Divide & Conquer (D&C) Technique

Introduction

- Many useful algorithms are recursive in structure: to solve a given problem, they call themselves recursively one or more times.
- These algorithms typically follow a divide-and-conquer approach:
- The divide-and-conquer approach involves three steps at each level of the recursion:
  1. **Divide:** Break the problem into several sub problems that are similar to the original problem but smaller in size.
  2. **Conquer:** Solve the sub problems recursively. If the sub problem sizes are small enough, just solve the sub problems in a straightforward manner.
  3. **Combine:** Combine these solutions to create a solution to the original problem.

Comparison based sorting –

- **Bubble sort**
- **Insertion sort**
- **Selection sort**
- **Merge sort**
- **Heap sort**
- **Quick sort**

Non-comparison based sorting –

- **Radix sort**
- **Count sort**
- **Bucket sort**

Radix Sort

- Radix Sort puts the elements in order by comparing the digits of the numbers.
- Each element in the $\boldsymbol{n}$-element array $\boldsymbol{A}$ has $\boldsymbol{d}$ digits, where digit 1 is the lowest-order digit and digit d is the highest order digit.

```
Algorithm: RADIX-SORT(A, d)
 for i ← 1 to d
      do use a stable sort to sort array A on digit i
```

- Sort following elements in Ascending order using radix sort.

    363, 729, 329, 873, 691, 521, 435, 297

Radix Sort - Example

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

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

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

Sort on column 1

Sort on co

2

The entire array is sorted now.

3

Example

- 321,420,5,26,90,2,223

Counting Sort

Counting Sort – Example

- Sort the following elements in Ascending order using counting sort.

3	6	4	1	3	4	1	4	2

Step 1 Given elements are stored in an input array $A[1, \ldots, 9]$

Index	1	2	3	4	5	6	7	8	9
Elements	3	6	4	1	3	4	1	4	2

Step 2 Define a temporary array C. The size of an array C is equal to the **maximum element** in array A. Initialize $C[1, \ldots, 6]$ to 0.

Index	1	2	3	4	5	6
Elements	0	0	0	0	0	0

Counting Sort – Example

- Sort the following elements in Ascending order using counting sort.

