MODULE-2
DIVIDE AND CONQUER

Divide and Conquer → Given a function to compute on n inputs the divide and conquer strategy suggests splitting the inputs into k distinct subsets, $1 < k \le n$, yielding k subproblems. These subproblems must be solved, and then a method must be found to combine subsolutions into a solution of the whole.

MERGE SORT analysis inbuilt → bubble, selection, insertion, counting } $n \log n$

We assume ~~that~~ throughout that the elements are to be sorted in nondecreasing order. Given a sequence of elements $a[1], \ldots, a[n]$, ~~the~~ it is split into 2 sets,

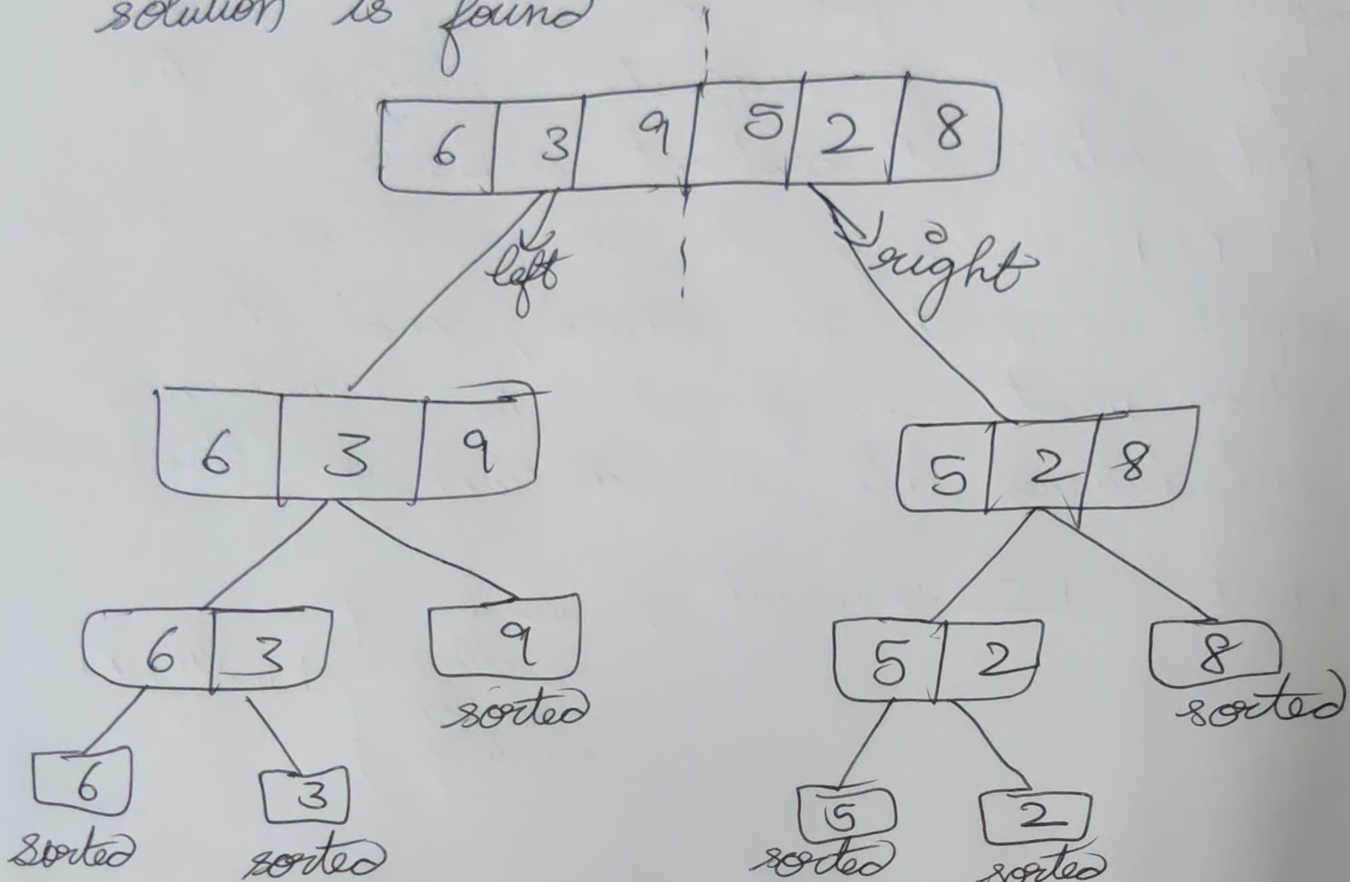
$$a[1] \ldots a\left[\frac{n}{2}\right] \text{ and } a\left[\frac{n}{2}+1\right], \ldots, a[n]$$

