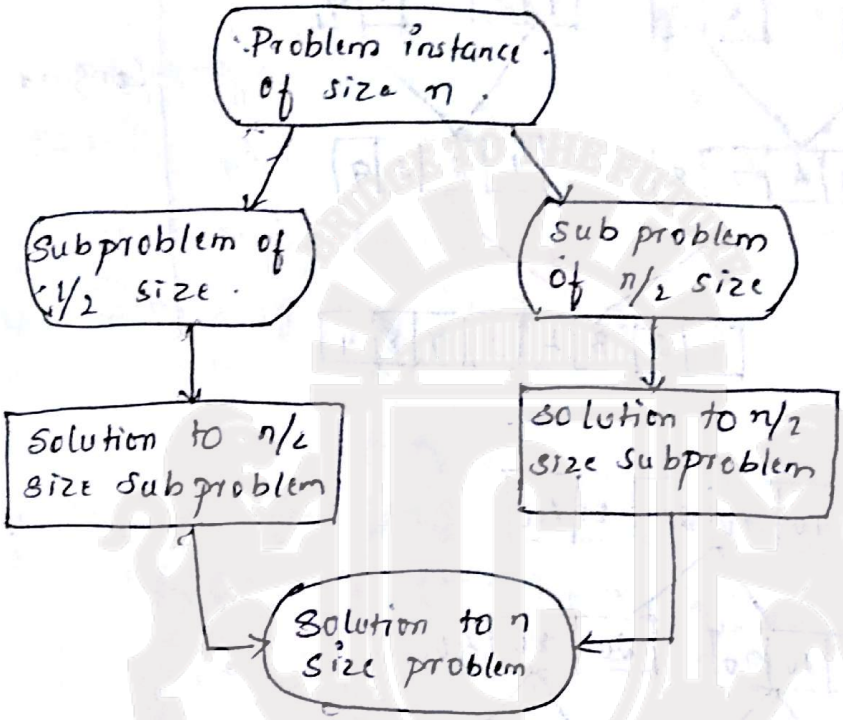
MODULE - 2

DIVIDE AND CONQUER

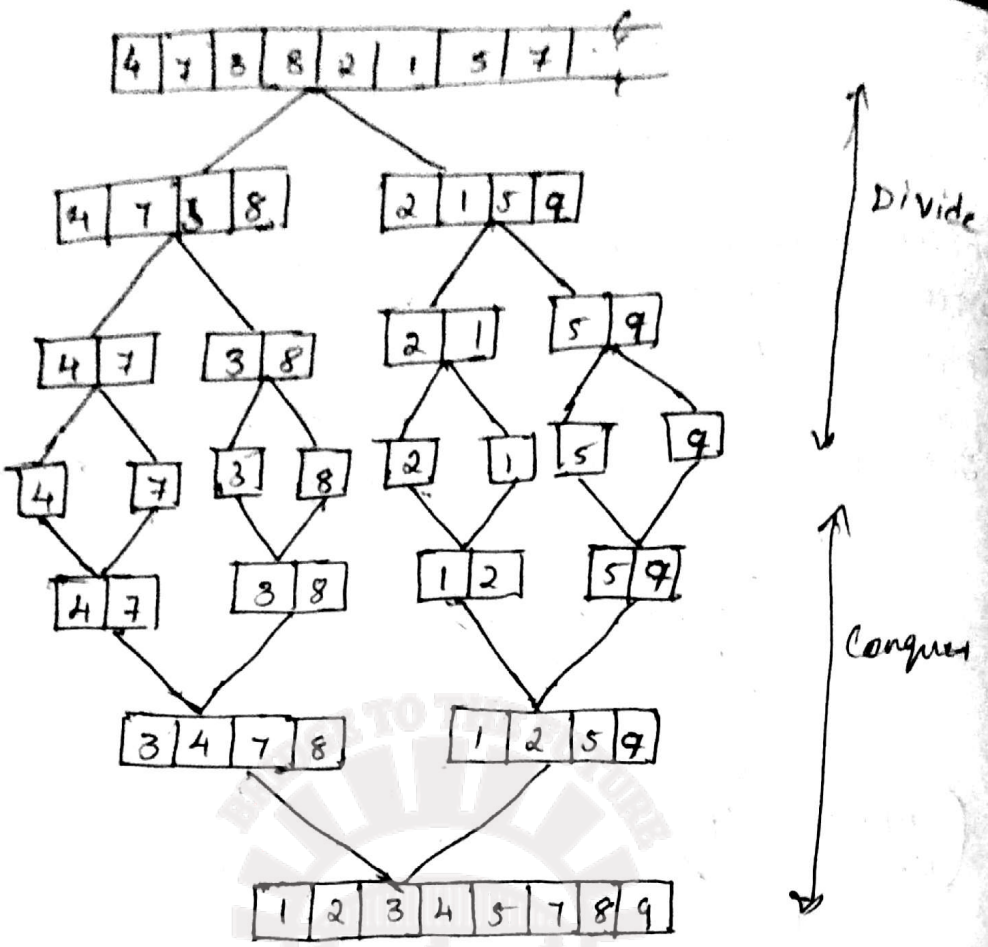
1. Problem is divided into sub-problems [until no further division is required] → Divide.
2. Solve the sub-problem with known method [simple] } conquer.
3. If required, combine the solution of the sub-problem instances to find the solution of overall problem }