Input array A

3	6	4	1	3	4	1	4	2

Step 3 Update an array C with the occurrences of each value of array A

Index	1	2	3	4	5	6
Elements	**2**	**1**	**2**	**3**	**0**	**1**

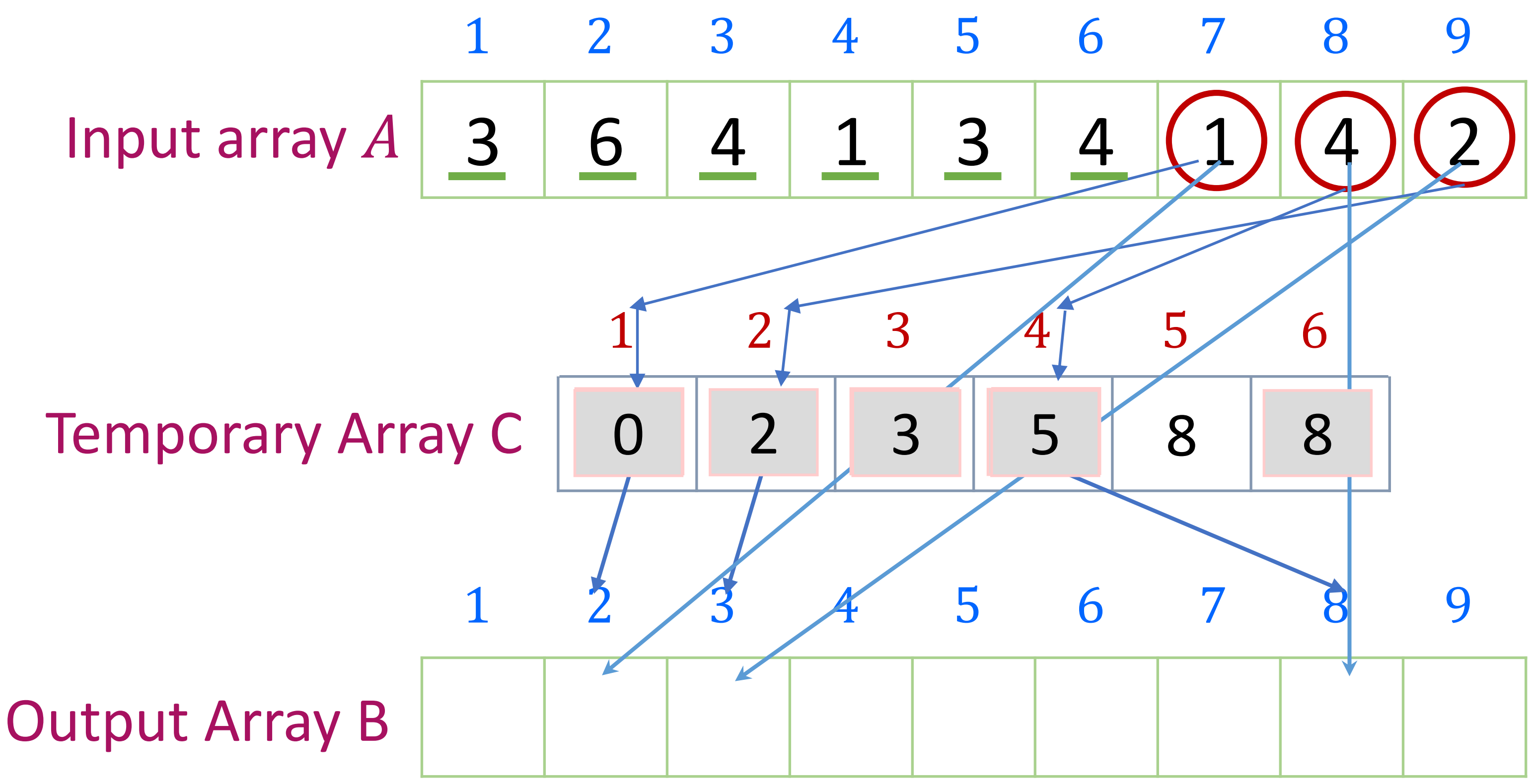
+ +

Step 4 In array C, from index 2 to n add the value with previous element

Index	1	2	3	4	5	6
Elements						

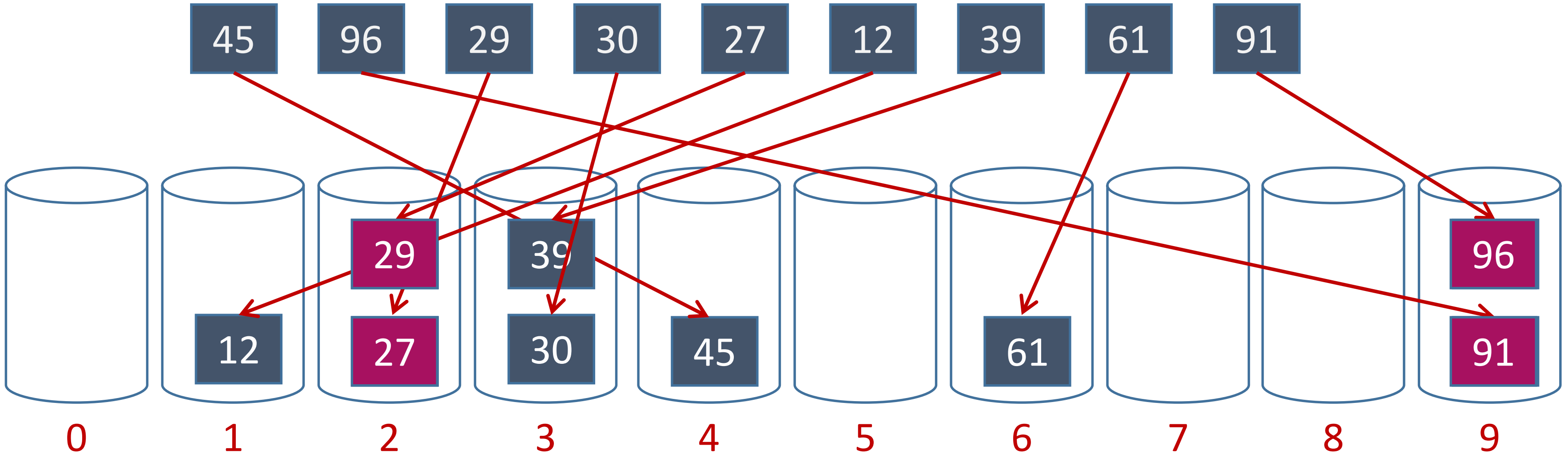
Counting Sort – Example

- Create an output array $B[1\ldots9]$. Start positioning elements of Array A to B as shown below.



Bucket Sort – Example

45	96	29	30	27	12	39	61	91



Sort each bucket queue with insertion sort

Merge all bucket queues together in order

Polynomial Multiplication

Polynomial Multiplication

$X1=5+10x^2+6x^3$

$X2=1+2x^1+4x^2$

$X3=?$

What will be the degree of X3?

Degree of X1+Degree of X2

3+2=5

```
for(i=0;i<=m;i++)
for(j=0;j<=n;j++)
{
k=i+j;
X3[k]=X3[k]+X1[i]*X2[j];
}
```

Example

Given two polynomials represented by two arrays, multiplies given two polynomials.

A[]={0,3,5,0,7}

B[]={0,3,0,5}

Time Complexity

- Time complexity of the above solution is O(mn).
- If size of two polynomials same, then time complexity is $O(n^2)$.

Strassen’s Algorithm for Matrix Multiplication

Matrix Multiplication

- Multiply following two matrices. Count how many scalar multiplications are required.

$$\begin{bmatrix} 1 & 3 \\ 7 & 5 \end{bmatrix} \cdot \begin{bmatrix} 6 & 8 \\ 4 & 2 \end{bmatrix}$$

$$answer = \begin{bmatrix} 1 \times 6 + 3 \times 4 & 1 \times 8 + 3 \times 2 \\ 7 \times 6 + 5 \times 4 & 7 \times 8 + 5 \times 2 \end{bmatrix}$$

- To multiply 2×2 matrices, total 8 (2^3) scalar multiplications are required.

```
void multiply(int A[][N], int B[][N], int C[][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            C[i][j] = 0;
            for (int k = 0; k < N; k++)
            {
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
}
```

- Time Complexity of above method is $O(N^3)$

Matrix Multiplication

- In general, A and B are two 2×2 matrices to be multiplied.

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

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

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{aligned}$$

- Computing each entry in the product takes $\boldsymbol{n}$ **multiplications** and there are $\boldsymbol{n^2}$ **entries** for a total of $\boldsymbol{O(n^3)}$**.**

Strassen's Algorithm for Matrix Multiplication

- Consider the problem of **multiplying** two $n \times n$ matrices.
- Strassen's Matrix multiplication can be performed only on **square matrices** where **n** is a **power of 2**. Order of both of the matrices are **n × n**.
- The main idea is **to save one multiplication** on a small problem and then use recursion.

