

$$c_{11} = a_{11} \cdot b_{11} + a_{12} \cdot b_{21} + \dots + a_{1n} \cdot b_{n1}$$

n element multiplications for each element of result matrix.

for (i=1; i < n; i++) → P

for (j=1; j < p; j++) → R

$$\{ C[i,j] = 0 \}$$

$$\text{for } (k=1; k < n; k++) \rightarrow Q$$

$$C[i,j] = C[i,j] + C[i,k] * B[k,j]$$

}

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

or

$$\Theta(p \cdot q \cdot r)$$

Q. If  $A_{pxq}$ ,  $B_{qxr}$  are two matrices. How many element multiplication reqd. to multiply matrices  $A$  &  $B$ ?  $P \cdot Q \cdot R$  [Total element mul.] Cost of matrix multiplication.

### # D&C Matrix Multiplication:

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

→ if  $n > 2$  divide each matrix into 4 submatrices with  $\frac{n}{2} \times \frac{n}{2}$  order.

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$$

$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$$

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

Algo - D&C (A, B, n)

```

if (n ≤ 2)
    return (A · B)
else
    // Divide
    Divide A & B into 4 submatrices each
    P = D&C (A11, B11, n/2)
    Q = D&C (A12, B12, n/2)
    R = D&C (A21, B21, n/2)
    S = D&C (A22, B22, n/2)
    t = D&C (A11 + A12, B11 + B12, n/2)
    u = D&C (A21 + A22, B21 + B22, n/2)
    v = D&C (A11 - A12, B11 - B12, n/2)
    w = D&C (A21 - A22, B21 + B22, n/2)

    C11 = P + Q
    C12 = Q + S
    C21 = t + U
    C22 = V + W

    return [C11 C12; C21 C22]
}

```

D&C Matrix mul. Recurrence relation

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

$\boxed{T(n) = \Theta(n^3)}$

# Strassen's Matrix Multiplication: (Most efficient alg)

→ Divide each sub matrix into 4 sub matrices:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}_{n \times n} \quad \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}_{n \times n}$$

\* To multiply matrices of  $n \times n$   
 → 7 submatrix mult. of  $n/2 \times n/2$   
 → 8 matrix add/sub

$$P = (A_{11} + A_{12}) \cdot (B_{11} + B_{22})$$

$$Q = (A_{21} + B_{22}) \cdot B_{11}$$

$$R = A_{11} \cdot (A_{12} - A_{22})$$

$$S = A_{21} \cdot (B_{21} - B_{11})$$

$$t = (A_{11} + A_{12}) \cdot B_{22}$$

$$u = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$$

$$v = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

$$C_{11} = P + S + t + v \quad | \quad C_{21} = Q + t + s$$

$$C_{12} = r + t \quad | \quad C_{22} = p + r - q + u$$

To multiply matrices of  $n \times n$

$$T(n) = \begin{cases} 7T(n/2) + C \cdot n & ; n > 2 \\ a & ; n \leq 2 \end{cases}$$

$$= \Theta(n^{\log_2 7})$$

$\boxed{T(n) = \Theta(n^{2.81})}$

25log(2017) DS Topic

- Graph and Tree representation
- Priority Queue [Min heap / Max heap]
- Set algorithms [Union / Find Algo]

### Tree

- Single rooted (Starting point). Every tree operation starts from root.
- Always connected.
- Always acyclic.
- Always directed.
- Tree with  $n$  vertices must be exactly  $(n-1)$  edges

### Graph

- Any vertex can be used as starting point.
- May / may not connected.
- May / may not acyclic.
- May / may not directed.
- [Simple graph with  $n$  vertices can be at least 0 Edge [Null graph]. atmost  $\frac{n(n-1)}{2}$  Edges [complete connected graph]]

\* Simple Graph: No self loops.  
 No parallel edges.

$\textcircled{A} \rightarrow \textcircled{B}$  Not simple graph.

Q. How many simple undirected graph possible with  $n$  vertices?

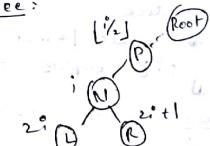
$$n=4 \rightarrow \frac{n(n-1)}{2} = 6$$

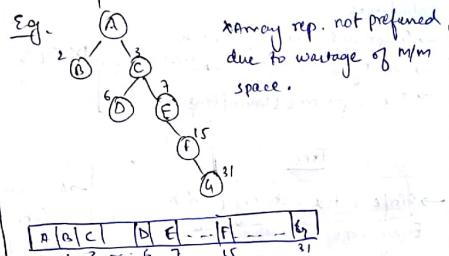
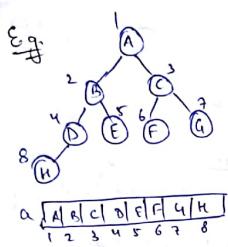
$$e=6 \quad \left| \begin{array}{l} \textcircled{O} \textcircled{O} \\ \textcircled{O} \textcircled{O} \\ \textcircled{O} \textcircled{O} \end{array} \right. + c_1 + c_2 + c_3 + \dots + c_6 = 2^6$$

# Tree Representation: ① Using array  
 ② Using linked list

\* Array representation to store binary tree:

Root node index = 1  
 If Node N away index i  
 Then left child of N is  $2^i$  index  
 right child of N is  $(2^i + 1)$  index  
 parent of N is  $\lfloor \frac{i}{2} \rfloor$  index





\* Array rep. not preferred due to wastage of memory space.

Q. Complete k-ary, every internal node exactly k child nodes.

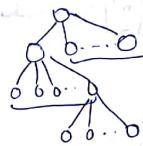
How many # of leaf nodes with n internal nodes.

$$n \cdot k$$

$$n(k-1)+1$$

$$(k-1)k+1$$

$$n(k-1)$$



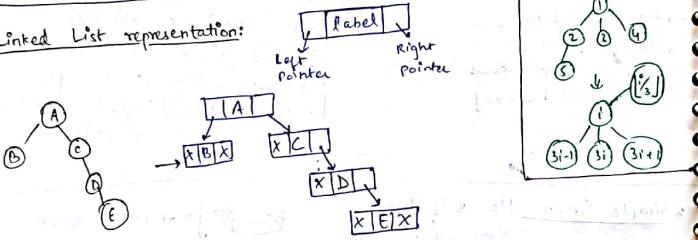
$$0 \rightarrow 1$$

$$1 \rightarrow 2$$

$$2 \rightarrow 2k-1$$

$$3 \rightarrow 3k-2$$

\* Linked List representation:



→ To store binary tree of n nodes:  $2n$  pointers

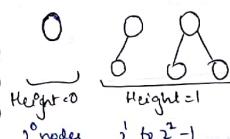
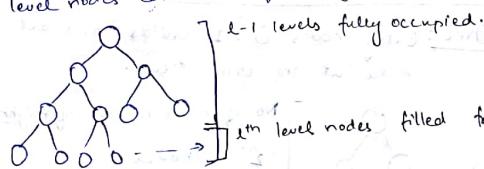
[ $n+1$  null pointers]  
2 [ $n-1$  non-null pointers]

\* If node indexes are continuous integer from 1 to n, then array representation preferred. [No overhead of pointers].

\* Complete Binary Tree:

→ Complete binary tree of  $2^h$  levels must be  $(2^h-1)$  levels fully occupied.

→  $h^{th}$  level nodes can add left to right.



\* Complete binary tree of height "h":  
 $(2^h \text{ to } 2^{h+1}-1)$  nodes

\* Array representation preferred to store complete binary tree & FBST

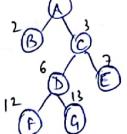
→ Complete binary tree of "n" nodes:  $\lceil \frac{n}{2} \rceil + \# \text{ of internal nodes}$

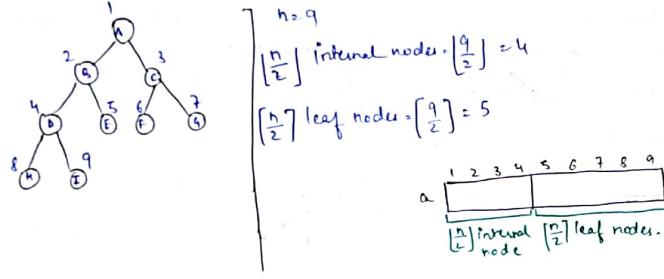
or

FBST  $\lceil \frac{n}{2} \rceil + \# \text{ of leaf nodes.}$

\* Strict binary tree: Every node of tree must have 0 or 2 children.

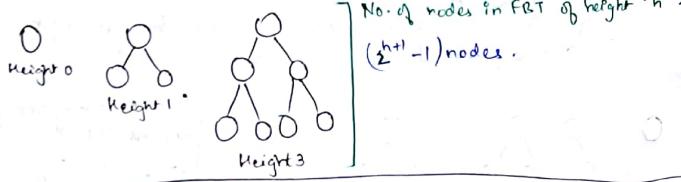
\* Linked list representation preferred to store strict binary tree.





\* Height of CBT for "n" nodes:  
 of  
 nodes  
 max  
 nodes  
 $n = 2^h \Rightarrow h = \log_2 n$   
 $n = 2^{h+1} - 1 \Rightarrow h = \log_2(n+1)-1$

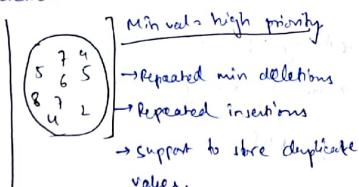
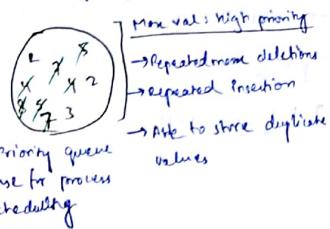
\* Full Binary Tree: Every nodes 0 & 2 childs and all leaf nodes at the same level.



### # Priority Queue Data Structure:

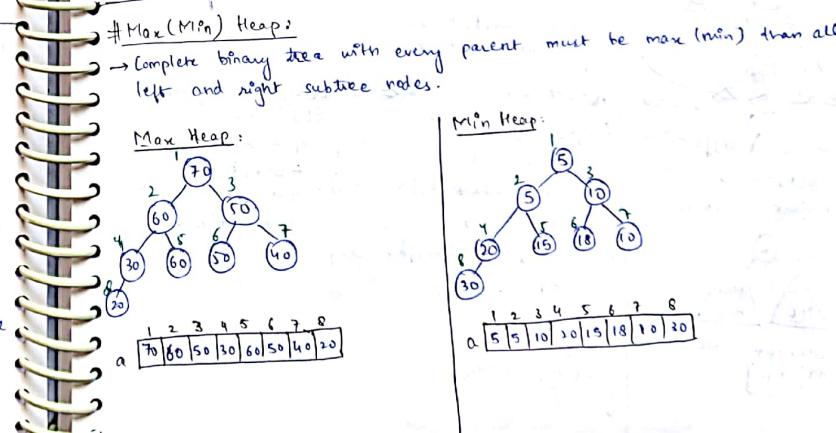
→ DS which should support efficient way to perform:

- repeated min/max element deletion
- Repeated insertions
- May required to store duplicate element.

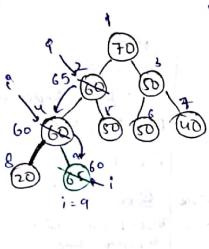


Q. Algo	① Build list for n elements [Assume elements should insert one after another and may not be distinct]	② Repeat Min del. over list n times	Balanced Binary Search Tree	min heap
Sorted array //Descending	$n \times O(1)$ $= O(n^2)$	$1+2+3+\dots+(n-1)$ $= O(n)$	$n \times O(1)$ $= O(n^2)$	$O(n \log n)$ $= O(n \log n)$
Unsorted array	$n \times O(1)$ $= O(n)$	$(n-1)+(n-2)+\dots+2+1$ $= O(n^2)$	$n \times O(1)$ $= O(n^2)$	$O(n \log n)$ $= O(n \log n)$
Sorted linked list //Asc.	$n \times O(1)$ $= O(n^2)$	$n \times O(1)$ $= O(n^2)$	$n \times O(1)$ $= O(n^2)$	$O(n \log n)$ $= O(n \log n)$
Unsorted linked list	$n \times O(1)$ $= O(n)$	$n \times O(1)$ $= O(n)$	$n \times O(1)$ $= O(n^2)$	$O(n \log n)$ $= O(n \log n)$

Note:  
 \* Balanced BST best DS for searching element.  
 ① Insert / delete  
 ③ discard duplicate elements.



\* Insertion into Maxheap:  $a[1 \dots n-1]$  array elements already in max heap.  
Insert element  $x$  into Max heap.

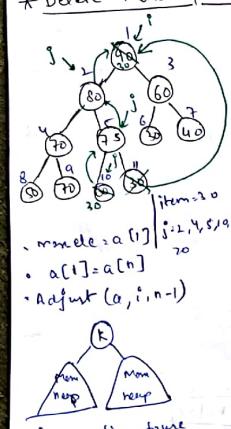


```
Algo InsertMaxHeap(a, n, x)
{ i=n;
  item=x;
  while (i >= 1 && a[i/2] < item)
  {
    a[i]=a[i/2];
    i=i/2;
  }
  a[i]=item;
}
```

→ Comp. and shift operations insert element into max heap:  $\Theta(\log n)$   
( $\log n$  height)

→ TC to insert element into max heap:  $\Theta(\log n)$  in worst & avg. case.  
 $\Theta(1)$ , in best case.