Each set is individually sorted and the resulting sorted sequences are merged to produce a single sorted sequence of n elements.

There are ~~8~~ 2 steps :-

6	3	9	5	2	8

① Divide → Divide the set into ~~the~~ subproblems and stop where the smallest sorted ~~num~~ solution is found

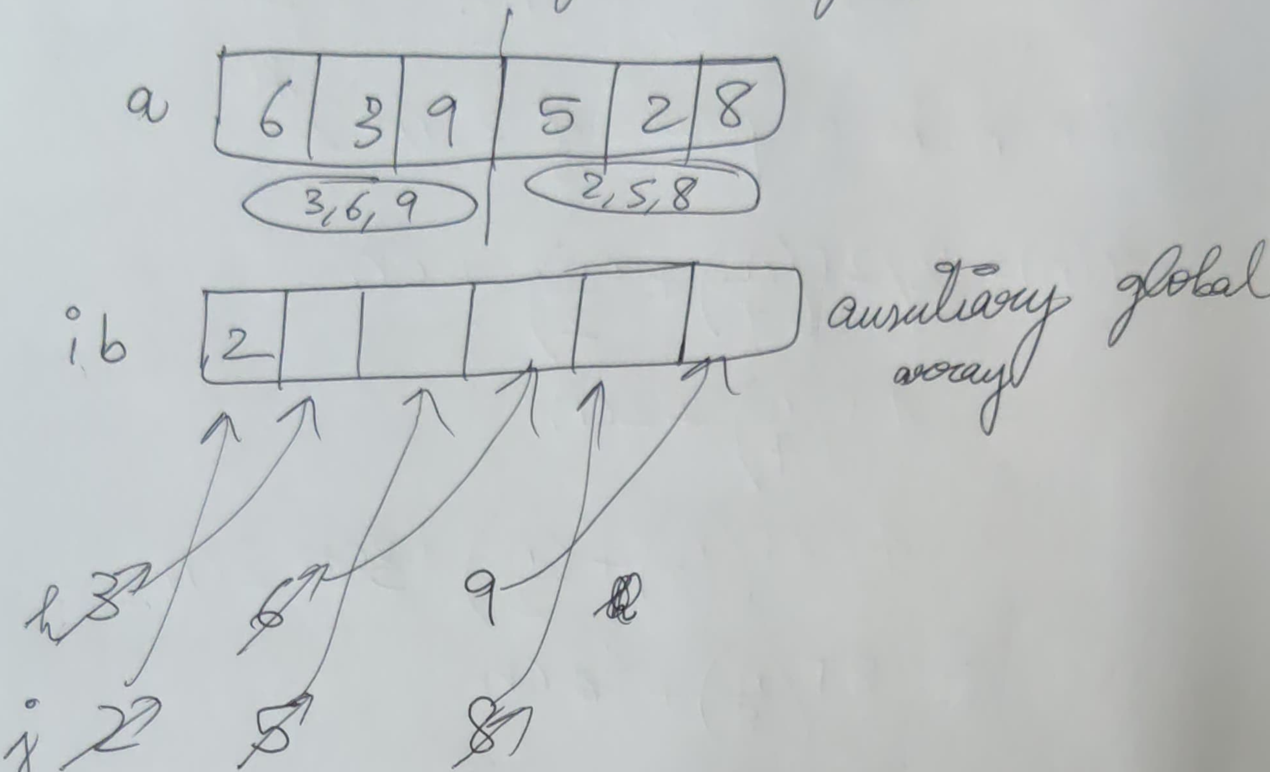


② mergeSort(left)
mergeSort(right)