Strassen's Algorithm for Matrix Multiplication

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

Step 1

$S_1 = B_{12} - B_{22}$

$S_2 = A_{11} + A_{12}$

$S_3 = A_{21} + A_{22}$

$S_4 = B_{21} - B_{11}$

$S_5 = A_{11} + A_{22}$

$S_6 = B_{11} + B_{22}$

$S_7 = A_{12} - A_{22}$

$S_8 = B_{21} + B_{22}$

$S_9 = A_{11} - A_{21}$

$S_{10} = B_{11} + B_{12}$

Step 2

$P_1 = A_{11} \odot S_1$

$P_2 = S_2 \odot B_{22}$

$P_3 = S_3 \odot B_{11}$

$P_4 = A_{22} \odot S_4$

$P_5 = S_5 \odot S_6$

$P_6 = S_7 \odot S_8$

$P_7 = S_9 \odot S_{10}$

All above operations involve only **one multiplication**.

Step 3

Final Answer:

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

Where,

$C_{11} = P_5 + P_4 - P_2 + P_6$

$C_{12} = P_1 + P_2$

$C_{21} = P_3 + P_4$

$C_{22} = P_5 + P_1 - P_3 - P_7$

No multiplication is required here.

Strassen's Algorithm - Analysis

- It is therefore possible to multiply two 2×2 matrices using only **seven scalar multiplications**.
- Let $t(n)$ be the time needed to multiply two $n \times n$ matrices by **recursive use of equations**.