\* Delete Max from Max heap:  $\Theta(\log n)$



```
Algo DelMaxHeap(a, n)  $\Rightarrow$  TC to delete max from heap.
{ MaxEle = a[1];           of n elements:  $\Theta(\log n) \rightarrow$  WL & AC
  a[1] = a[n];
  Adjust(a, 1, n-1);
  return(MaxEle)

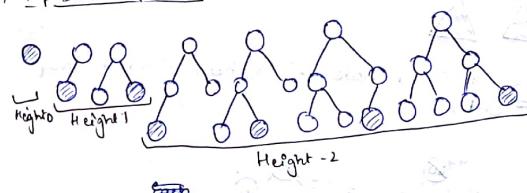
Algo Adjust(a, i, n)  $\Leftarrow$  TC =  $\Theta(\log n)$  w/ 8 Avg.操,
  0(1) in AC.
{ j=2*i ; item = a[i];
  while (j <= n)
  {
    if (j < n) & a[j] < a[j+1]) .adjust of element "x"
    { j=j+1; }                      at height "n".
    if (j a[j] < item) {break; }
    a[j/2] = a[j];
    j=2*j;
  }
  a[i/2] = item;
}
```

[Assumption done  
adjust h]

\* Building of heap:  $2 + \log_2(n+1)$ :

- ① Top Down approach: [To build max heap of elements [key] should enter one by one after another]  $\Theta(n)$
- ② Bottom Up approach: [To build max heap if all elements are available]

\* Top Down Approach:



Each insertion of height h in [n comp. & n shift]

Height	No. of nodes	Cost per insertion [comp. & shift]
0	1	0
1	2	1
2	2^2	2
3	2^3	3
4	2^4	4
5	2^5	5
6	2^6	6
7	2^7	7
8	2^8	8
9	2^9	9
10	2^10	10

Cost of building max heap in top down approach.

$$\begin{aligned}
 T(h) &= 2 \times 1 + 2^2 \times 2 + 2^3 \times 3 + \dots + 2^h \times h \\
 -2T(h) &= 2^2 \times 1 + 2^3 \times 2 + 2^4 \times 3 + \dots + 2^{h+1} \times h \\
 -T(h) &= 2 \times 1 + 2^2 \times 2 + 2^3 \times 3 + \dots + 2^h \times h \\
 -T(h) &= 2 \times 1 + 2^2 + 2^3 + 2^4 + \dots + 2^h + 2^{h+1} \times h \\
 -T(h) &= (2 + 2^2 + 2^3 + 2^4 + \dots + 2^h) + 2^{h+1} \times h \\
 T(h) &= 2^{h+1} \cdot h - (2 + 2^2 + 2^3 + 2^4 + \dots + 2^h) \\
 &\Rightarrow 2^{h+1} \cdot h - 2^{h+1} + 2 = 2^{h+1} \cdot h - (2^{h+1} - 1) + 1 \Rightarrow (n+1)(\log_2(n+1)) - n+1
 \end{aligned}$$

$$\begin{aligned}
 2^{2^n-1} &= 2^{n+1}-2 \\
 n \text{ elements in max heap} \\
 2^{n+1}-1 &= n \\
 2^{n+1} &= n+1 \\
 n &= \log_2(n+1)-1 \\
 n+1 &= \log_2(n+1)+1
 \end{aligned}$$

$T(n) = n \log_2 n$
* Bottom up approach:
* TC to build maxheap of height "h" using bottom up.
<img alt="Diagram showing a max heap of height h being built from an array [9

TC of Heap sort =  $\Theta(n \log n)$  [All cases]

→ In place sorting algo.

→ Not stable sorting algo.

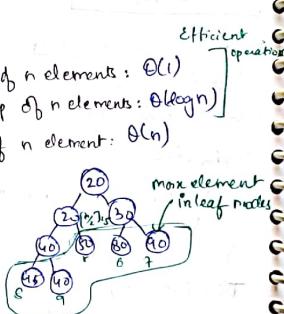
#### Heap Operations:

① TC to find min element in min heap of n elements:  $\Theta(1)$

② TC to delete min element from min heap of n elements:  $\Theta(\log n)$

③ TC to find max element in min heap of n elements:  $\Theta(n)$

$\max = a[\lceil \frac{n}{2} \rceil]$  Not efficient  
for ( $i = \lceil \frac{n}{2} \rceil + 1; i \leq n; i++$ )  
{ if ( $a[i] > \max$ )  
     $\max = a[i];$   
}



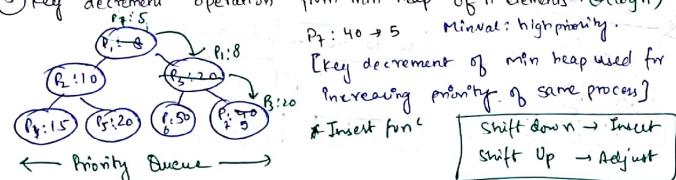
④ TC to delete max from min heap of n elements:

→ Find max pos in heap:  $\Theta(n)$

→ del max:  $\Theta(\log n)$

→  $\Theta(n)$

⑤ Key decrement operation from min heap of n elements:  $\Theta(\log n)$



⑥ Key increment operation TC from min heap of n elements:  $\Theta(\log n)$

→ use adjust func.

⑦ find k<sup>th</sup> smallest element in min heap of n distinct elements:

Method-1: ① delete min from min heap (k-1) times  $\Theta(k \log n)$   
② Root is k<sup>th</sup> minimum

#### Method 2:

1<sup>st</sup> min {a[1]}

2<sup>nd</sup> min {a[2], a[3]}

3<sup>rd</sup> min {a[4], a[5], a[6]}

4<sup>th</sup> min {a[7], a[8], a[9], a[10]}

⋮

k<sup>th</sup> min {k elements}

comp.

1

2

3

⋮

k-1

find k <sup>th</sup> min of min heap	if k is constant	if k is terms of n [k = O(n)]
Method 1 $\Theta(k \log n)$	$\Theta(\log n)$ efficient	$\Theta(n \log n)$ efficient
Method 2 $\Theta(1)$		$\Theta(n^2)$

Q. what is TC to find 7<sup>th</sup> min in min heap of n distinct elements?

A.  $\Theta(\log n)$     B.  $\Theta(n \log n)$     C.  $\Theta(1)$     D.  $\Theta(n^2)$

Q. what is TC to find 10<sup>th</sup> min in min heap of n distinct elements?

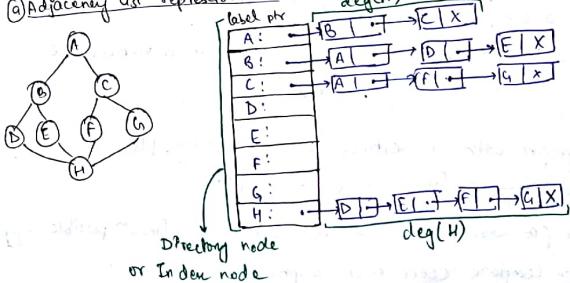
A.  $\Theta(\log n)$     B.  $\Theta(n \log n)$     C.  $\Theta(1)$     D.  $\Theta(n^2)$

#### Graph Representation:

① Adjacency List representation. [using linked lists]

② Adjacency matrix representation. [using 2-D array]

#### ③ Adjacency Mat rep:





## # Single source shortest path algorithms F.P.P 30/08/2017

① Dijkstra's Algorithm [Greedy]

② Bellman Ford algorithm [Not greedy]

### ① Dijkstra's Algo:

Graph (n,e) n: no. of vertices  
e: no. of edges

cost [1...n, 1...n] Edge costs

$cost[i,j] = \begin{cases} \text{Edge cost if } (i,j) \in \text{Edge} \\ \infty \text{ if } (i,j) \notin \text{Edge} \end{cases}$

s: source vertex

→ Compute shortest path from source (s) vertex to every vertex of graph. Determine dist [1...n] s.t. dist [i] shortest path cost from s to i.

Dijkstra's Procedure

// Initialisation

1. dist [1...n] = +∞;  
prev [1...n] = -1;

dist[s] = 0

list q of all vertices which are not completed relaxation

Hall n vertices in list

// Relaxation Procedure

2. choose vertex (u) from list which is min dist which is not relaxed before.

u is set of adj. of vertex (u)

for all v sat

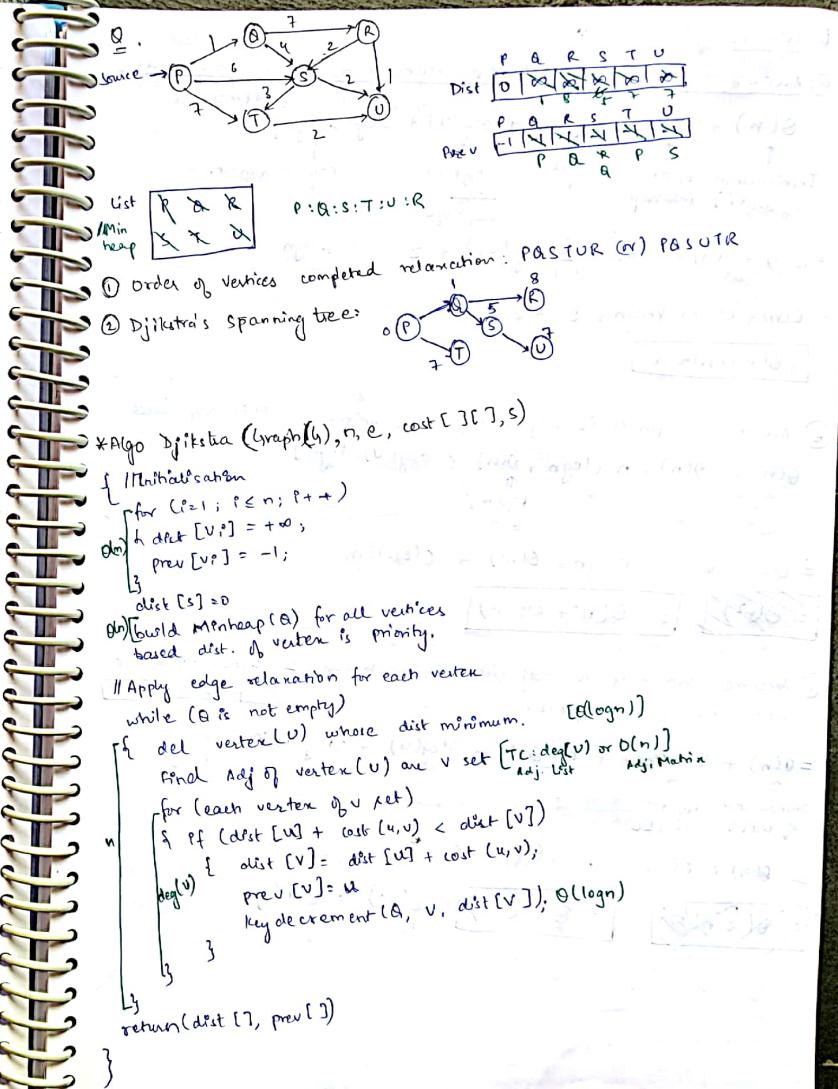
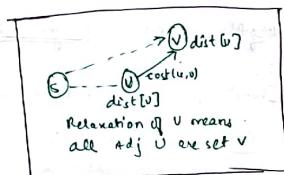
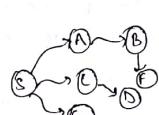
{ if (dist [u] + cost (u,v) < dist [v])

{ dist [v] = dist [u] + cost (u,v)

prev [v] = u

}

3. Repeat ② for all vertices



### # TC of Dijkstra's Algo:

(a) Assume graph  $G$  is Adj. list representation & Min heap used.

$$\Theta(n) + \Theta(n) + n [\log n + \deg(u) + \deg(v). \log n]$$

Initialization Build min key decrementation

$$= \Theta(n) + \Theta(n) + n \log n + \sum_{i=1}^n \deg(U_i) + \sum_{i=1}^n \deg(U_i). \log n$$

$$= \Theta(n) + \Theta(n) + n \log n + e + e \cdot \log n$$

$$= \Theta(n+e) \cdot \log n$$

(b) Assume graph  $G$  is Adj. matrix representation & Min heap used.

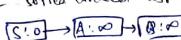
$$\Theta(n) + \Theta(n) + n [\log n + \Theta(n) + \deg(v). \log n]$$

Find Adj<sup>T</sup>

$$= \Theta(n) + \Theta(n) + n \log n + \Theta(n^2) + \Theta(e) \cdot \log n$$

$$= \Theta(n^2 + e \log n)$$

(c) Assume graph  $G$  is adj. list representation & sorted linked list used for priority queue.



$$= \Theta(n) + \Theta(n) + n [\Theta(1) + \deg(u) + \deg(v).n]$$

Initialization Build sorted LL

$$= \Theta(n) + \Theta(n) + n + \Theta(e) + e \cdot n$$

$$= \Theta(e \cdot n) \quad [ \Omega(n^2) \text{ to } O(n^3) ]$$

~~=  $\Theta(e \cdot n^2)$~~

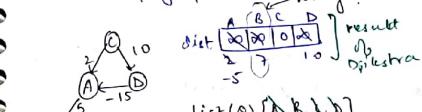
(d) Assume graph  $G$  in adj. list and unsorted array used as priority queue.

$$= \Theta(n) + \Theta(1) + n [n + \deg(u) + \deg(v). \Theta(1)]$$

$$= \Theta(n^2 + e) = \Theta(n^2)$$

### # Limitations of Dijkstra's Algo:

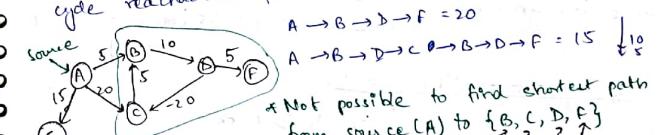
① Dijkstra's algo may fail to find shortest path from source to other vertices if graph consists -ve edges. [Assume no -ve cycle]



list(A) {A, B, C, D}

- ① vertex  $v$  completed relaxation
- ②  $\text{dist}[v]$  depends on  $\text{dist}[u]$
- ③ because of relaxation vertex  $(x)$ ,  $\text{dist}[u]$  reduces then only Dijkstra fails.

② Dijkstra's algo fails to detect negative cycle if neg negative cycle reachable from source.



\* Not possible to find shortest path from source (A) to {B, C, D, E, F} [Algo should detect -ve cycle]

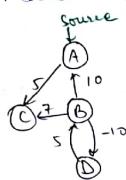
\* Not possible using Dijkstra's

not possible to compute shortest path from A to BCD.

|| (cycle detection required)

## ② Bellman Ford Algo: [Not greedy method]

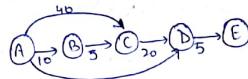
- Compute shortest path cost from source correctly even -ve edge exists. [Assume no -ve cycle]
- Detect negative cycle if negative cycle reachable from source.



\* Bellman Ford algo also doesn't detect -ve cycle.