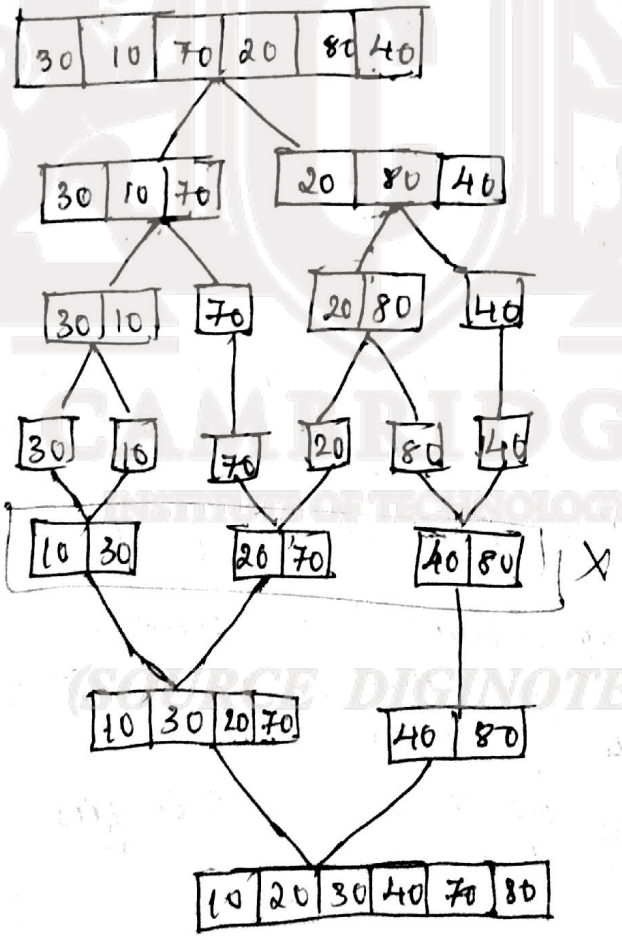
Merge Sort

→ The given list is considered to be as unsorted file.
→ Divide the file into two parts [approximately equal]
→ continue dividing until file reaches to a non divisible state. [one element file]
→ One element file by default is considered as sorted file
→ Combine the sorted files through simple comparision blw the elements of files.
→ The combineding of two files satisfies conquer part.

Ex 1



Ex 2



```
Algorithm  merge_Sort (A[0,... n-1])
//Input : An array A[0,.... n-1] of n oderable elements
//output: Array A[0, .... n-1] of ordered elements
  if (n > 1) Then
      copy  A[0,... ⌈n/2⌉] to B[0,..... ⌈n/2⌉]
      copy  A [⌈n/2⌉+1 .... n-1] to C[0.... ⌈n/2⌉]
       mergesort (B[0.... ⌊n/2⌋])
       mergesort(C[0.... ⌈n/2⌉])
        merge (B, C, A)
   end if

Algorithm  merge (B[0.... p-1], C[0....., q-1], A[0..... p+q-1]
//Input : Sorted array B[] and C[]
//output: sorted array A[].
  i ← 0, j ← 0, k ← 0
  while (i < p and j < q)
        if (B[i] < C[j]) then
            A[k] ← B[i]
              i ← i+1
        else
            A[k] ← C[j]
              j ← j+1
        end if
          k ← k+1
   end while
   if (i < p)
        copy B[i..... p-1] to A[k..... p+q-1]
   else
     copy C[j.... q-1] to A[k ..... p+q-1]
   end if
```

16/3/17

Analysis

1. Input parameter → 'n' size of input array.
2. Basic operation → comparison $B[i] < C[j]$
3. Minimal variation (within same order of growth)
4. Recurren relation Based on B.O.

$$C(n) = \begin{cases} 0 & n = 1 \\ C(n/2) + C(n/2) + n - 1 & n > 1 \end{cases}$$

5. $C(n) = 2C(n/2) + n - 1$ until $C(1) = 0$

upper bound.

- [Backward substitution moving from n to 1]

consider $n = 2^k$

$$C(n) = 2 \cdot C(2^k/2) + 2^k - 1$$
$$= 2 \cdot C(2^{k-1}) + 2^k - 1$$
$$= 2 \cdot [2 \cdot C(2^{k-2}) + 2^{k-1} - 1] + 2^k - 1$$

$$= 2[2 \cdot [2C(2^{k-3}) + 2^{k-2} - 1] + 2^{k-1} - 1] + 2^k - 1$$
$$= 2^2 C(2^{k-2}) + 2^k - 2 + 2^k - 1$$
$$= 2^2 C(2^{k-2}) + 2 \cdot 2^k - 2^1 - 2^0$$
$$= 2^3 C(2^{k-3}) + 3 \cdot 2^k - 2^2 - 2^1 - 2^0$$

$$= 2^k \cdot C(2^{k-k}) + k \cdot 2^k - 2^{k-1} - 2^{k-2} \ldots - 2^1 - 2^0$$

($C(2^{k-k}) \to C(0)$)

$$= k \cdot 2^k - [2^{k-1} - 2^{k-2} \ldots - 2^1 - 2^0]$$

Sum of n^{th} term of Geometric series

$$S_n = \frac{a r^n - 1}{r - 1}$$

$$= *.$$

$= k \cdot 2^k - \left[\frac{1 \cdot 2^k - 1}{2-1}\right]$

$= k \cdot 2^k - [1 \cdot 2^k - 1]$

$= k \cdot 2^k - [2^k - 1] = k \cdot 2^k - 2^k + 1$

$= \boxed{2^k[k-1] + 1}$ // $\approx 2^k \cdot k$.

upper bound

Apply $\log_2$ on both sides $n = 2^k$

$\log_2 n = \log_2(2^k) = k \log_2 2$

$= k$.