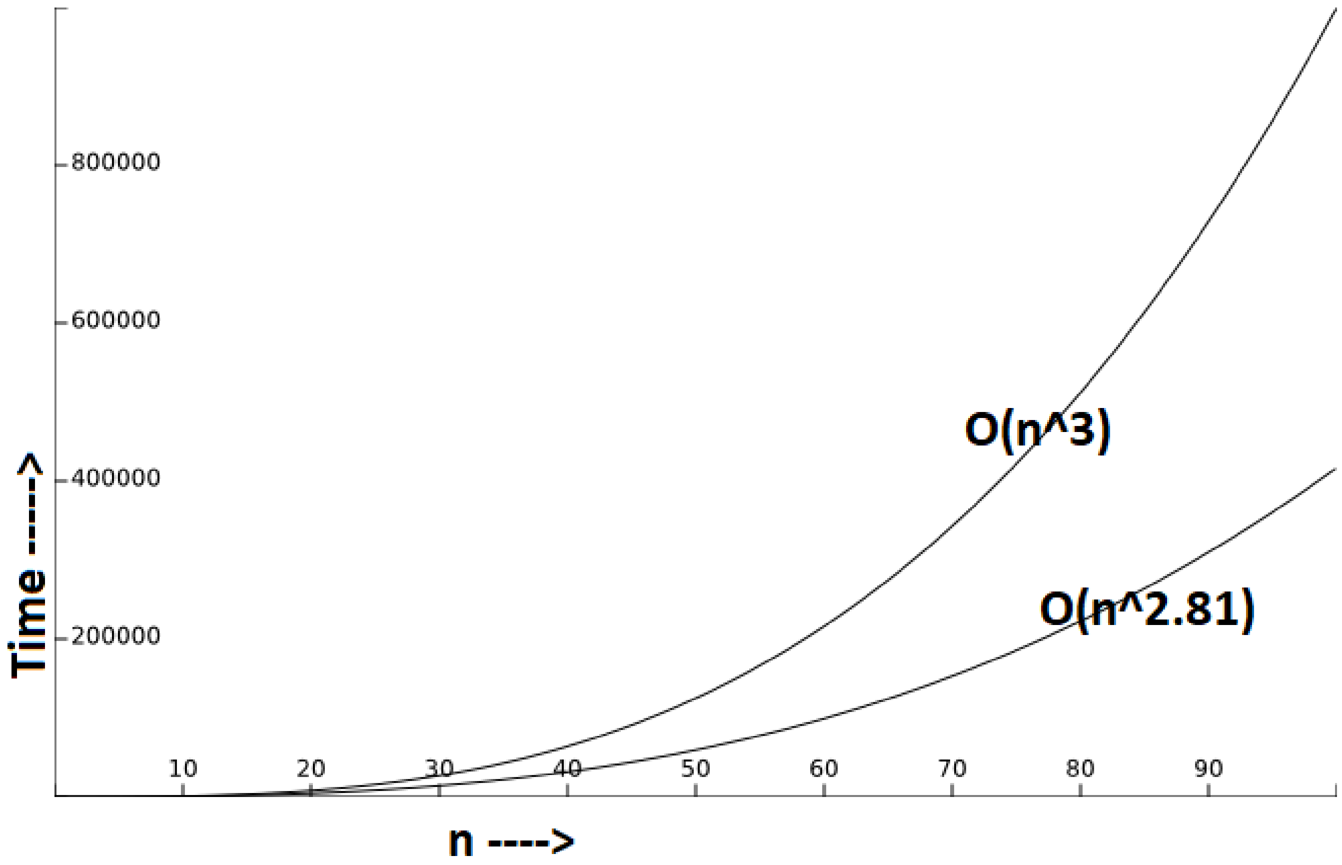
$$\boldsymbol{t(n) = 7t(n/2) + g(n)}$$

$$\boldsymbol{t(n) = lt(n/b) + g(n)}$$

Where $g(n) \in O(n^2)$.

- The general equation applies with $l = 7, b = 2$ and $k = 2$.
- Since $l > b^k$, the **third case** applies and $t(n) \in O\left(n^{lg7}\right)$.
- Since $lg7 \approx 2.81$, it is possible to multiply two $n \times n$ matrices in a time $\boxed{\boldsymbol{O(n^{2.81})}}$.

However, O(n2.81)O($n^{2.81}$) is not much improvement though but enough for n having large value as depicted in the graph below,



Min – Max Problem

Min Max Problem

Problem Statement

- The Max-Min Problem in algorithm analysis is finding the maximum and minimum value in an array.

Solution

- To find the maximum and minimum numbers in a given array ***numbers[]*** of size **n**, the following algorithm can be used.