\* If no negative cycle exists, shortest path from source(s) to vertex(A) should consist at most  $(n-1)$  edges. [n-vertices]

③ More  $(n-1)$  edges → in shortest path



A to E  $\Rightarrow$  4 edges b/w shortest path from A to E.



A - B - C - D - E  $\rightarrow$  40 ( $n-1$ ) edges  
A - B - C - B - C - D - E - F  $\rightarrow$  35 [more than  $(n-1)$  edges]

\* Procedure of Dijkstra's: Bellman Ford:

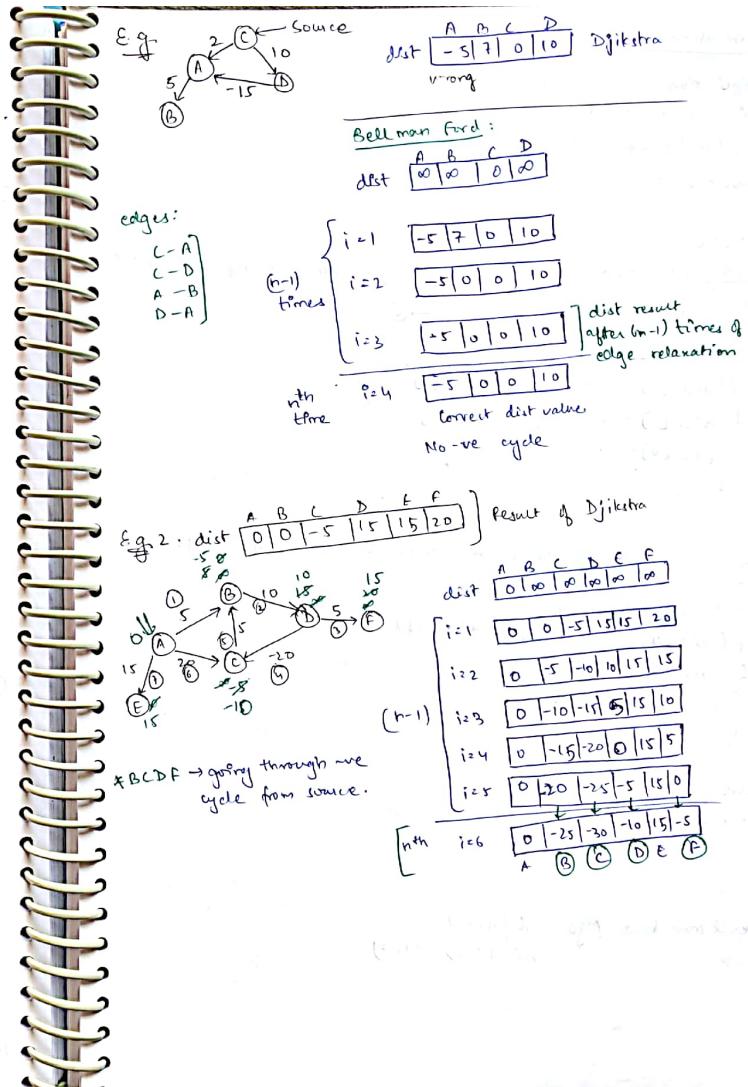
① Initialise  
dist [v<sub>i</sub>] =  $\infty$   
prev [v<sub>i</sub>] = -1 for all vertices  
dist [s] = 0

② Apply edge relaxation for all edges of graph.

22  
Repeat  $(n-1)$  times.

③ Cycle Detection

Apply edge relaxation  $n^{th}$  time:  
If any vertex dist[v] reduces  
then negative cycle going through source.



Algo BellmanFord (n, e, cost[], s, E[])

{ //Initialization  
for ( $i=1; i \leq n; i++$ )  
{ dist[v $_i$ ] = + $\infty$   
prev[v $_i$ ] = -1  
}  
}

dist[s] = 0  
//Apply edge relaxation for each edge of graph and repeat (n-1) times.

for ( $i=1; i \leq n-1; i++$ ),  $\textcircled{0} \rightarrow \textcircled{1}$   
{ for all edges of graph(u,v)  
{ if (dist[u] + cost[u,v] < dist[v])  
{ dist[v] = dist[u] + cost[u,v]  
prev[v] = u;  
}  
}

//Testing of negative cycle reachable from source exists or not.

for (all edges of graph(u,v))  
{ if (dist[u] + cost[u,v] < dist[v])  
{ //Negative cycle exists  
return (FALSE);  
}  
}  
return (dist[], prev[]);  
//No negative cycle

\* TC of Bellman Ford Algo :  $\Theta(n \cdot e)$   
WC TC  $\Rightarrow \Theta(n^3)$

# Minimal Spanning Tree: 31/08/2017

$\rightarrow$  Spanning Tree: of graph  $G(v, e)$  is  $G'$  such that  $G'(v, e')$  where  $e' \subseteq e$  and  $G'$  is connected and no cycle.

$\times n^{n-2}$  spanning trees possible in complete connected graph with  $n$ -vertices.

Complete connected 4 nodes  $\Rightarrow 4^{n-2} = 16$  spanning tree

\* Kirchhoff Theorem: To find no. of spanning trees of graph

- ① Represent graph  $G$  in adjacency matrix format.
- ② Multiply (-1) to adj. matrix
- ③ Diagonal element replace deg. of adj.(i,i)

$$\text{Cofactor of } \text{Adj}(1,1) = (-1)^{1+1} \det \begin{bmatrix} 3 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

$$= \{3(6-1) - (-1)(-2-1) + (-1)(1+3)\} \\ = 8 \text{ (STs of graph)}$$

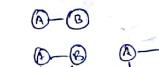
$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 0 & 1 & 1 \\ 3 & 1 & 0 & 1 \\ 4 & 0 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -1 & -1 & 0 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 \end{bmatrix}$$

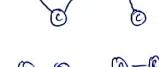
$$\begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ 0 & 1 & 1 & 2 \end{bmatrix}$$

Q. How many edge disjoint STs possible in complete connected graph of  $n$  vertices?  $[n \geq 2]$

Complete connected graph No. of edge disjoint STs



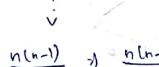
1



1



2



3



4



5



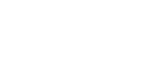
6



7



8



9



10



11



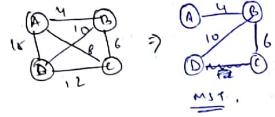
12



13

### Minimal Spanning Tree:

→ Sum of cost of edges of ST of graph is minimum.



Q. Distinct edge costs.  $\{1, 2, 3, 4, 5, 6\}$

Max cost of MST?

- (a) 6      (b) 7      (c) 9      (d) 10

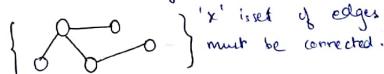


① Prim's Algo - Greedy

② Kruskal's Algo. Algorithms.

### Prim's Algorithm:

→ MST constructs edge by edge set edges which are included in MST must be always connected.



$(u, v)$  next edge of MST connected s.t.  $\{x \cup (u, v)\}$  must be connected & min inc. in sum of edge costs.

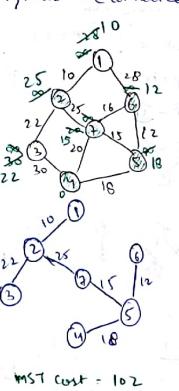
1	2	3	4	5	6	7
10	25	22	18	12	15	
25	22	18	12	15		
22	18	12	15			
18	12	15				

1	2	3	4	5	6	7
✓	✓	✓	✓	✓	✓	
✓	✓	✓	✓	✓	✓	
✓	✓	✓	✓	✓	✓	
✓	✓	✓	✓	✓	✓	

color  $[v_i] = \{w; \text{if } v_i \text{ not included into MST}\}$

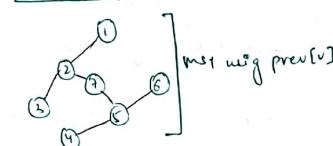
$\{B; \text{if } v_i \text{ included into MST}\}$

Minheap ( $Q$ ): All vertices based on  $dist(v)$  is priority



MST cost = 102

X 1 3 Y  
+ 1 4 X



cost MST: using  $dist[v]$

Relaxation

$v_1$   $v_2$   $v_3$   $v_4$   $v_5$   $v_6$   $v_7$

$dist[v_1] = 10$

$dist[v_2] = ?$

$dist[v_3] = ?$

$dist[v_4] = ?$

$dist[v_5] = ?$

$dist[v_6] = ?$

$dist[v_7] = ?$

if ( $dist[v] > cost(u, v)$ ) &  
color[v] = w  
dist[v] = cost(u, v)  
prev[v] = u  
prev[v] = 4

Algo Prims (Graph, n, e, cost[ ])

{ for ( $i=1; i \leq n; i++$ )

{     $dist[v_i] = \infty$

  prev[v\_i] = -1

  color[v\_i] = w

}

choose one vertex as source  $dist[s] = 0$

for (Build minheap) for all vertices based on  $dist[v_i]$  priority

while ( $Q$  not empty)

{ del vertex  $x(u)$  which min dist. from min heap.

color[u] = B // u included into MST.

v is adj. of u

for (all vertices (v) adj. of u)

{ if ( $dist[v] > cost(u, v)$  & color[v] = w)

{     $dist[v] = cost(u, v)$

  prev[v] = u

  keydel(Q, v, dist[v])

}

}

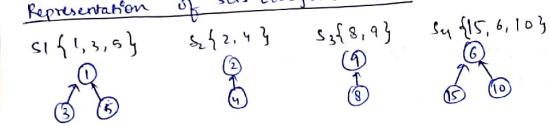
return (prev[ ], dist[ ])

### \* TC of Prim's Algo:

- (1) Assume Adj List graph & priority queue min heap:  $O(n+e).logn$
- (2) Assume Adj matrix graph & priority queue min heap:  $O(n^2 + e log n)$
- (3) Assume Adj list graph & priority queue unsorted array:  $O(n^2)$

\* SET Algo: [Used in Kruskal's algo to detect cycle in MST construction.]

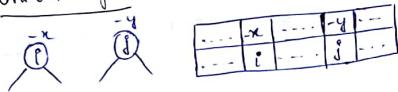
Representation of sets [disjoint set]:



P(L)	-3	2	1	2	1	-3	9	2	6	6
L	1	2	3	4	5	6	8	9	10	15

Representation of disjoint sets using array.

### \* Union Algo:



If set with root  $i$  has more elements than set with root  $j$ , then  $i$  becomes parent  $j$ .

### Algo Union ( $i, j$ )

```

{ z = p[i] + p[j] // set with root i
  if (p[i] ≤ p[j]) // set with root i has more elements.
  { p[j] = i;      -> p[i]
    p[i] = z;
  }
  else // set with root j has more elements.
  { p[i] = j;      -> p[j]
    p[j] = z;
  }
}
  
```

TC of Union =  $O(1)$

### \* Find Algo:

P(i)										
-7	4	2	4	2	3	30	-2	18		i

find( $i$ ): should return root set tree of element  $i$  present.

Find (22) = 18

Find (10) = 2

Find (4) = 2

Find (2) = 2

Algo Find ( $i$ )

{ while ( $p[i] > 0$ ) TC of find operation:  $O(\text{height of set tree})$

$i = p[i]$ ;

  return ( $i$ );

}

→ Initially  $n$  sets; each set one element

P(i)										
1	2	3	4	5	...	(n-1)	n			
i	-1	-1	-1	-1	-1	...	-1	-1		
1	2	3	4	5	...	n-1	n			

→ Min height of set tree if  $n$  sets merge into one set using union op.  $O(1) \times \text{min height}$

→ Max height of set tree if  $n$  sets merge into one set using union op.  $\text{max height} = O(n)$

→  $n$  sets, each set 2 elements. Height = 1

→  $\frac{n}{2}$  sets, each set 2 elements. Height = 2

→  $\frac{n}{4}$  sets, each set 2 elements. Height = 3

→  $\frac{n}{8}$  sets, each set 2 elements. Height = 4

→  $\frac{n}{16}$  sets, each set 2 elements. Height = 5

→  $\frac{n}{32}$  sets, each set 2 elements. Height = 6

→  $\frac{n}{64}$  sets, each set 2 elements. Height = 7

→  $\frac{n}{128}$  sets, each set 2 elements. Height = 8

→  $\frac{n}{256}$  sets, each set 2 elements. Height = 9

→  $\frac{n}{512}$  sets, each set 2 elements. Height = 10

→  $\frac{n}{1024}$  sets, each set 2 elements. Height = 11

→  $\frac{n}{2048}$  sets, each set 2 elements. Height = 12

→  $\frac{n}{4096}$  sets, each set 2 elements. Height = 13

→  $\frac{n}{8192}$  sets, each set 2 elements. Height = 14

→  $\frac{n}{16384}$  sets, each set 2 elements. Height = 15

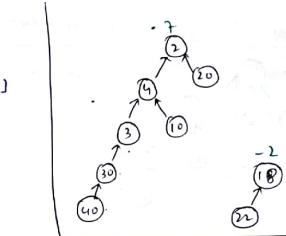
→  $\frac{n}{32768}$  sets, each set 2 elements. Height = 16

→  $\frac{n}{65536}$  sets, each set 2 elements. Height = 17

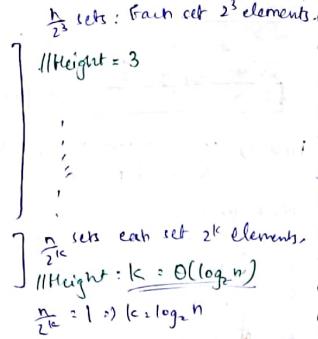
→  $\frac{n}{131072}$  sets, each set 2 elements. Height = 18

→  $\frac{n}{262144}$  sets, each set 2 elements. Height = 19

→  $\frac{n}{524288}$  sets, each set 2 elements. Height = 20



Union (1, 5) ... Union (9, 7, n-3)



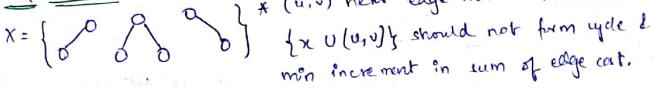
\*  $O(\log n)$  worst case TC of find operation.

# Kruskal's Algorithm:

→ Constructs MST edge by edge.

→ Let "n" be set of edges included into MST so far. Then "n" edges may

not connected.