ALGORITHM:-

```
ALGORITHM MergeSort(low, high)
{
  if(low<high)
  {
    mid = ⌊(low+high)/2⌋;
    MergeSort(low, mid);
    MergeSort(mid+1, high);
    Merge(low, mid, high);
  }
}
```

② Merge → Take ~~Merge~~ MergeSort(Left)
MergeSort(Right)



ALGORITHM:-

ALGORITHM Merge(low, mid, high)
{
h = low, i = low, j = mid+1;
while ((h ≤ mid) and (j ≤ high)) do
{
if (a[h] ≤ a[j]) then
{
b[i] = a[h]; h = h+1;
}
else
{
b[i] = a[j]; j = j+1
} } } i = i+1;

4

the computing time for merge sort :-

$$T(n) = \begin{cases} a, & n=1, \text{ a is a constant} \\ 2T\left(\frac{n}{2}\right) + cn & n>1, \text{ c a constant} \end{cases}$$

when n is a power of 2, $n = 2^k$

$$T(n) = 2\left(2T\left(\frac{n}{4}\right) + \frac{cn}{2}\right) + cn$$

$$= 4T\left(\frac{n}{4}\right) + 2cn$$

$$= 4\left(2T\left(\frac{n}{8}\right) + \frac{cn}{2}\right) + \cancel{3cn}\ 2cn$$

$$= 8T\left(\frac{n}{8}\right) + 3cn$$

$$= 2^k T(1) + kcn$$

$$= an + cn \log n$$

if $2^k < n \leq 2^{k+1}$, then $T(n) \leq T(2^{k+1})$

$$\boxed{T(n) = O(n \log n)}$$

~~Quick~~

QUICK SORT

↳ average $O(n \log n)$
worst $O(n^2)$
space $O(1)$

Worst case occurs when pivot element is always the smallest or the largest element.

~~1~~	~~2~~

6	3	9	8	2	5

~~pivot = 6~~

6	3	9

8	2	5

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

pivot = 4
~~&~~ all elements < 4 => left

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

↓ partition

1	3	2	4	5	6	7

left of 4   right of 4

1	2	3	4	7	5	6

left of 2   right of 2   right of 4

left of 4

all sorting ⇒ last element is the pivot.

1	2	3	4	5	6	7

$5 \leq 6$, $7 \geq 6$

a. (i)

Steps :-

(1) ~~Partition~~ Quick Sort

```
ALGORITHM QuickSort(p, q)
{
  if(p<q)
  {
     j = Partition(a, p, q+1);
     QuickSort(p, j-1);
     QuickSort(j+1, q);
  }
}
```

② Partition


```
Algorithm Partition (a, m, p)
{
   v = a[m];
   i = m;
   j = p;
   while (i ≤ j)
    {
       while (a[i] ≥ v)
        {
           i = i + 1;
        }
       while (a[i] ≤ v)
        {
          j = j - 1;
        } if (i < j), interchange (a, i, j);
    }
   a[m] = a[j];
   a[j] = v
   return j;
Algorithm Interchange (a, i, j);
{
    p = a[i];
   a[i] = a[j];
    a[j] = p;
}
```

Finding the maximum and minimum

Using direct method:-

~~Al~~ Algorithm StraightMaxMin (a, n, max, min)

```
{
    max = a[1];
    min = a[1];
    while (i ≥ 2)
    {
        if (a[i] > max) then max = a[i];
        if (a[i] < min) then min = a[i];
    }
}
```