1) **The naive method** and
2) **Divide and conquer approach**

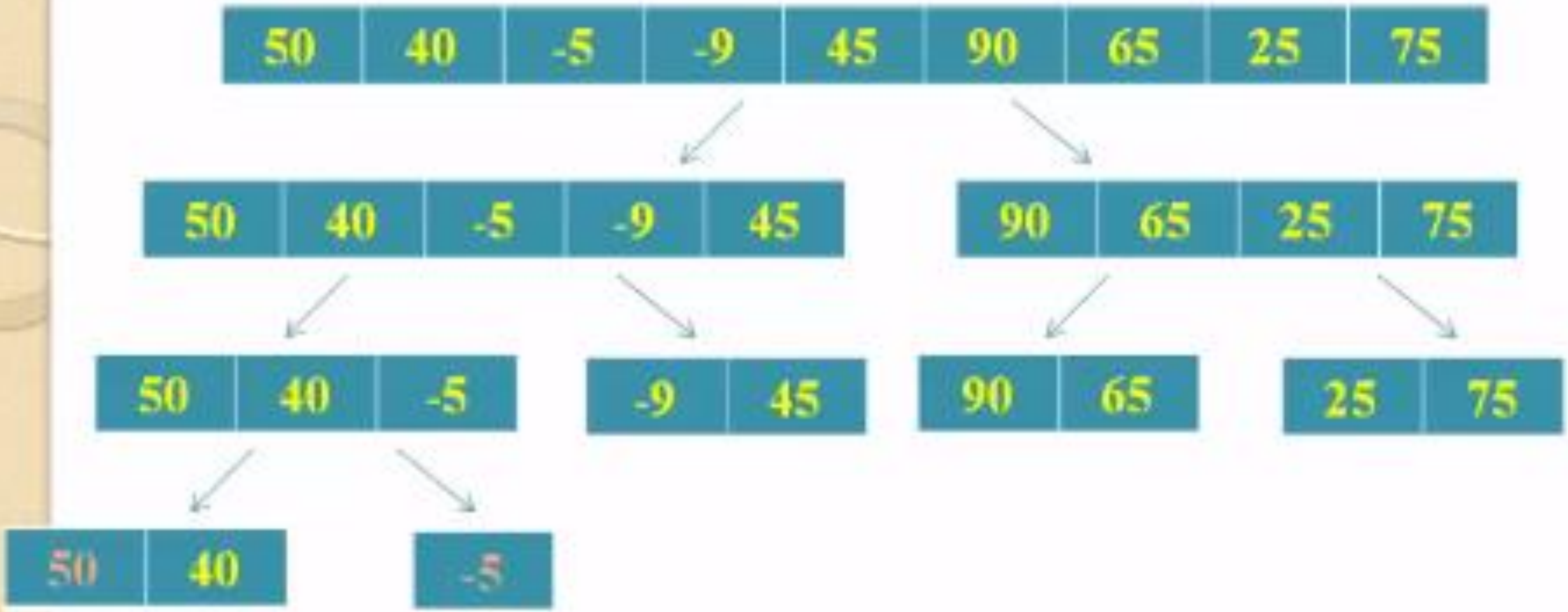
Naïve Method

- Naïve method is a basic method to solve any problem. In this method, the maximum and minimum number can be found separately. To find the maximum and minimum numbers, the following straightforward algorithm can be used.