$C(n) \approx n \cdot \log_2 n$

$\boxed{c(n) \in \Theta(n \log_2 n)}$

Theoretical upper bound

$C(n) = n \cdot [\log_2 n - 1] + 1$

$n = 10$

$10[\log_2 10 - 1] + 1$

$= 24.21 \approx 25$ //

To get the lower bound.

$C(n) = 2C(n/2) + n/2$ until $C(1) = 0$

$n = 2^k$

$= 2\{C(2^k/2) + 2^k/2$

$= 2[2C(2^{k-2}/2) + 2^{k-1}/2] + 2^k/2$

$= 2^2 C[2^{k-2}/2] + 2^{k-1} + 2^k/2$

$= 2[2C(2^{k-2}) + 2^{k-2}] + 2^{k-1}$

$= 2^2 \cdot C[2^{k-2}] + 2^{k-1} + 2^{k-1}$

$= 2^2 C[2^{k-2}] + 2[2^{k-1}]$

$= 2^3 \cdot C[2^{k-3}] + 3[2^{k-1}]$

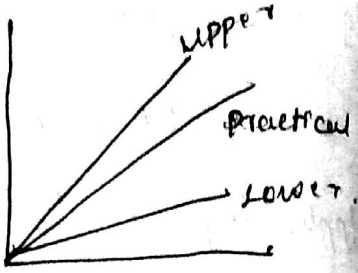
!

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

$= k(2^{k-1})$

$= K \frac{2^k}{2}$.

$$C(n) = \frac{n \log_2 n}{2}.$$ lower bound.



$n = 10$

$C(n) = 16.6$

≈ 17.

Masteris Theorem

If there is Recurrence relation

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

Where a -- no of subproblems to be solved

$b \rightarrow$ no. of fractions of input size n

$f(n)$ = time consumed for Divide / conquer

d = degree of n in $f(n)$.

then,

$$T(n) \in \begin{cases} \theta(n^d) & \text{if } a < b^d \\ \theta(n^d \log_b n) & \text{if } a = b^d \\ \theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Ex1 for Merge sort $C(n) = 2 \cdot C(n/2) + n - 1$

According to masteris theorem.

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

$a = 2,\ b = 2,\ f(n) = n-1,\ d = 1$

Relationship b/w a b^d

$2 = 2^1$

$\therefore$ $C(n) \in \theta(n \log_2 n)$.

It is applicable for divide and conquer algorithm.

Ex2 $C(n) = 2 \cdot C(n/2) + n/2$

According to masteris theorem.

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

$a = 2,\ b = 2,\ f(n) = n/2,\ d = 1$

relationship b/w a b^d

$2 = 2^1$

$$C(n) \in \theta(n^1 \log_2 n)$$

Example 3. $T(n) = 3 \cdot T(n/4) + n^2$

$a = 3,\ b = 4,\ f(n) = n^2,\ d = 2$

relationship b/w a b^d

$3 < 4^2$

$$C(n) \in \theta(n^2).$$

Example 4: $T(n) = T(n/2) + 1$

$a = 1,\ b = 2\ \ f(n) = 1\ \ d = 0$

relationship b/w a b^d

1 $2^0 = 1$

$$C(n) \in \theta(n^0 \log_2 n)$$

Ex 5: $T(n) = 4T(n/2) + n$

$a = 4,\ b = 2,\ f(n) = n,\ d = 1$

a b^d

$4 > 2$

$$T(n) \in \theta(n \log_2 4)$$

$$T(n) \in \theta(2n)$$

Ex 6 $T(n) \in 5T(n/2) + n$

$a = 5,\ b = 2,\ f(n) = n,\ d = 1$

a b^d

$5 > 4$

$$T(n) \in \theta(n^{\log_b a})$$

$$T(n) \in \theta(n^{\log_2 5}).$$

$$T(n) = \theta(n^{2.3}).$$

$$C(n) \in \theta(n^{2.3}).$$

Quick sort

→ One of the divide and conquer algorithm
→ partition results in un-even sub groups.
→ partition occurs based on pivot / Anchor element.
→ left group < pivot element & .
pivot < Right group