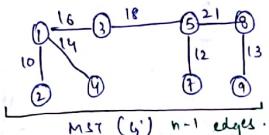
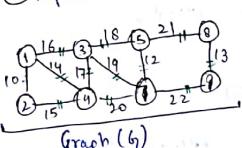
# Procedure of Kruskal's Algo:

n : vertices of graph  
e : edges of graph.

① Del Min cost edge (u, v) from graph G.

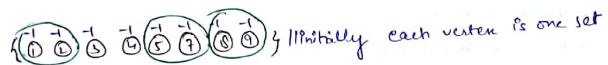
② Add (u, v) into MST if (u, v) not forms cycle in MST

③ Repeat ① & ② until MST becomes (n-1) edges or graph G becomes edges



\* Set Algo used to detect cycle in MST construction using Kruskal's algo.

if (u, v) edge min cost edge deleted from graph  
if (find(u)) != find(v)  
// (u, v) not forms cycle in MST  
else  
// (u, v) forms cycle in MST



Edges deleted from graph

1. (1,2)

find(1) ≠ find(2)

Edge forms cycle in MST

NO [Add (1,2) to MST]

2. (3,7)

find(3) ≠ find(7)

NO [Add to MST]

3. (8,9)

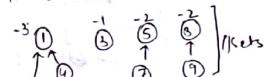
find(8) ≠ find(9)

NO "

4. (1,4)

find(1) ≠ find(4)

NO "



5. (2,4)

find(2) = find(4)

YES [Discard (2,4)]

6. (1,3)

find(1) ≠ find(3)

NO [Add to MST]

7. (3,4)

find(3) = find(4)

YES [Discard (3,4)]

8. (3,5)

find(3) ≠ find(5)

NO [Add to MST]



9. (3,7)

find(3) = find(7)

YES

10. (4,7)

find(4) = find(7)

YES

11. (5,8)

find(5) ≠ find(8)

NO

\* Continue until MST (n-1) edges or graph becomes '0' edges.



① Sum of frequency counts [message length] using Huffman coding:

$$= 11 \times 2 + 7 \times 2 + 5 \times 3 + 4 \times 3 + 2 \times 4 + 3 \times 4 + 3 \times 3$$

$$= 92 \text{ bits}$$

② Avg. bits per character using Huffman code:

$$\frac{\text{sum of freq.}}{\text{# of char.}} = \frac{92}{35} = 2.63$$

③ Min. & Max. bits for character [Min  $\rightarrow$  2 bits Max  $\rightarrow$  4 bits]

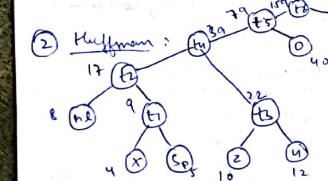
char	a	e	i	o	u	sp	nl	z	x
freq.	200	150	80	40	12	5	8	10	4

④ Sum of freq. count using fixed length,

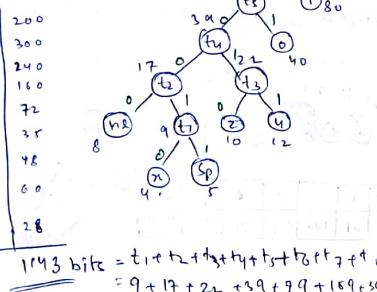
⑤ " " " " " Huffman.

⑥ 4 bits/character

$$\text{freq. count} = 4 \times (200 \times 3 + 150 + 80 + 40 + 12 + 5) = 1520 \text{ bits}$$



$a \rightarrow 0$	$\rightarrow 1 \times 200$
$e \rightarrow 10$	$\rightarrow 2 \times 150$
$i \rightarrow 111$	$\rightarrow 3 \times 80$
$o \rightarrow 110$	$\rightarrow 3 \times 40$
$u \rightarrow 110011$	$\rightarrow 6 \times 12$
$sp \rightarrow 1100011$	$\rightarrow 2 \times 5$
$nl \rightarrow 110000$	$\rightarrow 6 \times 8$
$z \rightarrow 110010$	$\rightarrow 6 \times 10$
$x \rightarrow 1100010$	$\rightarrow 7 \times 4$



$$1143 \text{ bits} = t_1 + t_2 + t_3 + t_4 + t_5 + t_6 + t_7 + t_8$$

$$= 9 + 17 + 22 + 39 + 99 + 189 + 309 + 89$$

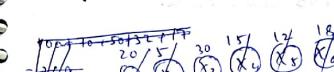
$$\text{④ Avg. bits/char} = \frac{1143}{509} = 2.25 \text{ bits.}$$

⑤ 2 way Optimal Merge Tree TC:

① Build min heap  $\xrightarrow{\Theta(n)}$   
 ② [2 Min del & Insertion]  $\xrightarrow{\text{repeat } (n-1) \text{ times.}}$   $\xrightarrow{\Theta(n \log n)}$   
 $\left[ (n-1)[2 \text{ min del}] + 1 \text{ insertion} \right]$  into min heap

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

Q. char  $[x_1, x_2, x_3, x_4, x_5, x_6]$   
 freq  $[20\%, 5\%, 30\%, 15\%, 12\%, 18\%]$

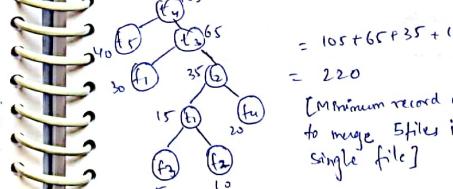


$$= 100 + 62 + 38 + 32 + 17 = 2.49$$

Q. Assume  $(n+m)$  record moves required to merge two files with  $n$  &  $m$  sorted records each into single file of sorted record. How many minimum record moves required to merge  $n=5$  files?

$f_1, f_2, f_3, f_4, f_5$  with  $20, 10, 5, 20, 40$  sorted records each into single sorted file.

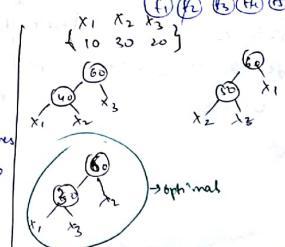
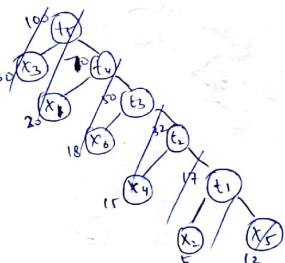
Sol: Merge 2 min file & so on.



$$= 105 + 65 + 35 + 15$$

$$= 220$$

[Minimum record moves to merge 5 files into single file]



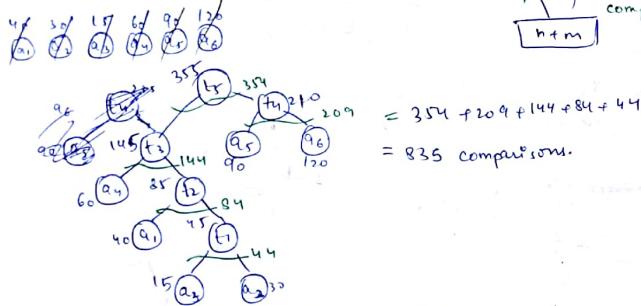
Q. 6 sorted arrays  $[a_1, a_2, a_3, a_4, a_5, a_6]$   
 $\{40, 30, 15, 60, 90, 120\}$  elements in sorted order.

How many comparisons required to merge 6 sorted arrays into single sorted array using optimal algorithm?

Sol. Merge two arrays which are min. elements and so on.

$$\text{comp.} = \frac{\max(n+m-1)}{2}$$

$$\text{comp.} = \frac{\max(n+m-1)}{2}$$



01/09/2017

# Fractional Knapsack Problem: knapsack(bag)

→ Given  $n$  objects &  $m$  capacity profits  $\{P_1, P_2, P_3, \dots, P_n\}$   
 weights  $\{W_1, W_2, W_3, \dots, W_n\}$ .

If  $x_i$  fraction of obj.  $i$  included into knapsack which require  $w_i * x_i$  weight and results  $P_i * x_i$  profit.

where  $0 \leq x_i \leq 1$ .  
 determine fraction values  $\{x_1, x_2, \dots, x_n\}$ , such that  $\sum w_i x_i \leq m$  & maximize profit?  
 [sum of weights included into knapsack]  $\leq m$  & maximize profit?

$$\begin{aligned} \text{Maximize } & \sum_{i=1}^n P_i x_i && \text{Profit} \\ \text{subjected to } & \sum_{i=1}^n w_i x_i \leq m && \text{KnapSack} \\ & 0 \leq x_i \leq 1 && \text{Problem} \end{aligned}$$

Soln. 6th object

$w_i$  unit weight  $\rightarrow P_i$  profit  
 Profit/unit weight  $\rightarrow \frac{P_i}{w_i}$

Greedy soln = Max P/w objects should choose first and so on.

$$\left\langle \frac{P_1}{w_1}, \frac{P_2}{w_2}, \frac{P_3}{w_3}, \frac{P_4}{w_4}, \frac{P_5}{w_5}, \frac{P_6}{w_6} \right\rangle = \left\langle \frac{50}{20}, \frac{30}{10}, \frac{70}{20}, \frac{40}{25}, \frac{60}{35}, \frac{30}{30} \right\rangle$$

$$\{2.5, 3, 3.5, 1.6, 1.71, 1.6\} = \{0.5, 0.2, 0.1, 0.5, 0.4, 0.6\}$$

\* fill knapsack by obj. in descending order of P/w ratios:  
 $w_i * x_i \leq 90 \Rightarrow 20 * 1 + 10 * 1 + 20 * 1 + 35 * 1 + 25 * \frac{1}{2.5} = 90$   
 $P_i * w_i \Rightarrow 70 * 1 + 30 * 1 + 50 * 1 + 60 * 1 + \frac{40 * 1}{2.5} = 210 = \text{Max profit}$

$\{x_1, x_2, x_3, x_4, x_5, x_6\}$  optimal solution  
 $\{1, 1, 1, 1, 0, 0\}$

Algo knapsack( $P[]$ ,  $W[]$ ,  $n$ ,  $m$ )  $\Rightarrow \Theta(n \log n)$  TC of fraction knapsack.

{ 1. sort objects based on decreasing order of their P/w ratios }  $\Theta(n \log n)$

2. for ( $i = 1$ ;  $i \leq n$ ;  $i++$ )  $\{ \text{profit} = 0, m = m \}$

{ if ( $w_i \leq m$ )  
 $\{ x_i = 1;$   
 $\text{profit} = \text{profit} + P_i;$   
 $m = m - w_i \}$   
 else  
 $\{ x_i = \frac{m}{w_i};$   
 $\text{profit} = \text{profit} + P_i * x_i \}$   
 }  
 3. return ( $x[]$ , profit) }  $\Theta(n)$

## # 0/1 Knapsack Problems

$$\begin{aligned} \sum_{i=1}^n p_i x_i & \text{ Maximize profit} \\ \sum_{i=1}^n w_i x_i & \leq m \quad [\text{Subjected to}] \\ x_i & = 0 \text{ or } 1 \end{aligned}$$

Eg.  $n=3$   
 $(p_1, p_2, p_3) = (60, 10, 30)$ ,  $w_1, w_2, w_3 = (20, 10, 10)$   
 $\text{order obj } (0, 0, 02)$   
 $\text{Profit} = 60 < x_1, x_2, x_3 >$   
 $\text{Not optimal. } < 0, 0, 0 >$

# Brute Force Algorithm:  
→ Compute cost of all feasible solution and return feasible set whose result is optimal.

→ 0/1 Knapsack Problem [Brute Force Method]  $T.C = \Theta(2^n)$

Compute profit for every subset of object.

obj	profit	weight
{ }	0	0
{0, 1}	60	20
{1, 2}	40	15
{2, 3}	30	11
{3, 4}	90	31X
{0, 1, 3}	100	35X
{0, 1, 2}	70	26
{0, 2, 3}	130	46X
{0, 1, 2, 3}		

① → SSS Ag.  
② → MST  
③ → Huffman

- # Job Scheduling based on deadline:  
→ Given  $n$  tasks  $[T_1, T_2, \dots, T_n]$  & profits  $< p_1, p_2, \dots, p_n >$  & deadline  $d_1, d_2, \dots, d_n$ .  
If  $T_i$  is executed only before  $d_i$  time, then resultant profit  $p_i$ , otherwise no profit.
- Maximize profit in order to schedule tasks with constraints:
- ④ only one task can execute at a time.
  - ⑤ every task ready to execute.
  - ⑥ one unit execution time for each task.
- Eg.  $\begin{array}{lllllll} \text{obj} & \text{DS} & \text{CO} & \text{Algo} & \text{DS} & \text{DBMS} & ? \\ \text{marks} & 20 & 15 & 25 & 30 & 18 & 3 \\ \text{Deadline} & 3 & 2 & 3 & 1 & 2 & 4 \end{array}$  days. Max Marks?
- Schedule assignments to get maximum marks.
- $\begin{array}{ccccccc} \text{DS} & 1 & 2 & 3 & 4 & 5 & 6 \\ \text{Time line} & \boxed{1} & \boxed{2} & \boxed{3} & \boxed{4} & \boxed{5} & \boxed{6} \\ \text{Task} & T_1 & T_2 & T_3 & T_4 & T_5 & T_6 \end{array}$
- Choose Max Marks task  $[T_i]$  &  $d_i$ .
- Time line:  $30 + 20 + 35 = 85$  days.
- # Greedy Solution: Job scheduling  $\Rightarrow T.C = \Theta(n^2)$
- ⑦ Sort tasks based on decreasing order of profits  $\left[ O(n \log n) \right]$   
 $\{T_1, T_2, T_3, \dots, T_n\}$  order of tasks  
 $p_1 \geq p_2 \geq p_3 \geq \dots \geq p_n$
- ⑧ Choose task  $T_i$  in above order ( $d_i$  deadline) & schedule  $T_i$  at  $d_i$  time, if free.  
if  $d_i$  time not free ( $d_i - 1$ )<sup>th</sup> time and so on.
- ⑨ If no time slot free from  $d_i$  to 1 then discard  $T_i$ .
- WB:  $p_1, p_2, p_3, \dots, p_n$  (⑩)
- $\begin{array}{cccccc} T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 \\ 15 & 20 & 20 & 18 & 18 & 10 & 23 & 16 & 25 \\ 7 & 2 & 5 & 3 & 4 & 5 & 2 & 7 & 3 \end{array}$
- $20 + 23 + 25 + 18 + 30 + 15 + 16 = 147$ .
- $\begin{array}{|c|c|c|c|c|c|} \hline & T_2 & T_7 & T_9 & T_5 & T_3 \\ \hline 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline \end{array}$
- ⑪ → d.  
⑫ → n.

