"Heaven's Light is Our Guide"

Department of Computer Science & Engineering

RAJSHAHI UNIVERSITY OF ENGINEERING & TECHNOLOGY

Lab Report

Course No.: CSE 2202

Topic: Sessional Based on CSE 2201

Submitted to:

Biprodip Pal
Assistant Professor
Department of Computer Science & Engineering

Submitted by:

Md. Al Siam
Department of Computer Science & Engineering
Session: 2016-2017
Section: A
Roll No.: 1603008

Problem

Observation of the number of comparisons for different types of approaches of finding the maximum and minimum number from a list.

Machine Configuration

Processor : Intel® Core™ i5-7200U CPU @ 2.50 GHz 2.71 GHz
Installed memory (RAM) : 8.00 GB(7.89 GB usable)

System type : 64-bit Operating System, x64-based processor

Implementation of different types of approaches in C++

Brute Force Approach

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

int main(){
    int n;
    cout << "How many numbers do you want to take?: ";
    cin >> n;

    freopen("G:\Random Numbers.txt", "r", stdin);
    freopen("G:\Output for naive approach.txt", "w", stdout);

    int num[n+2];

    for(int i = 0; i < n; i++)
        scanf("%d", num+i);

    int maxi = num[0];
    int mini = num[0];
    int kount = 0;
```

```
    for(int i = 1; i < n; i++){
        if(maxi > num[i]) maxi = num[i];
        if(mini < num[i]) mini = num[i];
        kount += 2;
    }

    cout << "Maximum element: " << maxi << endl;
    cout << "Minimum element: " << mini << endl;

    cout << "Number of comparison for " << n
        << " data is: " << kount << endl;

}
```

Using Recursive Function Handling One Base Leaf

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

int kount = 0;

typedef struct{
    int maxi;
    int mini;
} dataset;

dataset findmaxmin(int* num, int lo, int hi){
    dataset ret;
    if(lo == hi){
        ret.maxi = ret.mini = num[lo];
        return ret;
    }

    int mid = lo + (hi - lo) / 2;
```

```
    dataset ret1;
    dataset ret2;

    ret1 = findmaxmin(num, lo, mid);
    ret2 = findmaxmin(num, mid+1, hi);

    ret.maxi = max(ret1.maxi, ret2.maxi);
    ret.mini = min(ret1.mini, ret2.mini);

    kount += 2;

    return ret;
}

int main(){
    int n;
    cout << "How many numbers do you want to take?: ";
    cin >> n;

    freopen("G:\Random Numbers.txt", "r", stdin);
    freopen("G:\Output for one leaf approach.txt", "w", stdout);

    int num[n+2];

    for(int i = 0; i < n; i++)
        scanf("%d", num+i);

    dataset ans = findmaxmin(num, 0, n-1);

    cout << "Maximum element: " << ans.maxi << endl;
    cout << "Minimum element: " << ans.mini << endl;

    cout << "Number of comparison for " << n
        << " data is: " << kount << endl;
}
```

Using Recursive Function Handling Two Base Leaf

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

int kount = 0;

typedef struct{
    int maxi;
    int mini;
} dataset;

dataset findmaxmin(int* num, int lo, int hi){
    dataset ret;

     //Base 1
    if(lo == hi){
        ret.maxi = ret.mini = num[lo];
        return ret;
    }

     //Base 2
    if(lo+1 == hi){
        if(num[lo] > num[hi]){ret.maxi = num[lo]; ret.mini =
num[hi];}
        else {ret.maxi = num[hi]; ret.mini = num[lo];}
        kount += 1;
        return ret;
    }

    int mid = lo + (hi - lo) / 2;

    dataset ret1;
    dataset ret2;

    ret1 = findmaxmin(num, lo, mid);
    ret2 = findmaxmin(num, mid+1, hi);
```

```
    ret.maxi = max(ret1.maxi, ret2.maxi);
    ret.mini = min(ret1.mini, ret2.mini);

    kount += 2;

    return ret;
}

int main(){
    int n;
    cout << "How many numbers do you want to take?: ";
    cin >> n;

    freopen("G:\Random Numbers.txt", "r", stdin);
    freopen("G:\Output for two leaf approach.txt", "w", stdout);

    int num[n+2];

    for(int i = 0; i < n; i++)
        scanf("%d", num+i);

    dataset ans = findmaxmin(num, 0, n-1);

    cout << "Maximum element: " << ans.maxi << endl;
    cout << "Minimum element: " << ans.mini << endl;

    cout << "Number of comparison for " << n
        << " data is: " << kount << endl;
}
```

Complexity Observation

The naïve approach, the only loop runs for n-1 times. For the divide and conquer approach, we have to visit all the n data in base leaf, or some less for two leaf approach, but not so significantly to reduce complexity. Hence, the complexity for all the approaches is $\mathbf{O(n^3)}$

Experimental Result (Comparisons)

N	Brute Force	Divide & Conquer (One leaf)	Divide & Conquer (Two leaf)
10000	19998	19998	15902
50000	99998	99998	82766
100000	199998	199998	165534
150000	299998	299998	234462
250000	499998	499998	381070
300000	599998	599998	468926
350000	699998	699998	568926
400000	799998	799998	662142

Comment

The Brute Force Approach and the one leaf based divide and conquer approach gives the same efficiency as all the data are compared at least one time in both approach. But in two leaf based divide and conquer approach, we can sometimes lessen the depth of recursion tree, e.g, when there are only two data in the segment. Then comparison becomes less, but not that less to reduce theoretical complexity.