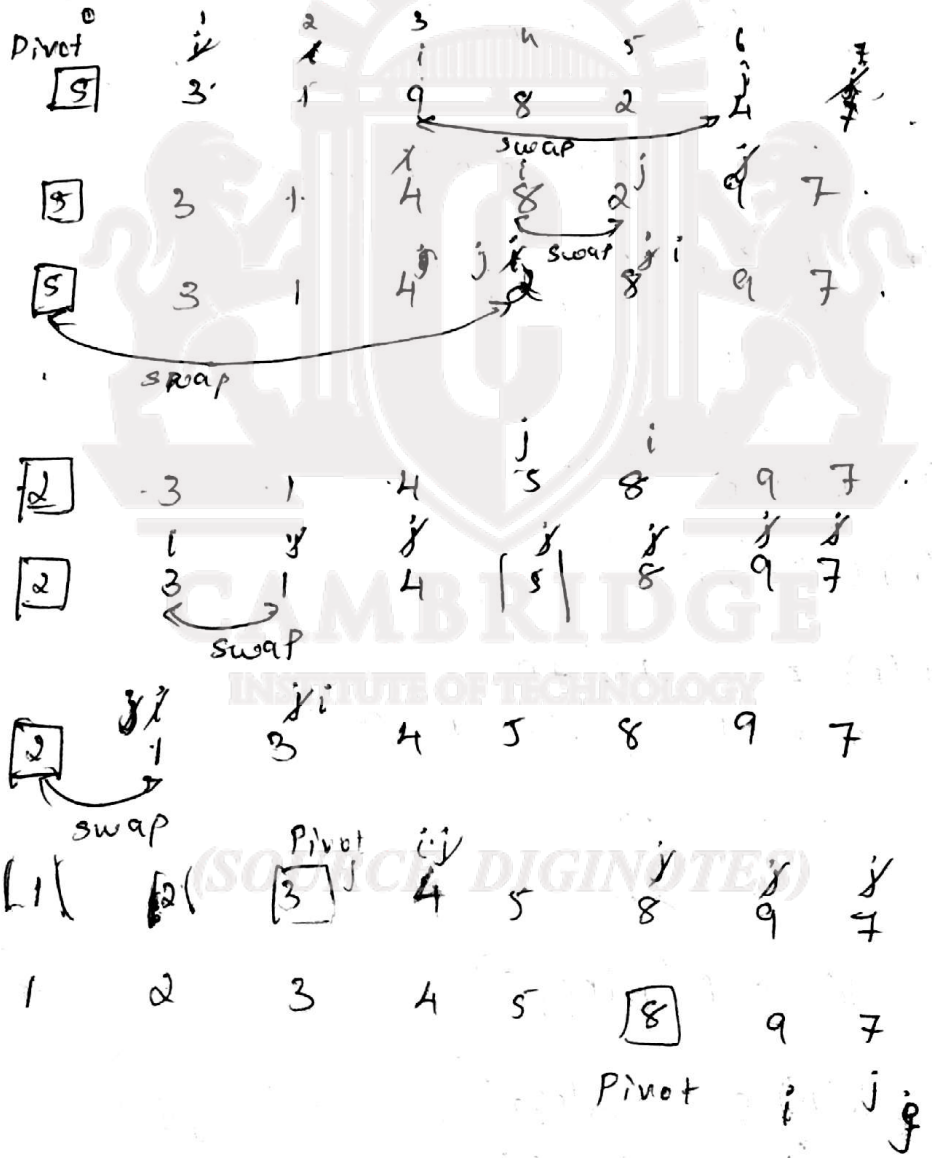
Left group < pivot < Right group.

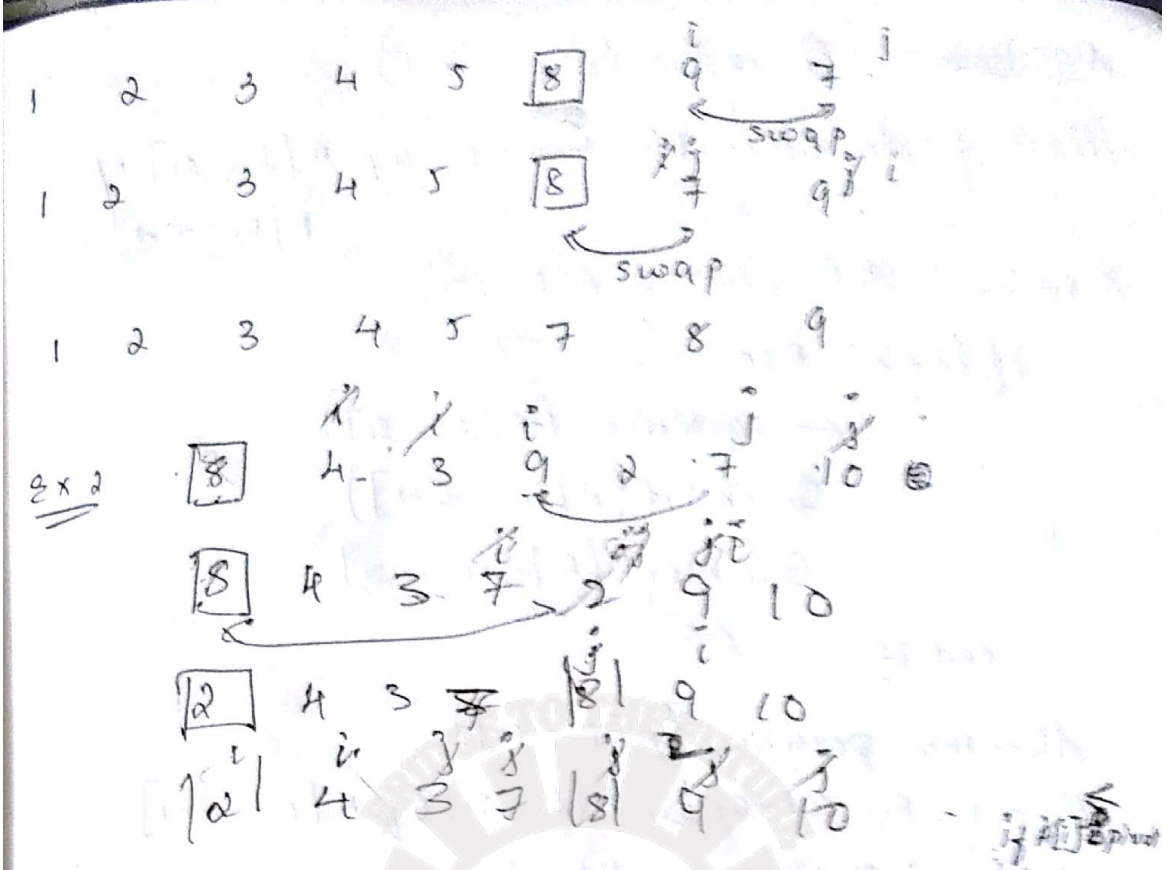
Ex & [5] 2 4 7 6 — p.

2 4 |5| 7 6.