## # Sorting Algorithms:

### ① Comparison based sorting algo:

Algorithm	Time complexity	Space complexity	Stable	Inplace
	BC	AC	sort	sort
① Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	No
② Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	YES
③ Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	NO
④ Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	YES
⑤ Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	NO
⑥ Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	YES

\* WC TC of any comparison based sorting algo =  $\Omega(n \log n)$

### ② Non-Comparison based sorting algo:

① Counting Sort:  $TC = O(n+k)$

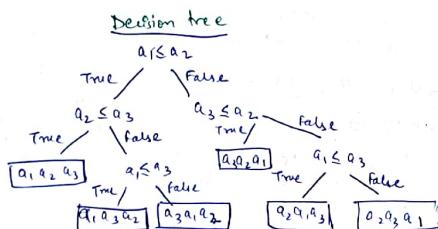
- Stable
- Not Inplace

② Radix/Bucket Sort:  $TC = O(d \times n)$

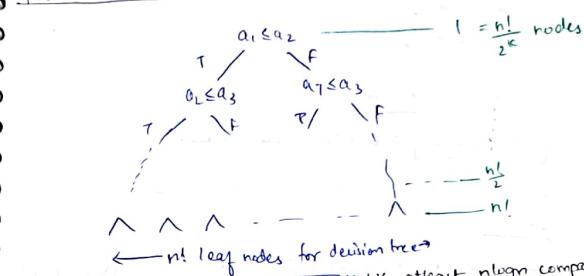
- Stable
- Inplace

\* Theorem: Worst case time complexity of comparison based sorting algo for  $n$  elements is  $\Omega(n \log n)$ .

$n=3$   $[a_1, a_2, a_3]$



Given  $n$  element decision tree:



$$\frac{n!}{2^k} = l \Rightarrow k = \log n! \Rightarrow k = n \log n$$

\* WC atleast  $n \log n$  comparisons required for any comparison based sorting algorithm.

# Bubble Sort Algo: Compare and exchange adjacency elements.

Stable  
& Inplace

+ Procedure:

```

Pass1: compare a[j] & a[j+1]
if a[j] > a[j+1] then swap(a[j], a[j+1])
for j=1 to n-1
  // a[n] gets sorted.
Pass2: compare a[j] & a[j+1]
if a[j] > a[j+1] then swap(a[j], a[j+1])
for j=1 to n-2
  // a[n-1, n] get sorted.
  
```

Pass (n-1) Repeat

Eg:	1 2 3 4 5 6
a	40 30 20 10 60 70
Pass 1	30 40 20 60 10 70
Pass 2	20 30 40 60 10 70
Pass 3	20 30 10 40 60 70
Pass 4	20 10 30 40 60 70
Pass 5	10 20 30 40 60 70

Algo. Bubblesort( $a, n$ )

```

{
  for (i=1; i<=n-1; i++)
  {
    for (j=1; j<=n-i; j++)
    {
      if (a[j] > a[j+1])
        swap(a[j], a[j+1])
    }
  }
}
  
```

	# of comp.	# of swap.
Pass 1	$n-1$	[0 to $n-1$ ]
Pass 2	$n-2$	[0 to $n-2$ ]
:	:	
Pass( $n-1$ )	1	[0 to 1]
$T_C = \frac{n(n-1)}{2} + \left\{ 0 \text{ to } \frac{n(n-1)}{2} \right\} = \Theta(n^2)$ In all cases.		

#### # Selection Sort Algorithm:

→ find position of min element from  $a[1]$  to  $a[n]$  and swap( $a[minpos], a[j]$ ) where  $j$  is  $1, 2, \dots, n-1$  for  $n-1$  passes.

#### + Procedure:

Pass 1 : Find position of min element  $a[1]$  to  $a[n]$  and swap with  $a[1]$   
 Pass 2 : Find position of min element  $a[2]$  to  $a[n]$  and swap with  $a[2]$   
 :

Pass  $n-1$  : Find position of min element  $a[n-1]$  to  $a[n]$  & swap with  $a[n-1]$ .

Eg.	1 2 3 4 5 6 7
Pass 1	60   50   30   60   50   0   20
Pass 2	10   20   30   60   50   60   20
Pass 3	10   20   30   50   60   60   20
Pass 4	10   20   30   50   60   60   50
Pass 5	10   20   30   50   50   60   60
Pass 6	10   20   30   50   50   60   60

	# of comp	# of swap
Pass 1	$n-1$	1
Pass 2	$n-2$	1
:	:	
Pass $n-1$	1	1

$$TC = \left\{ \frac{n(n-1)}{2} + (n-1) \right\} = \Theta(n^2) \text{ All cases.}$$

```
Algo SelectionSort(a,n)
{
  for (i=1; i<=n-1; i++)
    { // find pos of min element a[i] to a[n]
      MinPos = i; Li
      for (j=i+1; j<=n; j++)
        { if (a[j]<a[MinPos])
            MinPos=j;
        }
      swap(a[i], a[MinPos])
    }
}
```

→ Selection sort can be treated as special case of Quick sort where pivot element is minimum.  
 \*\*\*

→ Selection sort TC  $\rightarrow \Theta(n^2)$  [A]

Quick sort TC  $\rightarrow \Theta(n \log n)$  [AC]

[Selection sort runs faster than quicksort for less file array]  
 [Quicksort runs faster than selection sort for large file array]

```
Sorting(a,n)
{
  if (n<20)
    SelectionSort(a,n)
  else
    Quicksort(a,n)
}
```

#### # Insertion Sort:

→  $a[i]$  placed in correct position in sorted part of array  $a[1 \dots i-1]$ .  
 where  $i=2$  to  $n$  for  $(n-1)$  passes.

#### Procedure:

Pass 1 : place  $a[2]$  in correct position of sorted array  $a[1 \dots 1]$   
 //  $a[1 \dots 1]$  in sorted order  
 Pass 2 : place  $a[3]$  in correct position of sorted array  $a[1 \dots 2]$   
 //  $a[1 \dots 2]$  in sorted order.  
 :

Pass  $n-1$  : place  $a[n]$  in correct position of sorted array  $a[1 \dots n-1]$   
 //  $a[1 \dots n]$  in sorted order.

Algo InsertionSort(a,n)

```
{
  for (i=1; i<=n-1; i++)
    item=a[i+1]; j=i;
    while (j>1 & a[j]>item)
      a[j+1]=a[j];
      j=j-1;
    a[j+1]=item;
}
```

1 2 3 4 5 6 7
60   50   20   60   50   10   20

1 2 3 4 5 6 7
50   60   20   60   50   10   20

1 2 3 4 5 6 7
30   60   20   60   50   10   20

1 2 3 4 5 6 7
20   60   30   60   50   10   20

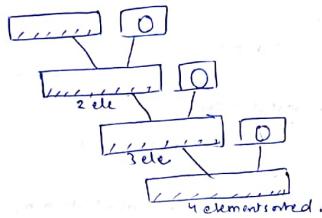
1 2 3 4 5 6 7
10   60   30   60   50   10   20

1 2 3 4 5 6 7
10   30   60   50   10   20

1 2 3 4 5 6 7
---------------

	Min & Min Comp. shift	Max & Max Comp. shift	Avg & Avg Comp Shift
Pass 1	1 & 0	1 & 1	1 & 1
Pass 2	1 & 0	2 & 2	2 & 2
Pass 3	1 & 0	3 & 3	3 & 3
⋮	⋮	⋮	⋮
Pass(n-1)	1 & 0	(n-1) & (n-1)	(n-1) & (n-1)
	<u>(n-1) &amp; 0 comp</u>	<u><math>\frac{n(n-1)}{4}</math> &amp; <math>\frac{n(n-1)}{2}</math> comp</u>	<u><math>\frac{n(n-1)}{4}</math> &amp; <math>\frac{n(n-1)}{2}</math> shift</u>
	<u>WC TC of insertion sort = <math>\Theta(n^2)</math></u>	<u>AVG TC of insertion sort = <math>\Theta(n^2)</math></u>	

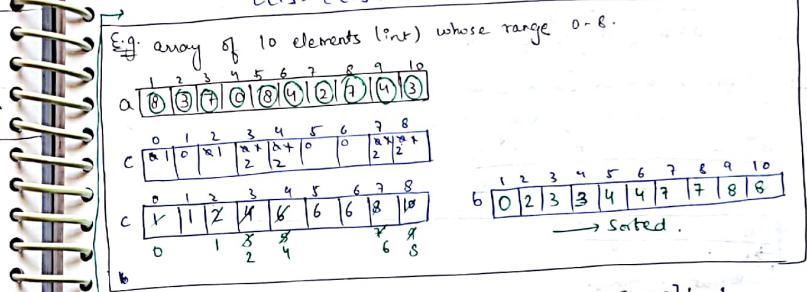
\* Insertion sort : special case of merge sort where merge is merging 2 sorted arrays with only one element in 2nd sorted array.



- If input array with less elements, insertion sort runs faster than merge sort.
- If input array with large more elements, merge sort runs faster than insertion sort.

# Non-comparison based sorting :

- ① Counting Sort Algo: array  $a[1..n]$  are  $n$  integers whose element values range  $0..k$ .
1. Initialize count array  $c[0..k]$  by 0's
2. for each element of array "a" increase count val. of count array
3. Increase count array by previous index of count array

$$c[i] = c[i] + c[i-1]$$


→ a[i] element place  $b[c[a[i]]]$  & decrease  $c[a[i]]$  by 1.

Algo CountingSort( $a, n, k$ )

{  $a[1..n]$  are  $n$  integers whose element range 0 to  $k$ . }

for ( $i=0$ ;  $i \leq k$ ;  $i++$ )

{  $c[i] = 0$ ; }  $\Theta(k)$

for ( $i=1$ ;  $i \leq n$ ;  $i++$ )

{  $c[a[i]] = c[a[i]] + 1$ ; }  $\Theta(n)$

for ( $i=1$ ;  $i \leq k$ ;  $i++$ )

{  $c[i] = c[i-1] + c[i]$ ; }  $\Theta(k)$

for ( $i=n$ ;  $i \geq 1$ ;  $i--$ )

{  $b[c[a[i]]] = a[i]$ ; }  $\Theta(n)$

{  $c[a[i]] = c[a[i]] - 1$ ; }

return ( $b[1..n]$ ); }

\* TC of counting sort =  $\Theta(n+k)$

\* Best Case TC =  $\Theta(n)$  [If  $k \leq n$ ]

\* Stable sorting

\* Not Inplace sorting

## Radix Sort Algo:

Radix ( $r$ ):  $r$  different symbols used to represent any number using number systems symbol val[0, 1, 2, ...,  $r-1$ ]

radix = 16 [0, 1, 2, 3, ..., 9, A, B, C, D, E, F]

radix = 10 [0, 1, ..., 9, A, B, C, D, E, F, G, H, I, J]

Radix sort:  $n \times d$  of elements of array [1..n]

$r$ : is radix of Number system

$d$ : # of digits of each element.

Step 1: define  $x$ -Queues [Queue0, Queue1, ..., Queue( $r-1$ )]

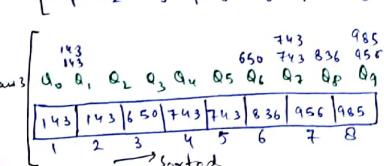
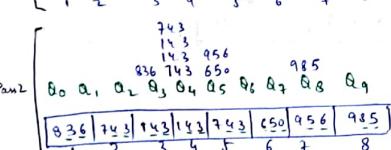
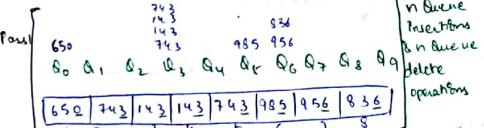
Step 2: For each element of array place elements into Queue based on  $k^{th}$  least significant digits of element.

$k$  is 1 to  $d$  for  $d$  passes  
and retrieve elements from Queue 0 to Queue( $r-1$ ) & store in array 'a'.

repeat  $d$  times.

E.g.  $n=8$   $d=3$   $x=10$  (decimal number)

1	2	3	4	5	6	7	8
a   956   743   143   650   836   985   193   743							



# TC of Radix sort:  $n \rightarrow \# \text{ of elements of array}$   
 $d \rightarrow \# \text{ of digits of each element}$   
 $r \rightarrow \text{radix}$

Best Case Tc of Radix sort =  $\Theta(n)$  [if  $d$  is const.]

$\Theta(n \lg r)$  ① :

Element val ( $x$ ) =  $n^3$

# of digits to denote  $x$  in decimal =  $\log_{10} n^3 = 3 \log_{10} n$

$$x = n^3$$

# of digits for  $x$  using radix =  $n$  (Number sys.)

$$\log_r x = 3 \log_{10} n^3 = 3 \text{ digits.}$$

# Radix sort Algo:

$a[1..n]$   $n$  elements in decimal val. of each element  $< 0..n^3$

① Convert each element of array from radix-10 to radix-n.  
( $x$  is element of array)

$$\log_n x = \log_{10} n^3 = \frac{3 \text{ digits}}{\text{constant}}$$

② Run Radix sort for above resulted array.

③ Convert each element of above result from radix-n to radix-10

$$\underline{\underline{\Theta(n) = TC}}$$

# TC of BFS:

- ① Using adj. list graph:  $\Theta(n+e)$
- ② Using adj. matrix graph:  $\Theta(n^2)$

# SC of BFS:  $\Theta(n)$  Queue DS space.

# Depth First Search: [start]

- Graph vertices visited based on depth.
- Vertex (v) which started exploration first is the vertex that completes exploration last.