Time complexity

$$T(n) = \begin{cases} T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + 2 & n > 2 \\ 1 & n = 2 \\ 0 & n = 1 \end{cases}$$

when n is a power of 2, $n = 2^k$

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

$$= 2\left(2T\left(\frac{n}{4}\right) + 2\right) + 2$$

$$= 4T\left(\frac{n}{4}\right) + 4 + 2$$

$$= 2^{k-1}T(2) + \sum_{1 \le i \le k-1} 2^i$$

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

(2) Using Divide and Conquer

Algorithm MaxMin(i, j, max, min)
{
if(i=j) then max=min=a[i];
else if(i=j-1) then
{
if(a[i]<a[j]) then
{
max=a[j];
min=a[i];
}
else
{
max=a[i];
min=a[j];
}
}
else
{
mid=(i+j)/2;
MaxMin(i, mid, max, min);
MaxMin(mid+1, j, max1, min1);
if(max<max1) then max=max1;
if(min>min1) then min=min1;
}
}

Analysis:-

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

$$= 4c\left(\frac{n}{4}\right) + 6 + 3$$

$$= 2^{k-1} C(2) + 3 \sum_{0}^{k-2} 2^i$$

$$= 2^k + 3 * 2^{k-1} - 3$$

$$\boxed{c(n) = \frac{5n}{2} - 3}$$

STRASSEN'S MATRIX MULTIPLICATION

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

$P_1 = A * (F - H)$

$P_2 = H * (A + B)$

$P_3 = E * (C + D)$

$P_4 = D * (\cancel{E}G - E)$

$P_5 = (A + D) * (E + H)$

$P_6 = (B - D) * (G + H)$

$P_7 = (A - C) * (E + F)$

$$\text{Matrix} = \begin{bmatrix} P_6 + P_5 + P_4 - P_2 & P_1 + P_3 \\ P_2 + P_4 & P_1 - P_3 + P_5 - P_7 \end{bmatrix}$$

Time complexity for strassen's

$$T(n) = \begin{cases} b & n \le 2 \\ 7T\left(\frac{n}{2}\right) + an^2 & n > 2 \end{cases}$$

$$T(n) = an^2\left[1 + \frac{7}{4} + \left(\frac{7}{4}\right)^2 + \cdots + \left(\frac{7}{4}\right)^{k-1}\right] + 7^k T(1)$$

$$\le cn^2\left(\frac{7}{4}\right)^{\log_2 n} + 7^{\log_2 n}$$, c a constant

$$= cn^{\log_2 4 + \log_2 7 - \log_2 4} + n^{\log_2 7}$$

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

BINARY SEARCH

Algorithm:-

```
Binary Search (a, n, x)          O(log n)
{
  low = 1;
  high = n+1;
  while (low < (high - 1)) do
  {
    mid = [low + high]/2;
    if (x < a[mid]) then high = mid;
    else low = mid;
  }
  if (x = a[low]) then return low;
}
```

Advantages of Divide and conquer:-

~~* Tends~~

* It solves difficult problems by dividing the problem into sub problems. It provides a simple solution.

* It allows for execution in multi-processor machines, especially shared memory systems.

* All the subproblems can be solved within the cache, without accessing the main memory.

Disadvantages :-

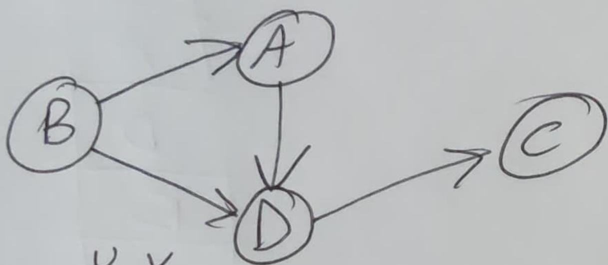
* Recursion is slow because of the overhead of the repeated subproblem calls.

TOPOLOGICAL SORTING

* It is linear ordering of graph vertices such that for every directed edge uv from vertex u to vertex v, u comes ~~f~~ before v in the ordering.

* Applicable on <u>Direct Acyclic Graph</u>
* Linear running time complexity.