17 | 3 | 17

Ex:






```
if A[i] <= pivot
i++
if A[j] >= pivot
j--
if (i<j) then
Swap(A[i], A[j])
if (i>j) then
Swap(pivot, A[j])
if (i=j)
```

Algorithm Quicksort (A[l...r])

//Input : An Orderable ~~Sub~~ -array A[l...r] of A[0...n-1]

// output : Sorted array A[l...r]

```
If (l < r) then
        s ← partition (A[l...r])
        Quicksort (A[l...s-1])
        Quicksort (A[s+1...r])
end if.
```

Algorithm partition (A[l...r])

//Input : An sub-array A[l...r] of A[0...n-1].

//output : partition position j

```
p ← A[l]
i ← l+1
j ← r
repeat
    repeat i ← i+1 until A[i] ≥ p
    repeat j ← j+1 until A[j] ≤ p.
    swap (A[i], A[j]) // last swap is Invalid
until i > j
    swap (A[i], A[j]) // to avoid Invalid swap
    swap (A[l], A[j])
    return j
```


2.

Time Analysis.

1) Input parameter - 'n' size of array A.
2) Basic operation - $A[i] >= P$ $A[j] \leq P$ - comparision.
3) There is variation of time for the same size 'n'.
∴ find Bestcase, worstcase and averagecase individually

4. **Best case :-**

Partition results in approximately equal sub-groups

ex:-

5 2 4 7 8

2 4 [5] 7 8

$n/2$ $n/2$

$\therefore C_B(n) = C_B(n/2) + C_B(n/2) + (n+1)$

↓ min. no. of comparision.

$C_B(n) = 2(C_B(n/2)) + (n+1)$

until $C_B(1) = 0$

5) Applying Backward substitution method

$n = 2^k$

$C_B(n) = 2 \cdot C_B(2^k/2) + (2^k+1)$

$= 2 \cdot C_B(2^{k-1}) + (2^k+1)$.

$= 2(2 \cdot C_B(2^{k-2}) + 2^{k-1}+1) + 2^k+1$

$= 2^2 C_B(2^{k-2}) + 2^k + 2 + 2^k + 1$

$= 2^2 C_B 2^{k-2} + 2 \cdot 2^k + 2^1 + 2^0$

$= 2^3 C_B 2^{k-3} + 3 \cdot 2^k + 2^2 + 2^1 + 2^0$

$= 2^k C_B(2^{k-k}) + k \cdot 2^k + 2^{k-1} + \ldots + 2^1 + 2^0$ (i → 0)

$= k \cdot 2^k + 2^k + 2^{k-1} + \ldots + 2^1 + 2^0$

$C_B(n) = k \cdot 2^k - \frac{[1 \cdot 2^k - 1]}{2-1}$ $\left\{ S_n = \frac{ar^n - 1}{r-1} \right.$

$C_B(n) = k \cdot 2^k - [2^k - 1]$.

$2^k[k+1] + 1$

$C_B(n) \approx 2^k[k]$ (for larger input value)

Apply $\log_2$ on both sides of $n = 2^k$

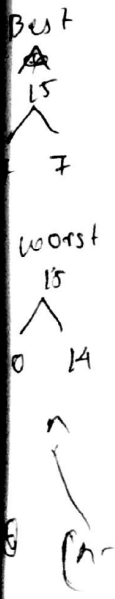
$\log_2 n = k \log_2 2$

$k = \log_2 n$

$\log_2(2^k) = k$

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

$\boxed{C_B(n) \in \Omega(n \log_2 n)}$ lower bound.



Worst case :-

If the partition creates one heavy group and the other empty group

$C_W(n) = C_W(0) + C_W(n-1) + (n+1)$ until $C_W(1) = 0$

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

$[C_W(n-2) + (n-1+1)] + n + 1$

$C_W(n-2) + n + (n+1)$

$C_W(n-3) + (n-2+1) + n + (n+1)$

$C_W(n-3) + (n-1) + n + (n+1)$

$C(0) + C(n+1)^{n-1}$

$C_W(n - (n-1)) + 3 + 4 + 5 + \ldots + n + (n+1)$

$C_W(n) = 3 + 4 + 5 + \ldots + n + (n+1)$.

$= [1 + 2 + 3 + 4 + 5 + \ldots + n + (n+1)] - 3$

↳ Arithmetic progression.

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

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

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

$$C_w(n) = \frac{n^2+3n+2-3}{2}$$

$$= \frac{n^2}{2} + \frac{3n}{2} + 1 - 3$$

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

For large value of Input (n)

$C_w(n) \approx n^2/2$

$\boxed{C_w(n) \in \Theta(n^2)}$ Quadratic Order of growth

Average case

$C_A(n) \approx 2n\log_e n$. $\quad 2n\log_e n$

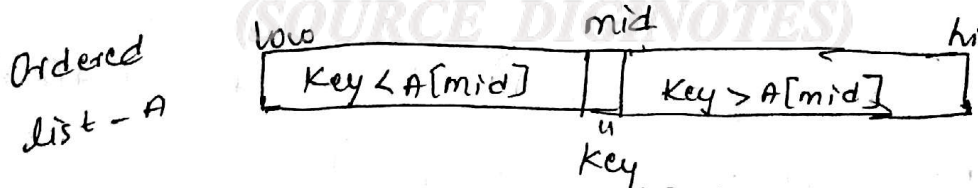
$C_A(n) \approx 1.33\, n\log_e n$

i.e 33 % more than Best case.

$\boxed{C_A(n) \in \Theta(n\log_e n).}$

18/3/17

Binary search



mid = (low + high)/2 ;

Algorithm Binary_search (A[low.....high], key).

//Input : An ordered array A[low.....high] of 'n' elements and search element - key.

