Assignment 2

→ **Problem Statement :**

Write a program to find Maximum and Minimum elements in an array using Divide & Conquer strategy and verify the time complexity.

→ **Theory :**

A] **Divide and Conquer Strategy :**

Divide and Conquer is an algorithm used to divide a large set of problem statements into smaller sub-problems and then combining the solution of these sub-problems.

1. Divide Instance of problem into two or more smaller instances.
2. Solve smaller instances recursively
3. Obtain Solution to original instance by combining these solutions.

B] **Finding Min-Max from an Array :**

1. **Conventional Way :**

- Initialize values of min & max of the 1st 2 elements respectively. Starting from 3rd location, compare each element with max & min, and change max & min accordingly

```
a. Algorithm MaxMin (a, n, max, min) {
b.     max := min := a[1]
c.     for i := 2 to n do {
d.         if (a[i] > max) then max := a[i]
e.         if (a[i] < min) then min := a[i]
f.     }
g. }
```

- This conventional approach requires $2(n-1)$ number of comparisons, which can be reduced to $n-1$ in best case by converting if into an if-else statement.
- If the elements of the arrays are polynomials, strings, vectors, etc; then the cost of element comparison is much higher.

---

2. <u>Divide & Conquer Method</u> :

```
a. Algorithm MaxMin (i, j, max, min) {
b.     if (i = j) then max := min := a[i]
c.     else if (i = j - 1) then {
d.         if (a[i] < a[j]) max := a[j], min := a[i]
e.         else max := a[i], min := a[j]
f.     }
g.     else {
h.     mid := (i + j) / 2
i.     MaxMin (i, mid, max, min)
j.     MaxMin (mid + 1, j, max1, min1)
```

k.   if (max < max1) then max := max1
l.   if (min > min1) then min := min1
m. }   }
n. }

Evaluation

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

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

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

$$T(n) = 4T\left(\frac{n}{4}\right) + 4 + 2 \quad — \quad (1)$$

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

$$= 2T\left(\frac{n}{8}\right) + 2$$

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

$$\therefore \boxed{T(k-1) = 2^{k-1} \cdot T\left(\frac{n}{2^{k-1}}\right) + \sum_{i=1}^{k-1} 2^i}$$

Put $n = 2^k$, therefore

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

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

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

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

Total number of comparisons $= \frac{3n}{2} - 2$

Hence, Divide & Conquer Strategy is 25% faster than the conventional approach.

Program:

```
#include <iostream>
using namespace std;

typedef struct {
  int min;
  int max;
} Pair;

//find maximum between 2 numbers
int max(int x, int y) {
  if(x > y) {
    return x;
  }
  return y;
}

//find minimumum between 2 numbers
int min(int x, int y) {
  if(x > y) {
    return y;
  }
  return x;
}

Pair minmax(int arr[], int left, int right) {

  Pair minMax, leftArr, rightArr;

  //one element
  if(left == right) {
    minMax.min = arr[left];
    minMax.max = arr[left];

    for(int i = left; i <= right; i++) {
      cout<< arr[i]<< " ";
    }
    cout<< endl<< minMax.min<< ", "<< minMax.max<< endl<<
endl;

    return minMax;
  }

  //2 elements
  if(left + 1 == right) {
    minMax.min = min(arr[left], arr[right]);
    minMax.max = max(arr[left], arr[right]);
```

```
    for(int i = left; i <= right; i++) {
      cout<< arr[i]<< " ";
    }
    cout<< endl<< minMax.min<< ", "<< minMax.max<< endl<<
endl;

    return minMax;
  }

  //more than 2 elements
  int mid = (left + right)/2;

  //recursive calls to find min and max
  leftArr = minmax(arr, left, mid);
  rightArr = minmax(arr, mid + 1, right);

  //find the min and max from the divided arrays
  minMax.min = min(leftArr.min, rightArr.min);
  minMax.max = max(leftArr.max, rightArr.max);

  for(int i = left; i <= right; i++) {
    cout<< arr[i]<< " ";
  }
  cout<< endl<< minMax.min<< ", "<< minMax.max<< endl<< endl;

  return minMax;
}

int main() {

  int arr[10];
  cout<< endl<< "Enter 10 numbers: ";
  //accepting 10 elements
  for(int i = 0; i < 10; i++) {
    cin>>arr[i];
  }
  cout<< endl<< "************************"<< endl<< "Working:
"<< endl<< endl;
  Pair minMax = minmax(arr, 0, 9);
  cout<< "************************"<< endl<< "Final Output:
"<< minMax.min<< ", "<< minMax.max<< endl<<
"************************"<< endl<< endl;
  return 0;
}
```

Output:

```
Someshwars-MacBook-Pro:GroupB someshwargaikwad$ g++
Assignment2.cpp
Someshwars-MacBook-Pro:GroupB someshwargaikwad$ ./a.out

Enter 10 numbers: 32 15 46 78 12 90 21 98 23 55

***********************
Working:

32 15
15, 32

46
46, 46

32 15 46
15, 46

78 12
12, 78

32 15 46 78 12
12, 78

90 21
21, 90

98
98, 98

90 21 98
21, 98

23 55
23, 55

90 21 98 23 55
21, 98

32 15 46 78 12 90 21 98 23 55
12, 98

***********************
Final Output: 12, 98
***********************

Someshwars-MacBook-Pro:GroupB someshwargaikwad$
```

→ <u>Conclusion</u> :

Topics Covered :

1. Divide & Conquer
2. Divide & Conquer strategy to find minimum & maximum numbers in an array.