DFS(4) Adj(4) = {2, 8}  
 | L DFS(2) Adj(2) = {1, 4, 5}  
 | L DFS(1) Adj(1) = {2, 3}  
 | L DFS(3) Adj(3) = {1, 6, 7}  
 | L DFS(6) Adj(6) = {3, 8}  
 | L DFS(8) Adj(8) = {4, 5, 6, 7}  
 | L DFS(5) Adj(5) = {2, 8}  
 | L DFS(7) Adj(7) = {3, 8}

4 5 2 1 3 6 5 8 5 5 e 7 5 7 e 6 e 3 e 1 e 2 e 4 e  
 DFS Sequence = 4, 2, 1, 3, 6, 8, 5, 7  
 [visited order]

Source: B

Back Edge: Graph edge which is connected child to parent.

---	→	Forward Edge
---	→	Back Edge
→	→	Tree Edges

Cross Edge = 0  
 Back Edge = 2  
 Forward Edg. Edge = 3  
 Tree Edges = 7

• DFS Spanning Tree Edges: Tree Edges / Cross Edge: Graph edges which connect same level type vertices.  
 • forward. Edge: Edge from prev. level to next level.

```

Algo DFS(b,n,e,s)
{
    for(i=1; i<=n; i++)
        if visited[i] == 0
            if s is source
                bFS(s)
                print(s)
                visited[s] = 1;
                v is set of adj of vertex(s) -> deg(s)
                for (vertices of v set)
                    if (visited[v] == 0)
                        call b-DFS(v)
                }
            print(s)
            ended
}
}

```

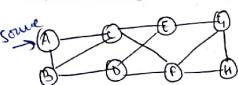
TC of DFS:  
 Adj. list  $\rightarrow \Theta(n + e)$   
 Adj. matrix  $\rightarrow \Theta(n^2)$

---

SC of DFS:  
 $= O(n)$

## \*\*\* Applications of BFS / DFS:

- ① BFS Algo can be used as single source shortest path algo for unweighted edges graph.



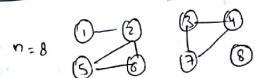
	A	B	C	D	F	G	H
dist-	0	1	1	2	2	2	3

$$T_C = (n+e) \text{ or } (n^2)$$

- Q. which DS used to compute single source shortest path algo for unweighted graphs?

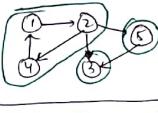
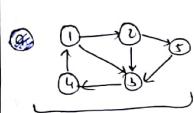
- unweighted graphs?
    - ⑥ Min heap
    - ⑥ Stack
    - ⑦ Queue
    - ⑧ Linked List

- ② BFS/DFS can be used to test no. of connected components of undirected graphs.



④ DFS algo can be used to test # of strongly connected components of directed graph.

Strongly connected component: If every pair of vertices  $(u, v)$  & there must be path from  $u \rightarrow v$  &  $v \rightarrow u$ .



one strongly connected 3-strongly connected.

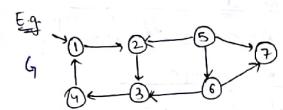
\* Procedure to test # of strongly connected components.  
TC:  $O(n + e)$

⑤ call DFS and store finish time of exploration of vertices in stack (S)

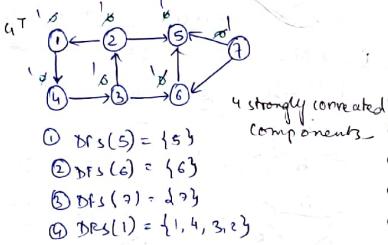
⑥ Compute  $G^T$ .

Transpose of  $G$ :  $G^T$

⑦ Call DFS for  $G^T$  based on unvisited vertex from top of stack as source.  
[# of DFS calls to visit all vertices of  $G^T$  is # of strongly connected component of  $G^T$ ].



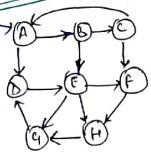
⑧ Call DFS for  $G$



- ① DFS(5) = {5}
- ② DFS(6) = {6}
- ③ DFS(4) = {4}
- ④ DFS(1) = {1, 2, 3}

4 strongly connected components

Pg. 20. Q31.



2 strongly connected components.

① DFS(A) = {A, C, B}  
DFS(B) = {B, D, F, G, H}

TC =  $O(n)$  [ $2n+1$  fn calls each call O(1 time)]

SC =  $\Omega(\log n)$  to  $O(n)$

⑤ DFS Algo can be used to detect cycle in directed graph.

⑥ DFS Algo can be used to detect cycle in undirected graph.

### # TREE TRAVERSALS:

⇒ In-order:

[Traverse Left subtree, Visit root, Traverse Right subtree]

⇒ Pre-order:

[Visit root, Traverse Left subTree, Traverse Right subTree]

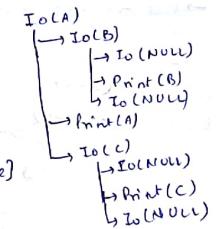
⇒ Post-Order:

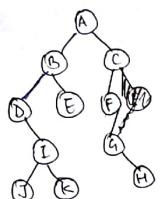
[Traverse Left subTree, Traverse Right subTree, Visit root]

\* Algo Inorder(t):

```
{algo if (t == NULL)
    return
else
    In-order(t->left);
    print(t. element);
    In-order(t->right);
}
```

# of func calls  
=  $n + [n+1]$   
 $\approx 2n+1$

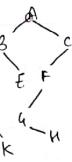




In order  $\rightarrow$  DJIKBEAGHFC

Pre order  $\rightarrow$  ABDIJKCEFGH

Post order  $\rightarrow$  JKIDEBHKGFCA



- \* \* If [In order & Pre order] or [Inorder & Post order] of sequence given, can construct unique binary tree.
- If [Pre order & post order] sequence given, not possible to construct unique binary tree.
- In-order sequence of Binary Search tree is ascending order of keys
- If pre order or post order of binary search tree given, can construct unique binary tree.

#### → Expression Tree:

In order sequence  $\rightarrow$  Infix expression

Pre order sequence  $\rightarrow$  Prefix expression

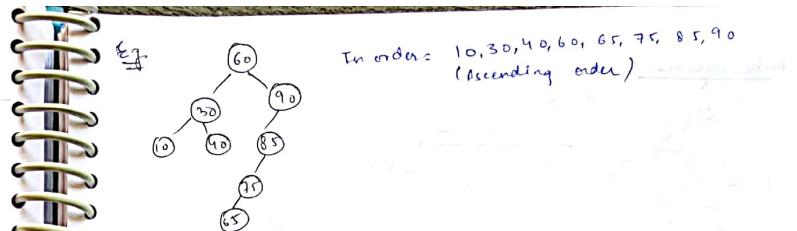
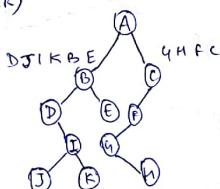
Post order sequence  $\rightarrow$  Post fix expression.

Q. What is post order sequence for binary tree with 9 nodes?

Inorder: DJIKBEAGHFC (LNR)

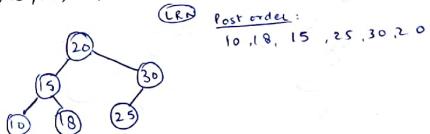
Preorder: ABDIJKCEFGH (NLR)

not  
a binary  
tree



In order = 10, 30, 40, 60, 65, 75, 85, 90  
(Ascending order)

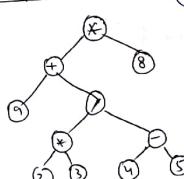
Q. What is post order sequence of BST with given pre order.  
20, 15, 10, 18, 30, 25. In order = 10, 15, 18, 20, 25, 30



LRN

Post order:  
10, 18, 15, 25, 30, 20

# Expression Tree:  $(a + ((2 \times 3) / (4 - 5))) \times 8$



IO  $\Rightarrow$  9 + 2 \* 3 / 4 - 5 \* 8 [Infix]

Pre  $\Rightarrow$  + \* 2 3 / 4 5 8 [Prefix]

Post  $\Rightarrow$  9 2 3 \* 4 5 / 1 + 8 \* [Postfix]

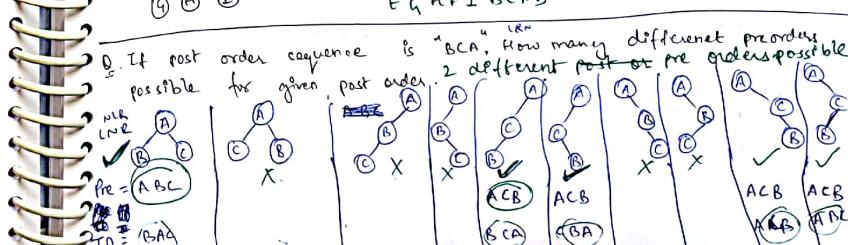
GATE:

In order: Traverse 1st subtree, Print, Traverse 2nd subtree, Traverse 3rd subtree

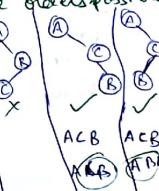
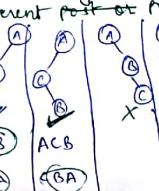
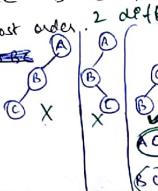
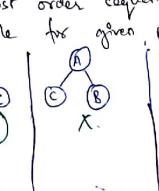
E B G F H I A C D

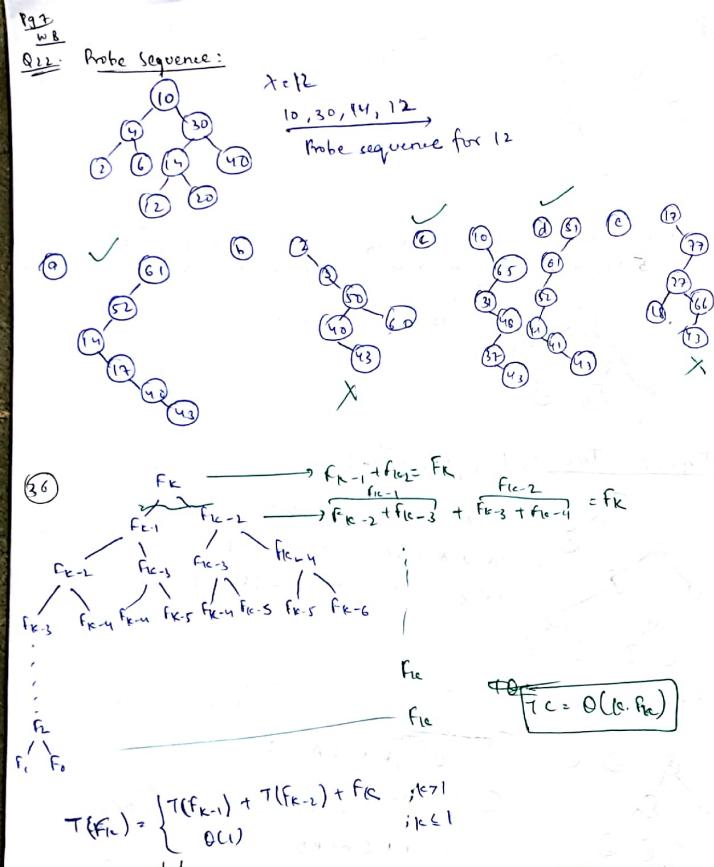
② Traverse, Traverse, Print, Traverse 1st s.t., 2nd s.t., 3rd subtree

E G H F I B C P D



Q. If post order sequence for given post order is "BCA", How many different preorders are possible?





38) sorted any two element sum  $< 1000$  exist or not?

If ( $a[1] + a[2] < 1000$ )  
    returns (True)

else return (false)

卷之三

## DYNAMIC PROGRAMMING: 03/09/2017

→ Also called Principle of optimality design technique.

→ Whatever cost of first decision, cost of remaining decisions along with first decision must be optimal.

$$\text{decisions } \xrightarrow{x_1} \xrightarrow{x_2} \xrightarrow{x_3} \dots \xrightarrow{x_{n-k+1}} \xrightarrow{x_n} \xrightarrow{x_{n-1}}$$

$$= \min \left\{ \begin{array}{l} x_1 + \text{cost}(x_1, k-1) \\ x_2 + \text{cost}(x_2, k-1) \\ x_3 + \text{cost}(x_3, k-1) \end{array} \right.$$

### Greedy Algo

→ Chooses one feasible solution using predefined method return optimal solution.

→ Runs in polynomial time complexity.

→ Fails to solve some optimization problems.  
E.g. 0/1 Knapsack problem.

### Dynamic Programming

→ Computes all feasible solutions and returns optimal solution.

→ May require exponential time complexity. [Almost polynomial TC.]

→ Solves every optimization problem because uses principle of optimality.

### Brute Force Algo

→ Computes all feasible solution and returns optimal.

→ If # of solutions exponential  $[2^n, n!]$ . Then TC of solution using brute force algo also exponential.

→ Less space complexity.

### Dynamic Programming

→ Computes all feasible solution such that avoids recomputation and returns optimal solution.

→ Even exponential feasible soln problem may solve in polynomial time because of avoiding recomputation.

→ More space complexity.

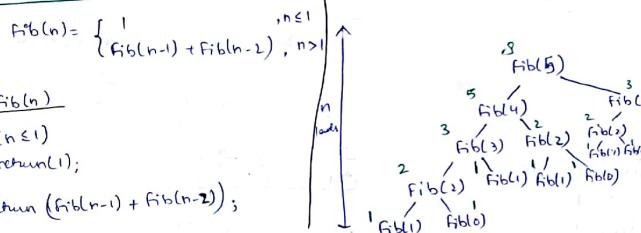
### # Step to Design Dynamic Programming:

- ① Solution of problem be formulated recursively. [To avoid recomputation]
- ② Formulate Recurrence Relation for solution of problem.
- ③ Solve Recurrence Relation for given input in Bottom-Up approach.
  - return optimal solution.
  - return optimal sequence.