//output : return 1 if key is found else 0.

```
if (low ≤ high) then
    mid ← (low + high) / 2
    if (key = A[mid]) then
        return (1)
    else if (key < A[mid]) then
        return (Binary_search (A[low....mid-1], key))
    else
        return (Binary_search (A[mid+1....high], key))
    end if
else
    return (0)
end if
```

Time Analysis

1) Input parameter - n size of array A.

2) Basic operation - comparision ← key = A[mid]

3) No of comparison vary based on the position of key ∴ Estimate best / worst / Average case individually

4) Best case

$C_B(n) = 1$. found at mid.

$C_B(n) \in \Omega(1)$ constant order of growth

Worst case

$C_W(n) = C_W(n/2) + 1$ till $C_W(1) = 1$.

$n = 2^k$.

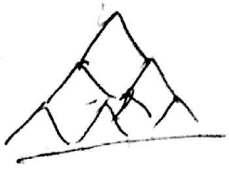
$C_W(n) = C_W(2^k/2) + 1$

$C_w(n) = C_w(2^{k-1}) + 1$

$= [C_w(2^{k-2}) + 1] + 1$

$= C_w(2^{k-2}) + 2.$

$= C_w(2^{k-3}) + 3.$



$\vdots$

$C_w(2^{k-k}) + k.$

$= 1 + k.$

$= k + 1$

$= n = 2^k$

= apply log on B.S. $\quad k = \log_2 n.$

$= \log_2 n + 1.$

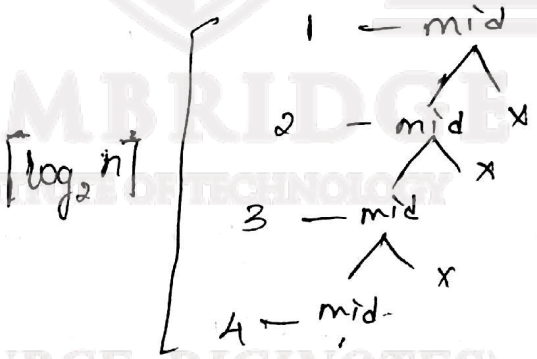
For large value of n:

$C_w(n) \approx \log_2 n.$

$\boxed{C_w(n) \approx O(\log_2 n)}$

Logarithmic order of growth.

Average case



if $n = 10$, $\lceil \log_2 10 \rceil = 4$

$\therefore C_a(n) = \frac{1}{\log_2 n} \sum_{i=1}^{\log_n 2} i$

$= \frac{1}{\log_2 n} [1 + 2 + 3 \cdots + \log_2 n]$

$$\sum_{i=1}^{\log_2 n} \frac{1}{\log_2 n} = \frac{\log_2 n(\log_2 n+1)}{2}$$

{ Arithmetic progression $\frac{n(n+1)}{2}$

$$C_A(n) = \frac{(\log_2 n+1)}{2}$$

$$C_A(n) \approx \tfrac{1}{2}\log_2 n$$

$$\boxed{C_A(n) \in \Theta(\log n)}$$

Maximum and minimum problem

Bruteforce technique

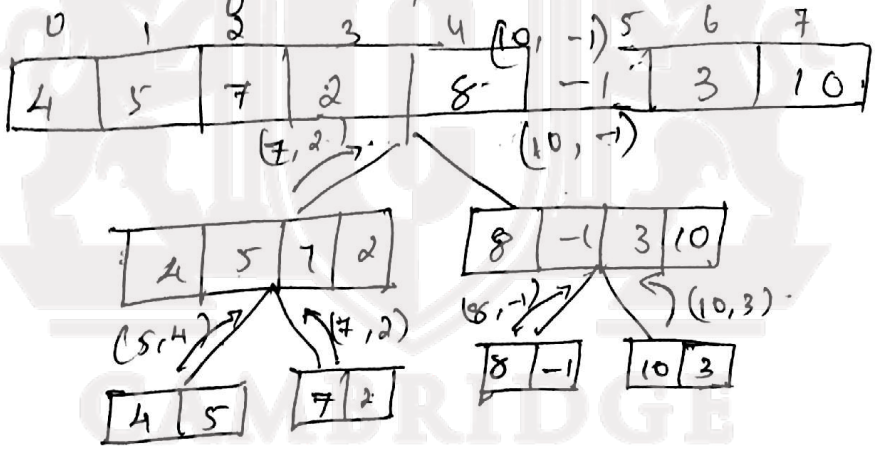
5 7 2 8 -1 3 10.

max: -999 5 7 8 10

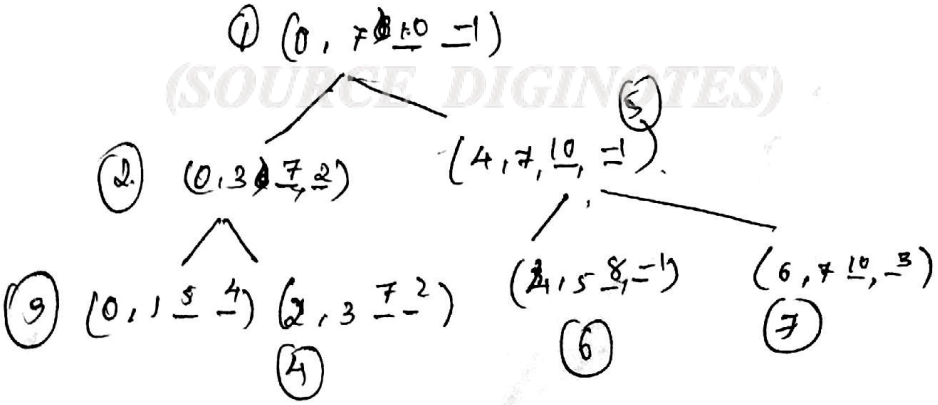
min: 999 5 2 -1 -1

2n no of comparision.

in divide and conquer technique, we reduce no of comparisim.



comparision = 10



(32, 17)

Ex: 20 17 10 25 32 11 7 22

(25, 10) (32, 17)

20 17 10 25 32 11 7 22

(20, 17) (25, 10) (32, 11) (22, 7)

20, 17 25, 10 32, 11 22, 7.