* In simple words, in any 2 letters among many, say AB, [A → B] only, no [B → A]



u v

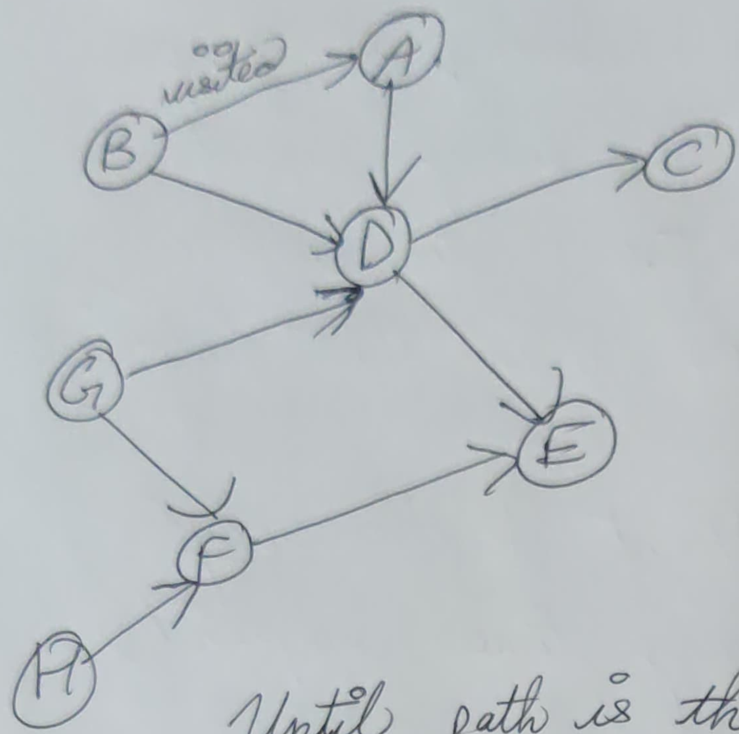
BAD C ✓

BDCA X

A B CD X

Using DFS (B)

TOPOLOGICAL = DFS



Until path is there, no backtrack

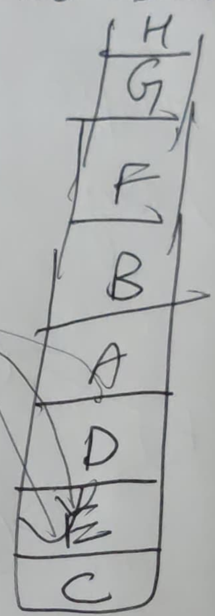
$B \to A \to D \to \cancel{C}$

When dead end, put in stack

$\cancel{B} \to \cancel{A} \to \cancel{D} \to \cancel{E}$

$G \to \cancel{F}$

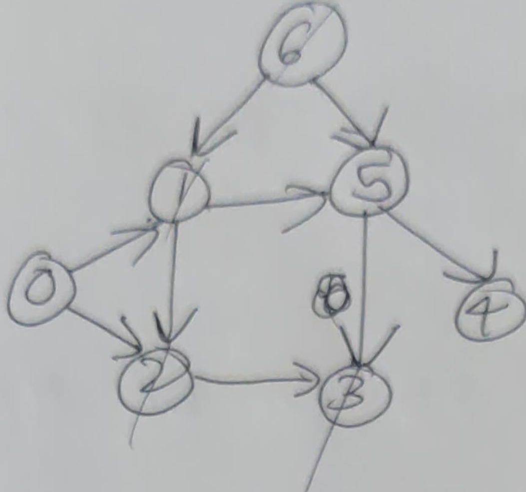
H



HGFBADEC

$T = O(V+E)$

space $= O(V)$



0
5
4
6
1
2
3

$6 \to 1 \to 2 \to \cancel{3}$

$5 \to 4$

Quick Sort

35 50 15 25 80 20 90 45

V=35

35 20 15 25 80 50 90 45 +∞

25 20 15 35 80 50 90 45

25 20 15 | 80 50 90 45

15 20 25 35 45 50 90