

**Heaven's Light is Our Guide**



# **Rajshahi University of Engineering and Technology**

## **Department of Computer Science and Engineering**

**Course No:** CSE.2202

**Course Title:** Sessional based on CSE.2201 (Computer Algorithms)

**Lab Report No:** 02

**Lab Report On:** Divide and Conquer Algorithm (Max\_Min)

**Submitted By**

Md. Ariful Islam  
Roll No: 1803046  
Section: A  
Department: CSE


**Submitted To**

Dr. Md. Ali Hossain  
Associate Professor  
Dept. of CSE, RUET

**Date:** 06-02-2021

❖ **Problem Statement:** The problem is to find the maximum and the minimum value of a given data sample in .txt file using **Straight Forward** strategy and **Divide & Conquer** strategy. The txt files contain 10k, 20k, 30k, 40k and 50k numbers. Then to find the number of steps required in both strategies and compare the number of steps (Graphs should be included if required).


❖ **Details Description and Algorithm:** The straight forward strategy to find maximum and minimum value is very simple. At first we have to assign the 1<sup>st</sup> value to the max and min. Then we have to iterate through the 2<sup>nd</sup> value to the last value and check if the value is greater than max or less than min. If greater than max then assign that to max or if less than min then assign that to min. The algorithm could be like below:

 **Straight( arr,n )** [ **Straight** is a function that find the **max** and **min** from **n** values in **arr** ]

1. Set **Max:= arr[1]** and **Min:=arr[1]**.
2. Repeat **i = 1** to **n** by **1**
3.     **IF Max <= arr[i]** then set **Max:= arr[i]**.
4.     **ELSE IF Min >= arr[i]** then set **Min:= arr[i]**.
- [ End of Repeat **step 2** loop ]
5. Exit.

The Divide & Conquer strategy splits the input into two sub problems of the same kind as the original problem. Then again splits these two problems if necessary. Then from the last two solved problems it continues to join and solve. Finally when we

return in the first call of recursion, we get the whole problem solved. The algorithm could be like following:

 **MaxMin( i, j, max, min )** [ **MaxMin** is a function that find the **max** and **min** between i to j in arr ]

1. IF  $i = j$  then set  $\text{max} := \text{min} := \text{arr}[i]$ .
2. ELSE IF  $i = j - 1$  then
3.     IF  $\text{arr}[i] < \text{arr}[j]$  then set  $\text{max} := \text{arr}[j]$  and  $\text{min} := \text{arr}[i]$ .
4.     ELSE set  $\text{max} := \text{arr}[i]$  and  $\text{min} := \text{arr}[j]$ .
- [ End of else if structure ]
5. ELSE set  $\text{mid} := (i + j) / 2$ .
6.     **MaxMin( i, mid, max, min )**
7.     **MaxMin( mid+1, j, max1, min1 )**
8.     IF  $\text{max} < \text{max1}$  then set  $\text{max} := \text{max1}$ .
9.     IF  $\text{min} > \text{min1}$  then set  $\text{min} := \text{min1}$ .
- [ End of else structure ]
10. Exit.

### ❖ Implemented Code:

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;

vector<ll>arr1;
ll C1,C2,Max,Min,Mx,Mn,mid;

void D_C(int i,int j){
    if(i==j){
        C1+=3;
```

```
        Mx=arr1[i];
        Mn=arr1[i];
    }
    else if(i==(j-1)){
        if(arr1[i]>arr1[j]){
            Mx=arr1[i];
            Mn=arr1[j];
            C1+=4;
        }
        else{
            Mx=arr1[j];
            Mn=arr1[i];
            C1+=4;
        }
    }
    else{
        mid=(i+j)/2;
        C1+=3;
        D_C(i,mid);
        D_C(mid+1,j);
        C1+=2;
    }
    if(Max<Mx){
        Max=Mx;
        C1+=2;
    }
    if(Min>Mn){
        Min=Mn;
        C1+=2;
    }
}

void S_F(){
    Max=arr1[0],Min=arr1[0];
    C2+=2;
    for(int i=1;i<arr1.size();i++){
        C2+=2;
        if(arr1[i]>Max){
            Max=arr1[i];
        }
    }
}
```

```

        C2+=1;
    }
    else if(arr1[i]<Min){
        Min=arr1[i];
        C2+=1;
    }
    C2+=2;
}
C2+=2;
}

void solve(string a){
    string b;
    ifstream f1;

    f1.open(a);
    arr1.clear();

    while(!f1.eof()){
        f1>>b;
        arr1.push_back(stod(b));
    }

    C1=0,C2=0;

    Max=-1;Min=1e9;
    D_C(0,arr1.size()-1);

    cout<<" 1.Divide & Conquer:- Max: "<<Max<<" Min: "<<Min<<" Count:
"<<C1<<endl;

    S_F();
    cout<<" 2.Straight Forward:- Max: "<<Max<<" Min: "<<Min<<" Count:
"<<C2<<endl;