Algorithm: (i, j, max, min)
// input: limits of array i, j max min
// output: Maximum max and Minimum min
Values of array.

```
if (i=j) then,        // one element
    max ← min ← A[i]
else if ((j-i)=1)  then  // two element
    if (A[i] < A[j]) then
        max ← A[j], min ← A[i].
    else
        Max ← [A[i], min ← A[j]
    end if
else
    mid ← (i+j)/2.
    MaxMin(i, mid, max, min)
    MaxMin(mid+1, j, max1, min1)
    if (max1 > max) then
        max ← max1
    end if
    if (min1 < min) then
        min ← min1
    end if
end if
```

Time Analysis :-

1) Input parameter - no of elements bw i - j as n.

2) Basic operation - comparison
A[i] < A[j], max1 > max, min1 < min

3) Depending only on array size - 'n'
no variations

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

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

$n = 2^k$.

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

$= 2[2 \cdot T(2^{k-2}) + 2] + 2$.

$= 2^2 \cdot T(2^{k-2}) + 2^2 + 2$.

$= 2^3 T(2^{k-3}) + 2^3 + 2^2 + 2$.

$\vdots$

$2^k T(2^{k-k}) + 2^k + 2^{k-1} + \ldots + 2$

$T(2^{k-k}) \to T(2^0) \to T(1) \to 0$

if upto $n = 1$

Geometric progression but we are missin 2^0

$= 2(2^{k-1} + 2)$

$2(2^{k-1} + 2^{k-2} + \ldots + 2 + 2^0)$

$S_n = \frac{a r^n - 1}{r - 1}$

$= 2 \cdot \left[\frac{1 \cdot 2^k - 1}{2 - 1}\right]$

$= 2[2^k - 1]$

$= 2(n - 1)$

for a very large value of n n.

$T(n) \approx 2n$

$\boxed{T(n) \in \Theta(n).}$

Date :
Page No :

2. using Masters Theorem $T(n) = a \cdot T(n/b) + f(n)$

Algorithm : recurrance relation

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

$a = 2, \quad f(n) = 0, \quad b = 2, \quad d = 0.$

$a \cdot b^d$

$2 > 2^0$

$2 > 1$

$T(n) \in \theta(n^{\log_b a})$

$\in \theta(n^{\log_2 2})$

$\boxed{T(n) \in \theta(n)}$

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

Matrix multiplication :-

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

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

$$T(n) = \begin{cases} 1 & n = 1 \\ 8T(n/2) & n > 1 \end{cases}$$

$T(n) = 8T(n/2) + 0$ until $T(1) = 1$

$a = 8, \quad b = 2, \quad d = 0, \quad f(n) = 0.$

$a \quad b^d$

$8 > 2^0$

$T(n) \in \theta(n^{\log_b a})$

$\in \theta(n^{\log_2 8})$

$T(n) \in \theta(n^3)$

Strassen's matrix multiplication.

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

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

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

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

multiplication = 7.
Addition / subtraction = 18

21/8/17

Ex :-

$$A = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$P = 30$. $T = 20$
$Q = 5$ $U = -3$.
$R = -4$ $V = -7$
$S = 8$

$$C = \begin{bmatrix} P+S \overset{11}{-} T+V & R+T \overset{16}{} \\ Q+S \overset{13}{} & P+R-Q+U \overset{18}{} \end{bmatrix}.$$

~~V. (A12 - A22)~~

Analysis

1) Input parameter 'n' order of matrix n×n.
2) B.O. – multiplication.
3) Depends only on 'n'
4)

$$T(n) = \begin{cases} 1 & n = 1 \\ 7T(n/2), & n \geq 2 \end{cases}$$

5) $T(n) = 7T(n/2)$ until $T(1) = 1$

$a = 7, b = 2, f(n) = 0$ $d = 0$.

$a > b^d$

$7 > 2^0$

$7 > 1$

$T(n) \in \theta(n^{\log_b a})$.

$T(n) \in \theta(n^{\log_2 7})$

$T(n) \in \theta(n^{2.81})$

Decrease and conquer

The problem of i/p size n is decreased by.

→ constant value. (Ex. ~~n → n-1~~ $a^n \rightarrow a^{n-1}$)
→ constant factors (ex $a^n \rightarrow a^{n/2}$, $a^{n/2}$.
→ Decrease by variable size (Ex GCM (m, n%m)

→ constant value → BFS, DFS, Insertion sort, Topological sorting

Topological ordering :- / sorting

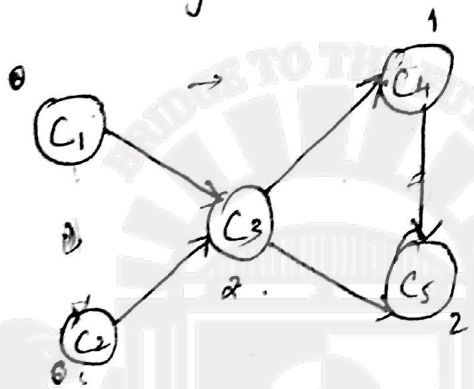
Its a graph traversal technique exclusively on DAG → Directed Acyclic technique.

C_1, C_2, C_3, C_4, C_5.