### ① n<sup>th</sup> Fib Number:

Algo fib(n)

```
{ if (n ≤ 1)
    return(1);
else
    return (fib(n-1) + fib(n-2));
```



$$\begin{aligned}
 \# \text{ of func calls} &= 1 + 2 + 2^2 + 2^3 + \dots + 2^{n-3} + 2^{n-2} + 2^{n-1} \\
 &= 2^{n-1} \\
 \text{TC} &= O(2^n) \quad SC = O(n) \text{ || Depth of recursion.}
 \end{aligned}$$

### Dynamic Solution:

```
Algo nthfib(n)
{ for (p=0; i≤n; i++)
  { a[i] = -1;
    fib(n);
    if (n ≤ 1)
      a[n] = 1;
      return (1);
    else
      { if (a[n-1] == -1)
          a[n-1] = fib(n-1);
        if (a[n-2] == -1)
          a[n-2] = fib(n-2);
        a[n] = a[n-1] + a[n-2];
      }
    return (a[n]);
  }
```

0	1	2	3	4	5
1	1	2	3	5	8

fib(5)<sub>a[5]=8</sub> = a[4]+a[3]

fib(4)<sub>a[4]=a[3]+a[2]</sub>

fib(3)<sub>a[3]=a[2]+a[1]</sub>

fib(2)

fib(1)

# of function calls = n+1

[Each func call required constant time]

TC of algo = O(n) // polynomial

### SC in Algo:

a[0..n] (n+1) entries occupy m/m units

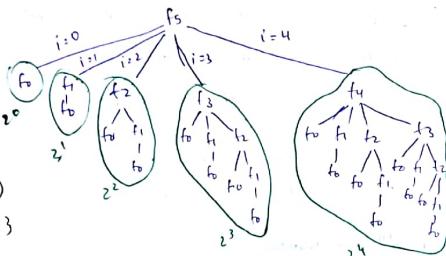
Stack : n entries

O(n)

WAP Pg 12

Q. 61. double foo(int n)

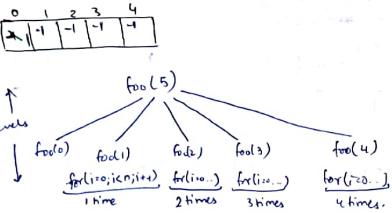
```
{
    if (n == 0) return 1;
    else
        sum = 0;
        for (i = 0; i < n; i++)
            sum += foo(i);
        return sum;
}
```



$$\text{No. of funcn calls} = 2^{[1+2+2^2+\dots+2^{n-1}]+1} = 2^n$$

$$TC = \Theta(2^n)$$

$$SC = \Theta(n)$$



$$\# \text{ of funcn calls} = (n-1)$$

$$TC \text{ of algo} = [n+1] + \{1+2+\dots+(n-2)+(n-1)\}$$

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

$$SC \text{ of algo: } a[0 \dots n-1] = \Theta(n)$$

$$\text{start space} = \Theta(1) / 2 \text{ entries} \\ \underline{\Theta(n)}$$

Q. 62

```
for (i = 0; i < n; i++)
    a[i] = -1;
```

foo(n)

if (n == 0)

{ a[0] = 1;

return (1)

}

else

{ sum = 0

for (i = 0; i &lt; n; i++)
 if (a[i] == -1)

{ a[i] = foo(i); }

sum = sum + a[i];

}

return (sum)

}

## # Matrix Chain Problem:

 $\rightarrow A_{pq} B_{qxr} \text{ cost of multiplication} = p \cdot q \cdot r$  (element multiplication required)

→ Matrix Multiplication Associativity: [A, B, C, Matrices]

$$(A \times B) \times C \equiv A \times (B \times C)$$

But no. of multiplications differ.

$$A_{100 \times 50}, B_{50 \times 75}, C_{75 \times 10}$$

① Cost of  $(A \times B) \times C$ :

$$100 \times 50 \times 75 = 375000 \\ 100 \times 75 \times 10 = \frac{75000}{450000} \text{ (scalar mult.)}$$

② Cost of  $A \times (B \times C)$ 

$$\begin{aligned} \text{optimal} \\ \text{Multi. of A, B, C} \\ \text{matrix} \end{aligned} \rightarrow 50 \times 75 \times 10 = 37500 \\ 100 \times 50 \times 10 = \frac{50000}{87500} \text{ (scalar mult.)}$$

## # Problem Statement:

Given  $n$  matrices:  $A_1, A_2, A_3, \dots, A_n$  and  $P_0, P_1, P_2, P_3, \dots, P_n$  order set such that cost of multiplication of  $n$ -matrices is minimum.

$$P_0, P_1, P_2, P_3, P_4 \\ \langle 10, 5, 3, 6, 7 \rangle$$

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

# of Matrices # of Parenthesizations

1. $A_1$	1
2. $A_1, A_2$	1
3. $A_1 \times A_2$	2
4. $A_1, A_2, A_3, A_4$	5
5. $A_1, A_2, A_3, A_4, A_5$	14
$n$	$\frac{(2n-2)!}{(n-1)! (n-1)!}$

$T(n) = \#$  of parenthesizations for  $n$ -matrices

$$T(n) = \begin{cases} 1 & ; n \leq 2 \\ \sum_{k=1}^{n-1} T(k) \cdot T(n-k) & ; n > 2 \end{cases}$$

$A_1 A_2 A_3 A_4 \dots A_{n-1} A_n$

$$T(n) = T(1) + T(n-1) + T(2) \cdot T(n-2) + \dots + T(n-1) \cdot T(1)$$

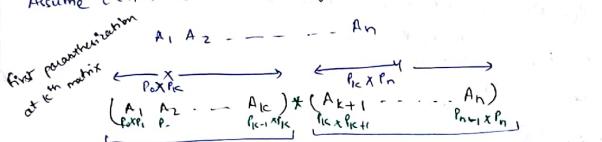
$$T(n) = \frac{1}{n} \left[ \frac{(2n-1)!}{((n-1)!(n-1)!)} \right] \rightarrow \# \text{ of feasible solution for matrix chain problem.}$$

Finite Force Algo  $\Rightarrow T(n) = \Omega(2^n)$

# DP Solution [recursive solution] of matrix chain problem  $[T(n) = \Theta(n^3)]$

$A_1 A_2 A_3 \dots A_n$  are  $n$  matrices  
 $\langle p_0 p_1 p_2 \dots p_n \rangle$  are order set

Assume  $c(i, n)$  is mincost of Matrices.



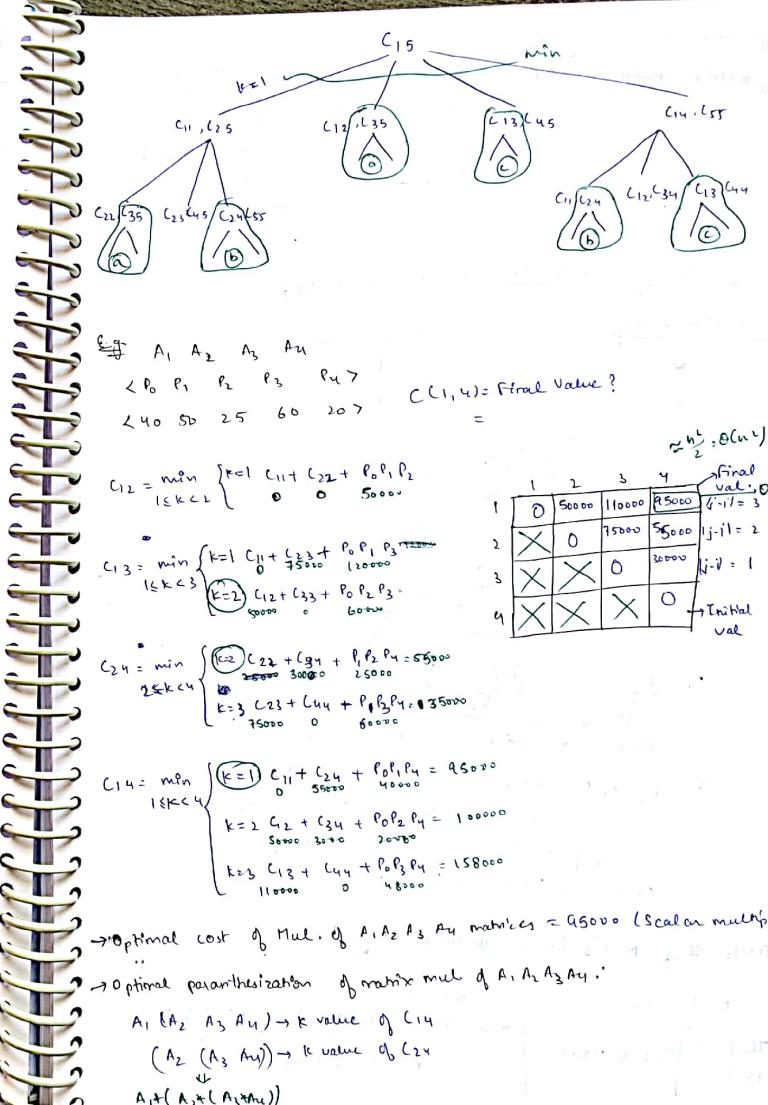
Min cost of Mul of  
 $A_1 \dots A_k$  is  $c(1, k)$

$$c(1, n) = \min_{1 \leq k < n} \{ c(1, k) + c(k+1, n) + p_0 \cdot p_k \cdot p_n \}$$

\* General Recurrence Relation of Matrix chain:

$A_i A_{i+1} \dots A_j$  Matrices  
 $\langle p_{i-1} p_i \dots p_{j-1} p_j \rangle$  Order set

$$c(i, j) = \min_{i \leq k < j} \{ c(i, k) + c(k+1, j) + p_{i-1} \cdot p_k \cdot p_j \} ; i < j$$



e.g.  $A_1 A_2 A_3 A_4$   
 $\langle p_0 p_1 p_2 p_3 p_4 \rangle$   
 $\langle 40 50 25 60 20 \rangle$

$$c(1, 4) = ? \text{ final value?}$$

$$c_{1,2} = \min_{1 \leq k \leq 2} \{ c_{1,1} + c_{2,2} + p_0 p_1 p_2 \}$$

$$k=1: c_{1,1} + c_{2,2} + p_0 p_1 p_2 = 50000$$

$$k=2: c_{1,2} + c_{2,3} + p_0 p_2 p_3 = 120000$$

$$c_{1,3} = \min_{1 \leq k \leq 3} \{ c_{1,1} + c_{2,3} + p_0 p_1 p_3 \}$$

$$k=1: c_{1,1} + c_{2,3} + p_0 p_1 p_3 = 75000$$

$$k=2: c_{1,2} + c_{2,3} + p_0 p_2 p_3 = 60000$$

$$c_{1,4} = \min_{1 \leq k \leq 4} \{ c_{1,1} + c_{2,4} + p_0 p_1 p_4 \}$$

$$k=1: c_{1,1} + c_{2,4} + p_0 p_1 p_4 = 95000$$

$$k=2: c_{1,2} + c_{2,4} + p_0 p_2 p_4 = 100000$$

$$k=3: c_{1,3} + c_{2,4} + p_0 p_3 p_4 = 135000$$

$$k=4: c_{1,4} + c_{2,4} + p_0 p_4 p_4 = 158000$$

	1	2	3	4	final val
1	0	50000	110000	95000	95000
2	X	0	75000	55000	55000
3	X	X	0	30000	30000
4	X	X	X	0	Initial val

$\approx \frac{n^3}{2} = O(n^3)$

Bottom

$\rightarrow$  Optimal cost of Mul. of  $A_1 A_2 A_3 A_4$  matrices = 95000 (Scalar multiplication)

$\rightarrow$  Optimal parenthesization of matrix mul of  $A_1 A_2 A_3 A_4$ :

$A_1 (A_2 A_3 A_4) \rightarrow k$  value of  $C_{1,4}$

$(A_2 (A_3 A_4)) \rightarrow k$  value of  $C_{2,4}$

$A_1 ((A_2 A_3) A_4) \rightarrow k$  value of  $C_{1,2}$

TC of matrix chain using DP =  $\Theta(n^3)$

SC of matrix chain using DP =  $c[1..n][1..j]/\text{to store } c(i,j)$   
 $k[1..n][1..n]/\text{to store } k^{\text{th}} \text{ value of}$   
 $\min. \text{ cost of } c[i,j]$   
 $\Theta(n^2)$

### ① No. of Binary search Tree for n distinct keys:

# of distinct keys	# of BSTs
0	1
1	1
2	2
3	5
4	14
5	42
6	132
7	429
8	1430
9	4504
10	14300

$T(n) = \# \text{ of BST's for } n \text{ distinct key.}$

e.g.  $k_1, k_2, k_3, \dots, k_{n-1}, k_n$

$$T(0) = 1 \quad \left[ \begin{array}{l} \text{Initial value} \\ T(0) = 1 \\ \dots + T(n-1), T(0) \end{array} \right]$$

$$T(n) = T(0) \cdot T(n-1) + T(1)T(n-2) + T(2)T(n-3) + \dots + T(n-1)T(0)$$

$$T(n) = \begin{cases} 1 & n \leq 1 \\ \sum_{k=1}^n T(k-1) \times T(n-k) & n > 1 \end{cases}$$

$$T(n) = \frac{1}{n+1} \binom{(2n)!}{n! \cdot n!}$$

# of BSTs possible for indistinct keys.

$$\frac{2^n}{n+1} C_n$$

→ No. of parenthesizations for  $(n+1)$  matrices = # of BST for  $n$  distinct keys.

### ② No. of unlabeled Binary Trees for n nodes:

# of nodes	# of unlabeled BTs
1	1
2	2
3	5
4	14
5	42

$\# \text{ of unlabeled BTs for } n \text{ nodes} = \# \text{ of BSTs for } n \text{ distinct keys.}$

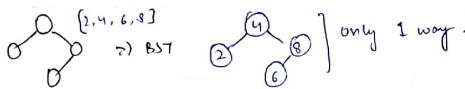
### ③ No. of labeled BTs for n keys:

No. of nodes	# of labeled BTs
1	1
2	$2 \times 2! = 4$
3	$5 \times 3! = 30$
n	$\frac{n!}{n+1} \binom{(2n)!}{n! \cdot n!}$

No. of labeled BTs for nkeys =  $n!$ . [No. of unlabeled BTs]

Q. How many ways can label given unlabeled binary tree of n-nodes  
Δ n distinct keys into Binary search tree.

$$\textcircled{a} \ n! \quad \textcircled{b} \ \frac{1}{n+1} \binom{(2n)!}{n!} \quad \textcircled{c} \ \frac{1}{n+1} \binom{(2n)!}{n!, n!, n!, n!} \quad \textcircled{d} \ 1$$



only 1 way.