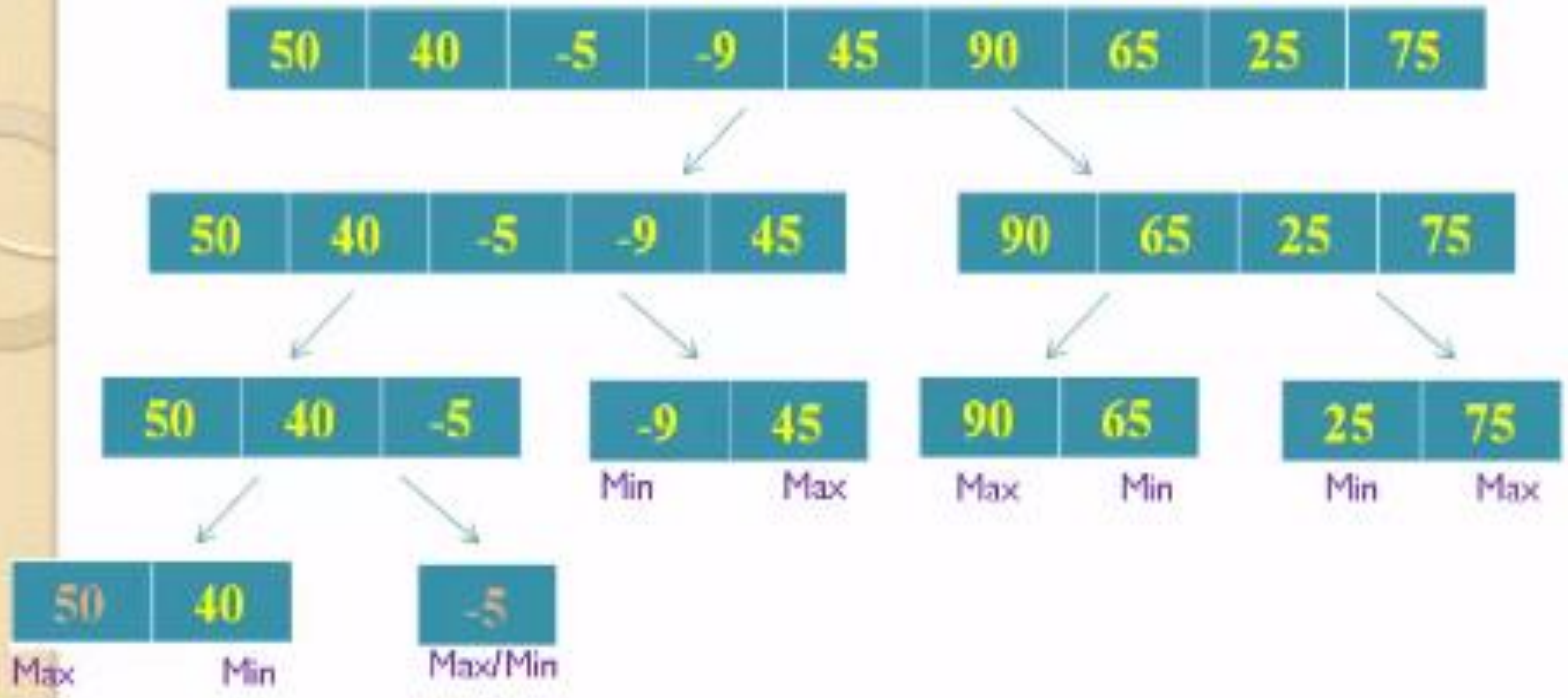
```
Algorithm: Max-Min-Element (numbers[])
max := numbers[1]
min := numbers[1]
for i = 2 to n do
        if numbers[i] > max then
                max := numbers[i]
        if numbers[i] < min then
                min := numbers[i]
return (max, min)
```