→ To attempt to C_3, either C_1 or C_2 should be complete

→ To attempt C_4, C_3 should be complete.

→ To attempt C_5, either C_4 or C_3 should be complete



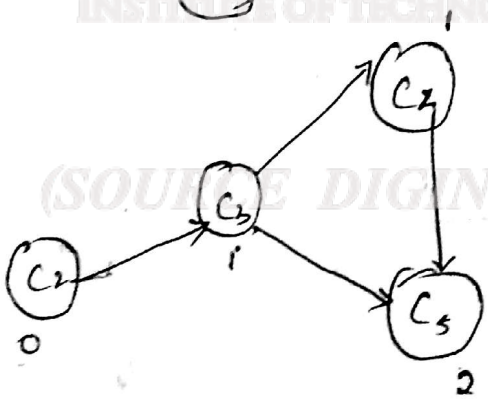
To solve Topological ordering

1) DFS method
2) Source removal method.

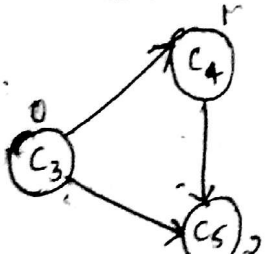
1. Source Removal method

source - vertex with indegree - zero.

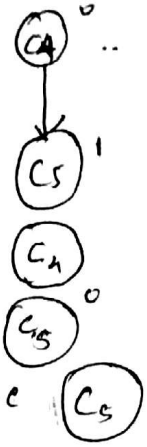
Remove C_1



Remove C_2



Remove C_3

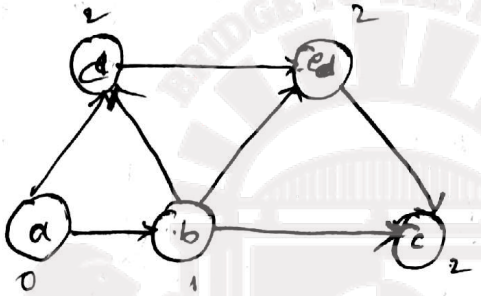


Remove C_4

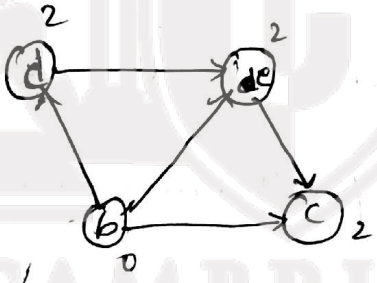
C_5 0

Remove C_5

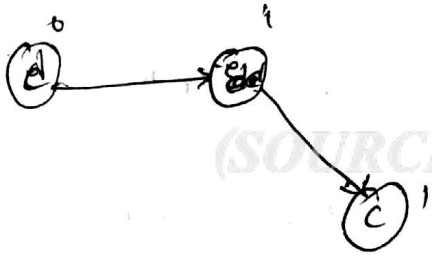
Ex 2 :-



Remove a



Remove b

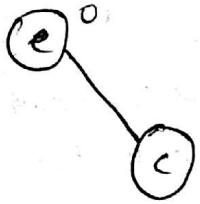


Remove d

Remove e.

e c

Remove c.

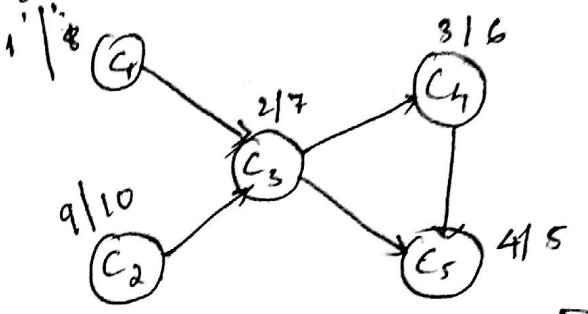


a b d e c topological order

Topological ordering using DFS method

entry/exit

(1)



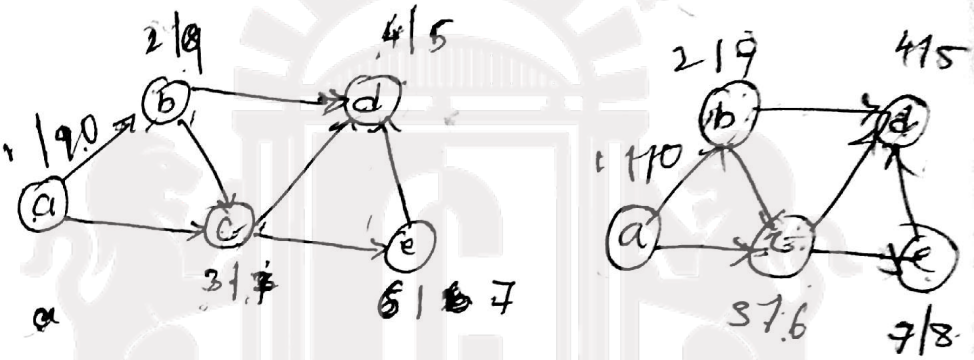
source clearance method

Entry time / Exit time

C_2 C_1 C_3 C_4 C_5

↓ topological order.

(2)



a b c e d. a b c e d

Control abstract for Divide and conquer

Type D and C (P)
{
  if Small(P) then ← Smallest enough to solve P by straight forward method
    return S(P) ⟶ solution of P
  else
    Divide problem ~~instance~~ instance
    P into P_1 P_2 P_3 ... P_k where $k \geq 1$
    solve each subproblem by D and C (P1),
    D and C (P2) .....
    return combine (D and C(P_1) ~~and~~
      D and C (P_2) ... D and C(P_k))
  end if.
}