# Largest Common Subsequence Problem [LCS Problem]:

Subsequence: Sequence of char from string which may not be consecutive.

$X = "abade"$   
Some subsequence of  $X$ :  
 $\begin{cases} abd \\ abde \\ aae \\ abade \end{cases}$

string  $n$  with  $n$  characters.

# of possible subsequence =  $2^n$

$X \Rightarrow "ab\ ac"$

$2^4 = 16$  possible subsequences of  $X$

→ Common Subsequence: Subsequence which is common for two strings  $X$  &  $Y$ .

$X = abade$  : aad. Some common subsequences of strings  $X$  &  $Y$ .

$Y = baada$  : bad. Largest common subsequence of  $X$  &  $Y$  = length(3)

→ Largest Common Subsequence: Common subsequence of  $X$  &  $Y$ , which is maximum length.

# Recursive Solution of LCS:

$\text{Lcs}(n, m)$ : Largest common subsequence length of two strings  $n$  &  $m$  char. respectively.

$$X = [x_1, x_2, x_3, \dots, x_{n-1}, x_n]$$

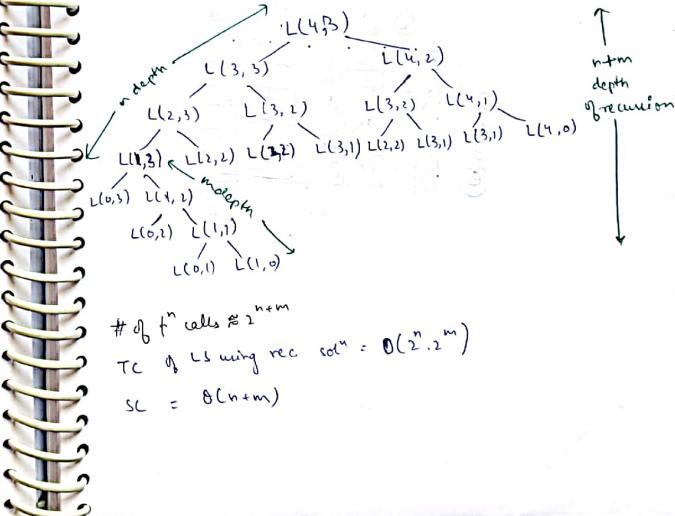
$$Y = [y_1, y_2, y_3, \dots, y_{m-1}, y_m]$$

$$\text{LCS}(n, m) = \begin{cases} 0 & ; n=0 \text{ or } m=0 \\ 1 + \text{Lcs}(n-1, m-1) & ; n>0 \text{ and } m>0 \text{ and } x_n = y_m \\ \max \{\text{Lcs}(n-1, m)\} & ; n>0 \text{ and } m>0 \\ \text{Lcs}(n, m-1) & ; x_n \neq y_m \end{cases}$$

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

General Recurrence Relation for LCS.

$$\text{L}(n, m) = ? \quad // \text{Final value.}$$



# of  $t^n$  cells  $\approx 2^{n+m}$

TC of LCS using rec soln =  $O(2^n \cdot 2^m)$

SC =  $O(n+m)$

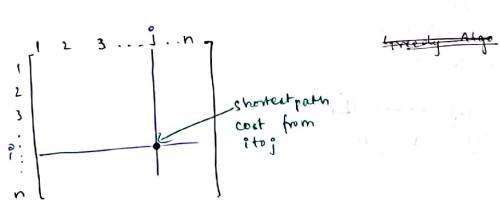


### # All pair shortest path Algo:

Graph G  
List  $[1..n][1..n]$  edge costs  
 $\text{cost}(i, j) = \begin{cases} 0 & \text{if } i=j \\ \text{Edge cost} & \text{if } (i, j) \in \text{Edge} \\ \infty & \text{if } (i, j) \notin \text{Edge} \end{cases}$

Given

\* Compute  $A[1..n][1..n]$  such that  $A[i][j]$ : shortest path cost from vertex  $i$  to vertex  $j$ .



### # Greedy Algo [BFS/Extrema]:

for ( $i=1$ ;  $i \leq n$ ;  $i++$ )  
{  $A[i][1..n]$  Dijkstra( $n, n, e, i$ ) }  
source

$$TC = (n^2 \times n \cdot \log n)$$

### # DP All pair shortest path Algo: [Floyd Warshall's Algo]

$A_{ij}^0 = \text{cost}(i, j)$  // Edge cost  
Initial value.  
 $A_{ij}^0 = \min\{0 \rightarrow 1 \text{ or } 1 \rightarrow 0\}$   
①  $A_{ij}^1 = \min\{A_{ij}^0, A_{i1}^0 + A_{1j}^0\}$  // shortest path from ① to ② going through ①.

②  $A_{ij}^2 = \min\{A_{ij}^1, A_{i2}^1 + A_{2j}^1\}$   
 $A_{ij}^2 = \min\{0 \rightarrow 2, 2 \rightarrow 0, 0 \rightarrow 1, 1 \rightarrow 0, 1 \rightarrow 2, 2 \rightarrow 1\}$

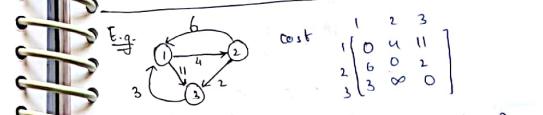
// shortest path cost from ① to ② going through ① and ②.

$$n \rightarrow A_{ij}^n = \min\{A_{ij}^{n-1}, A_{in}^{n-1} + A_{nj}^{n-1}\}$$

### # Recurrence Relation:

$$A_{ij}^n = \min_{1 \leq k \leq n} \{ A_{ik}^{n-1} + A_{kj}^{n-1} \}$$

$$A_{ij}^0 = \text{cost}(i, j)$$
 // Edge costs.



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

$$A^2 = \begin{bmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \quad \text{Result of All pair shortest path.}$$

### Algo AllSP - Floyd Warshall's ( $n, \text{cost}[ ][ ]$ )

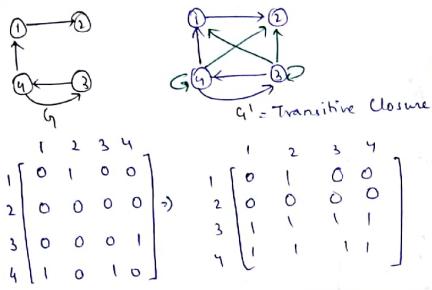
{ for ( $i=1$ ;  $i \leq n$ ;  $i++$ )  
{ for ( $j=1$ ;  $j \leq n$ ;  $j++$ )  
     $A_{ij} = \text{cost}(i, j);$  } }

{ for ( $k=1$ ;  $k \leq n$ ;  $k++$ )  
{ for ( $i=1$ ;  $i \leq n$ ;  $i++$ )  
{ for ( $j=1$ ;  $j \leq n$ ;  $j++$ )  
    { if ( $A_{ij} > A_{ik} + A_{kj}$ )  
        {  $A_{ij} = A_{ik} + A_{kj};$  } } } } }

$$TC = \Theta(n^3)$$

$$SC = \Theta(n^2)$$

### # Transitive closure of graph:



Algo TCh - FloydWarshall's(n, AdjL[]){}

```

    { for (i=0; i<=n; i++)
    { for (j=1; j<=n; j++)
        Adjj = Adji[i][j];
    }
    for (k=1; k<=n; k++)
    { for (i=1; i<=n; i++)
        { for (j=1; j<=n; j++)
            { Adjj = Adjij ∨ (Adjik ∧ Adjkj)
            }
        }
    }
    return (AdjL[])
}

```

$$\begin{aligned} A_{ij}^0 &= \text{Adj}_{ij} \\ \text{① } A_{ij}^1 &= \begin{cases} 1 & \text{if } \text{Adj}_{ij} \\ 0 & \text{otherwise} \end{cases} \\ \text{② } A_{ij}^2 &= A_{ij}^0 ∨ (A_{ii}^0 \wedge A_{jj}^0) \\ A_{ij}^3 & \text{is } \sum \text{ if there exist path from } i \text{ to } j \text{ going through } k \\ \text{③ } A_{ij}^4 &= \begin{cases} 1 & \text{if } \text{Adj}_{ij} \\ 0 & \text{otherwise} \end{cases} \\ A_{ij}^5 &= A_{ij}^4 ∨ (A_{i1}^4 \wedge A_{1j}^4) \\ \vdots & \\ n \Rightarrow A_{ij}^n &= A_{ij}^0 ∨ (A_{i1}^{n-1} \wedge A_{1j}^{n-1}) \end{aligned}$$

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

### # 0/1 Knapsack Problem:

→ Given 'n' objects and m capacity knapsack & p<sub>1</sub>..p<sub>n</sub> profits w<sub>1</sub>..w<sub>n</sub> weights

$$\text{Maximize } \sum_{i=1}^n p_i x_i$$

subjected to  $\sum_{i=1}^n w_i x_i \leq m$ ; where  $x_i = 1$  or  $0$ .

\* 0/1 knapsack problem failed to solve using greedy method.

\* 0/1 knapsack problem TC using Brute Force Algo: O(2<sup>n</sup>)

### # DP Solution of Knapsack Problem:

v(n, m) = Max profit for n objects and m capacity knapsack.

$$v(n, m) = \max \{ v(n-1, m), v(n-1, m-w_n) + p_n \}$$

$\leftarrow x_n = 0 \quad \leftarrow x_n = 1$

v(i, j) = Max profit for i objects & j knapsack capacity.

$$v(i, j) = \begin{cases} 0 & ; i=0 \& j \geq 0 \\ v(i-1, j) & ; i>0 \& w_i \geq j \text{ // weight of object more than knapsack free space.} \\ \max \{ v(i-1, j), v(i-1, j-w_i) + p_i \} & ; i>0 \& w_i \leq j \end{cases}$$

v(n, m) = final(v(0, m))

E.g. n=3 m=6

$\langle p_1, p_2, p_3 \rangle = \langle 20, 10, 15 \rangle$

$\langle w_1, w_2, w_3 \rangle = \langle 2, 1, 4 \rangle$

. define v[0..n, 0..m]

. Initialize 1st row 0's.

$$v(1, 0) = v(0, 0) \quad w_1 > 1$$

$$v(1, 1) = \max \{ v(0, 1), v(0, 0) + 20 \}$$

$$v(2, 1) = \max \{ v(1, 1), v(1, 0) + 10 \}$$

$$v(2, 2) = \max \{ v(1, 2), v(1, 1) + 10 \}$$

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	20	20	20	20	20
2	0	10	20	-	-	-	-
3	0	-	-	-	-	-	-

Algo 01knapsack ( $n, m, w[1..n], p[1..n]$ )

{  $\text{lv}[0..n, 0..m]$  define array }

$\{0^m\} \left[ \text{for } (i=0; i \leq m; i++) \right. \\ \left. \{ \dots [n] : j = 0 \} \right]$

for (i=1; i<=n; i++)

```
for (j=0; j < m; j++)  
    if (w[j] > i)
```

return  $\{x[n]\}, v[n][m]\}$   
object seq.

\* TC of knapsack using DP = ~~O(n)~~  $O(n \times m)$

\* SC of knapsack using DP =  $\Theta(n \cdot m)$

TC : O(1) Knapsack  
 ↓  
 Brute Force  $O(2^n)$       DP  $O(ntm)$

$n=10$ & $m \geq 1000$  $n = 20$ $m = 100000$	$10^2$ 4 Corp.  $2^{20}$	$10000$  $200000$ ✓
---	--------------------------------	---------------------------

### # Sum of subset Problem: (SOS)

$[a_1, a_2, a_3, \dots, a_n] : n$  integers.

Test any subset of given  $n$  integers sum equal to "m" exists/not exists.

$$\text{e.g. } n=4 \quad \{a_1, a_2, a_3, a_4\} = \{5, 4, 8, 6\} \quad \Delta m = 16$$

$\sigma_1(f, m) = \text{False}$

$m=15 \Rightarrow \text{sos}(n, m) = \text{True}$

→ Somebody method fails to solve SOS Problem.

→ Brute Force Algo to solve SOS problem  $\rightarrow \Theta(2^n) TC$

$\rightarrow$  DP sol<sup>n</sup> of SOS:

$Sos(n,m) = \begin{cases} \text{True} & \text{if any subset of } n \text{ elements sum equal to } m \text{ exists} \\ \text{False} & \text{if no such subset exists.} \end{cases}$

$$soc(n,m) = soc(n-l,m) \vee (soc(n-l,m-a_n) \wedge m \geq a_n)$$

$$sos(i, j) = \begin{cases} \text{True} & , i=0 \wedge i \geq 0 \\ \text{False} & ; j > 0 \wedge i \neq 0 \\ sos(i-1, j) & ; j < a_i \\ sos(i-1, j) \vee (sos(i-1, j-a_i) \wedge j \geq a_i) & \end{cases}$$

$$\text{E.g. } n=3 \qquad m=7$$

$$[m \quad 0 \quad q_2] = [3 \quad 2 \quad 5]$$

	0	1	2	3	4	5	6	7
0	T	F	F	F	F	F		F
1	T	F	F	T	F	F		F
2		T	F	T	T	F	T	F
3		T	F	T	T	F	T	F

• first column initialize True.

first row  $\{f_0\}[1..n]$  false.

$$\begin{aligned} \text{sos}(1,2) &= \text{sos}(0,2) & j < a_2 \\ \text{sos}(2,2) &= \underline{\text{sos}(1,2)} \cup \underline{\text{sos}(1,0)}, \end{aligned}$$

```
Alg0 sos(n,m,a[1...n])  
{ 1) sos [0...n, 0...m] define array.  
   for ( i=0; i < n; i++)  
     sos[i][0] = True  
   for ( i=1; i < m; i++)  
     sos[0][i] = False  
   for ( i=1; i < n; i++)  
     for ( j=0; j < m; j++)  
       if ( a[i] > j )  
         sos[ i ][ j ] = sos[ i - 1 ][ j ];  
       else  
         sos[ i ][ j ] = sos[ i - 1 ][ j ] || sos[ i - 1 ][ j - a[i] ]  
   }  
   return ( sos[n][m] )  
 }
```

TC	$\oplus$ sos = $O(n \times m)$
SC	$\oplus$ sos = $O(n \times m)$