- The number of comparison in Naive method is **2n - 2**.
- The number of comparisons can be reduced using the divide and conquer approach.

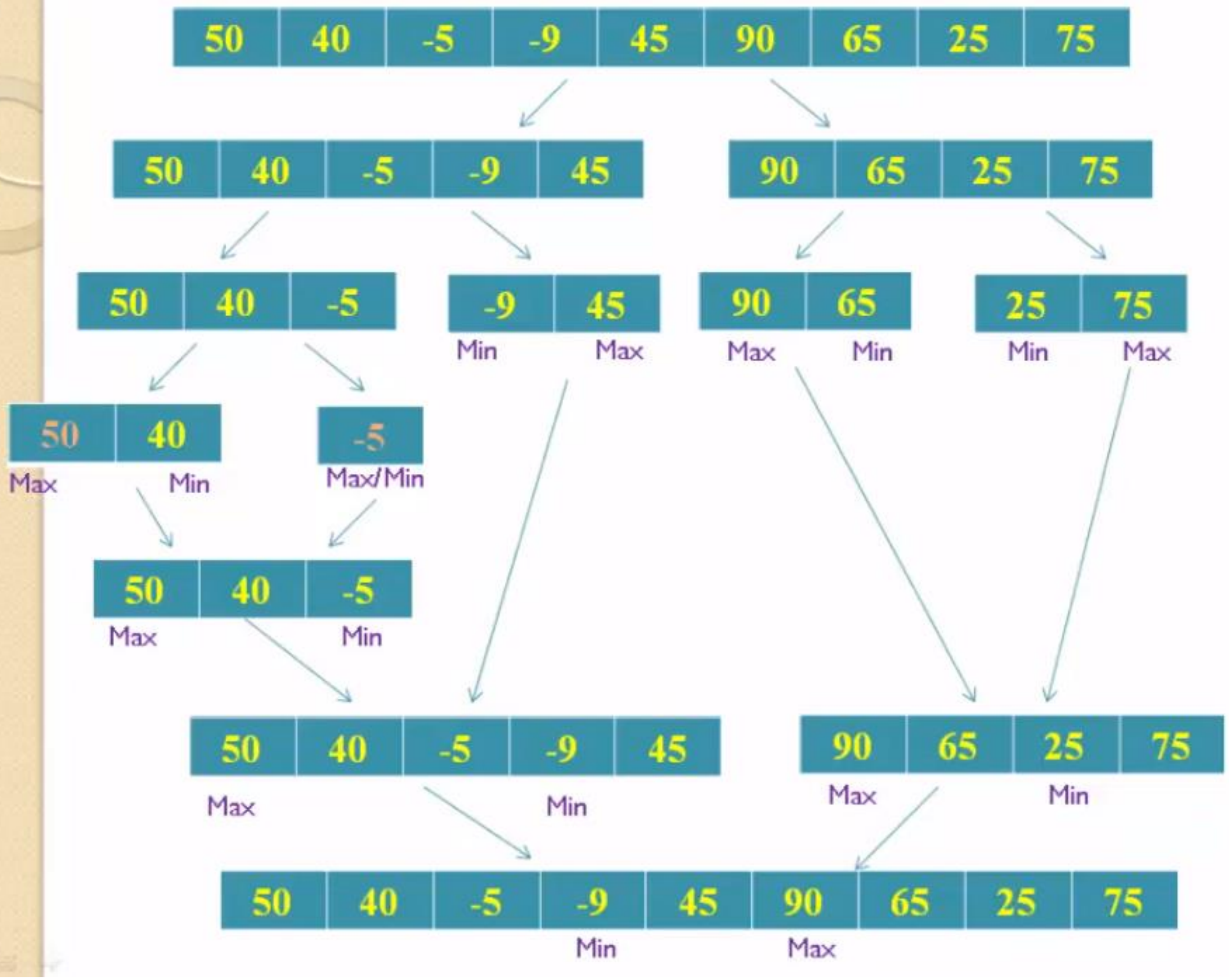
50	40	-5	-9	45	90	65	25	75


50 40 -5 -9 45 90 65 25 75
50 40 -5 -9 45
90 65 25 75
50 40 -5
-9 45
90 65
25 75
50 40
-5

Example


50 40 -5 -9 45 90 65 25 75
50 40 -5 -9 45
90 65 25 75
50 40 -5
-9 45
Min Max
90 65
Max Min
25 75
Min Max
50 40
Max Min
-5
Max/Min

Example


50 40 -5 -9 45 90 65 25 75
50 40 -5 -9 45
90 65 25 75
50 40 -5
-9 45
Min Max
90 65
Max Min
25 75
Min Max
50 40
Max Min
-5
Max/Min
50 40 -5
Max Min
50 40 -5 -9 45
Max Min
90 65 25 75
Max Min
50 40 -5 -9 45 90 65 25 75
Min Max

Divide and conquer approach

a. Let P = (n, a [i],……,a [j]) denote an arbitrary instance of the problem.

b. Here ‘n’ is the no. of elements in the list (a [i],….,a[j]) and we are interested in finding the maximum and minimum of the list.

c. If the list has more than 2 elements, P has to be divided into smaller instances.

d. For example, we might divide ‘P’ into the 2 instances, P1=([n/2],a[1],……..a[n/2]) & P2= (n-[n/2], a[[n/2]+1],….., a[n]) After having divided ‘P’ into 2 smaller sub problems, we can solve them by recursively invoking the same divide-and-conquer algorithm.

```
Algorithm Max_Min(i
    ,j,max,min)
{        if ( i == j )
         {        max ← A[i]
                  min ← A[j]
         }
         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
    Max_Min(i , mid, max , min)
    Max_Min(mid+1 , j, max_new , min_new)

    if (max < max_new ) then
         max ← max_new

    if (min > min_new ) then
         min ← min_new
}
```

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

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

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

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

$$T(n) = 2^i T\left(\frac{n}{2^i}\right) + 2^i + 2^{i-1} + \ldots\ldots\ldots\ldots\ldots\ldots + 2$$

$$\frac{n}{2^i} = 2 \Rightarrow n = 2^{i+1}$$

$$T(n) = 2^i\, T(2) + 2^i + 2^{i-1} + \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots + 2$$

$$= 2^i .1 + 2^i + 2^{i-1} + \ldots\ldots\ldots\ldots\ldots\ldots\ldots + 2$$

$$= 2^i + \frac{2(2^i - 1)}{2-1}$$

$$= 2^{i+1} + 2^i - 2$$

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

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