}

void menu(){
    cout<<"\n\tData Set\n -----"<<endl;

```

```
cout<<" 1. 10k\n 2. 20k\n 3. 30k\n 4. 40k\n 5. 50k\n 0. Exit\n"<<endl;
cout<<" Enter your choice: ";
}

int main(){
    int x;

    while(1){
        menu();
        cin>>x;
        switch(x){
            case 1: solve("10k.txt"); break;
            case 2: solve("20k.txt"); break;
            case 3: solve("30k.txt"); break;
            case 4: solve("40k.txt"); break;
            case 5: solve("50k.txt"); break;
            case 0: cout<<"\n Exiting..."<<endl; break;
            default: cout<<"\n Invalid Input...\n"<<endl; break;
        }
        if(x==0){
            break;
        }
    }

    return 0;
}
```

### ❖ Sample Input & Output:

- **Input:** The code takes user-input to select data set from five text files named **10k.txt**, **20k.txt**, **30k.txt**, **40k.txt** and **50k.txt** .

➤ **Output:**

```
"F:\4th Semester\CSE\CSE.2202\Lab 2\lab_2.exe"

Data Set
-----
1. 10k
2. 20k
3. 30k
4. 40k
5. 50k
0. Exit

Enter your choice: 5
1.Divide & Conquer:- Max: 51983 Min: 0 Count: 434743
2.Straight Forward:- Max: 51983 Min: 0 Count: 200020

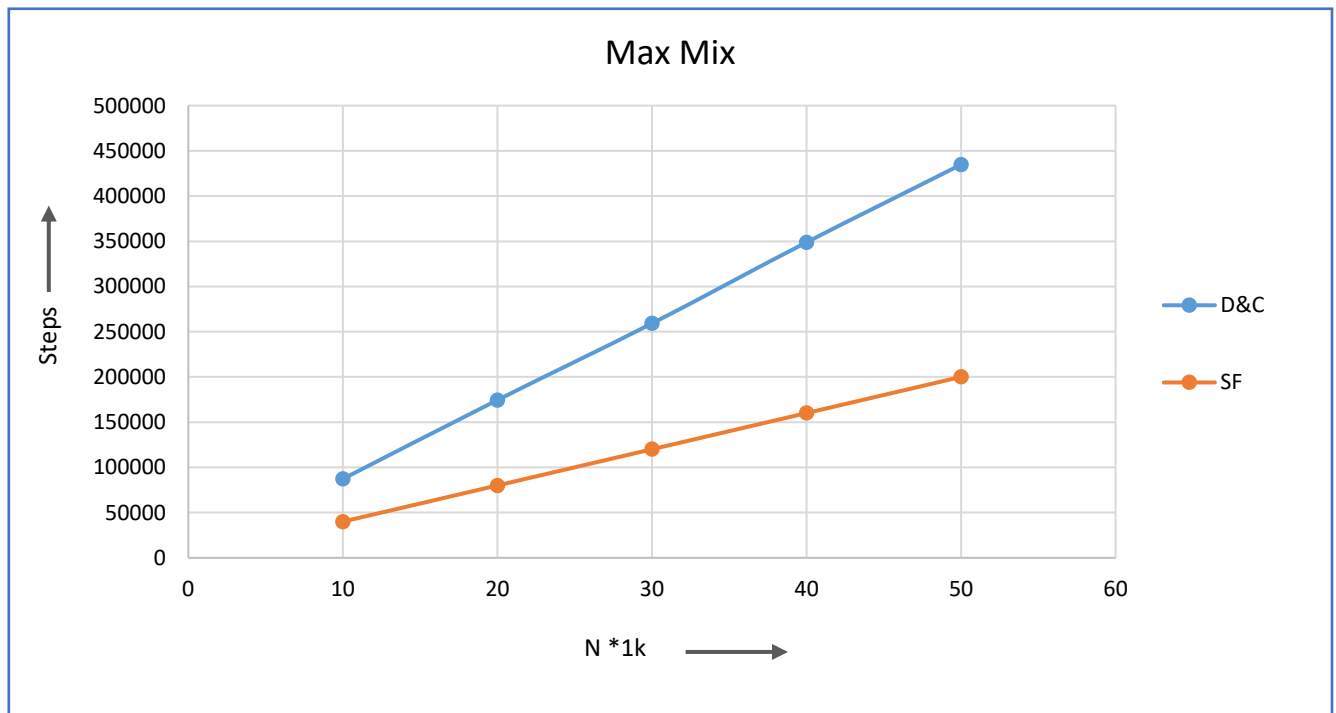
Data Set
-----
1. 10k
2. 20k
3. 30k
4. 40k
5. 50k
0. Exit

Enter your choice: 0

Exiting...

Process returned 0 (0x0)   execution time : 4.927 s
Press any key to continue.
```

➤ **Graph:**



❖ **Discussion & Conclusion:** Theoretically we know that the Divide & Conquer strategy is more efficient than the Straight Forward Strategy to find the maximum and the minimum value from a data set.

But from the experimental value we get higher steps in the Divide and Conquer strategy than the Straight Forward strategy, though it shouldn't happen. This might be happened for the recursive calling of MaxMin() function in the Divide & Conquer strategy.

\*\*\* END \*\*\*