RAJSHAHI UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Lab report: 02
Course No.: CSE 2202

Date of Experiment: 05.11.2018
Date of Submission: 13.11.2018

Submitted to:
Biprodip Pal
Assistant Professor,
Department of Computer Science and Engineering
Rajshahi University of Engineering and Technology

Submitted by:
Riyad Morshed Shoeb
Roll No: 1603013
Section: A
Department of Computer Science and Engineering
Rajshahi University of Engineering and Technology

Problem: Efficiency consideration in terms of comparisons for maximum and minimum finding from a set of elements using brute force technique and divide conquer approach.

Using brute force approach

Code:

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <fstream>
#include <vector>
using namespace std;
int main(void)
{
  ifstream read_file("input.txt");
  vector <int> input;
  int iterator;
  int INPUT_SIZE;
  int temp;
  int max;
  int min;
  double step_required;
  cout << "Enter input size: ";
  cin >> INPUT_SIZE;
  for(iterator=0; iterator<INPUT_SIZE; iterator++)
  {
    read_file >> temp;
    input.push_back(temp);
  }
  max = input[0];
  min = input[0];
  step_required = 0;
  for(iterator=1; iterator<INPUT_SIZE; iterator++)
  {
    if(input[iterator] > max)
      max = input[iterator];
    if(input[iterator] < min)
      min = input[iterator];
    step_required++;
  }
  cout << "Max = " << max << endl;
  cout << "Min = " << min << endl;
  cout << "Required steps = " << step_required << endl;
}
```

Theoretical Complexity:

The program loops through the given data only once and each time, it does two comparisons. Hence, total number of comparisons=2n.
∴ Time complexity $= O(n)$

Using divide and conquer approach (Case 1)

Code:

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <fstream>
using namespace std;
int step_required = 0;
struct couple
{
  int max;
  int min;
};
struct couple MAX_MIN(int data[], int left, int right)
{
  struct couple max_min;
  if(left == right)
  {
    max_min.max = max_min.min = data[left];
    return max_min;
  }
  else if((left+1) == right)
  {
    if(data[left] > data[right])
    {
      max_min.max = data[left];
      max_min.min = data[right];
    }
    else
    {
      max_min.max = data[right];
      max_min.max = data[left];
    }
    return max_min;
  }
  else
  {
    struct couple right_subarray;
    struct couple left_subarray;
    int mid;
    mid = (left + right)/2;
    left_subarray = MAX_MIN(data, left, mid);
    right_subarray = MAX_MIN(data, mid+1, right);
    if(left_subarray.max > right_subarray.max)
      max_min.max = left_subarray.max;
    else
      max_min.max = right_subarray.max;
    if(left_subarray.min < right_subarray.min)
      max_min.min = left_subarray.min;
    else
      max_min.min = right_subarray.min;
    step_required++;
    return max_min;
```

```
  }
}
int main(void)
{
  ifstream read_file("input.txt");
  int INPUT_SIZE;
  int iterator;
  struct couple max_min;
  cout << "Enter input size: ";
  cin >> INPUT_SIZE;
  int input[INPUT_SIZE];
  for(iterator=0; iterator<INPUT_SIZE; iterator++)
    read_file >> input[iterator];
  max_min = MAX_MIN(input, 0, INPUT_SIZE-1);
  cout << "Max = " << max_min.max << endl;
  cout << "Min = " << max_min.min << endl;
  cout << "Required steps = " << step_required << endl;
}
```

Theoretical Complexity:

$\mathrm{T(n)} = 2\mathrm{T}\left(n/2\right) + c$
From master theorem, $\mathrm{T(n)} = O(logn)$

Using divide and conquer approach (Case 2)

Code:

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <fstream>
using namespace std;
int step_required = 0;
struct couple
{
  int max;
  int min;
};
struct couple MAX_MIN(int data[], int left, int right)
{
  struct couple max_min;
  if(left == right)
  {
    max_min.max = max_min.min = data[left];
    return max_min;
  }
  else
  {
    struct couple right_subarray;
    struct couple left_subarray;
    int mid;
    mid = (left + right)/2;
    left_subarray = MAX_MIN(data, left, mid);
```

```
    right_subarray = MAX_MIN(data, mid+1, right);
    if(left_subarray.max > right_subarray.max)
      max_min.max = left_subarray.max;
    else
      max_min.max = right_subarray.max;
    if(left_subarray.min < right_subarray.min)
      max_min.min = left_subarray.min;
    else
      max_min.min = right_subarray.min;
    step_required++;
    return max_min;
  }
}
int main(void)
{
  ifstream read_file("input.txt");
  int INPUT_SIZE;
  int iterator;
  struct couple max_min;
  cout << "Enter input size: ";
  cin >> INPUT_SIZE;
  int input[INPUT_SIZE];
  for(iterator=0; iterator<INPUT_SIZE; iterator++)
    read_file >> input[iterator];
  max_min = MAX_MIN(input, 0, INPUT_SIZE-1);
  cout << "Max = " << max_min.max << endl;
  cout << "Min = " << max_min.min << endl;
  cout << "Required steps = " << step_required << endl;
}
```

Theoretical Complexity:

Since, this program divides every element and checks all of them, the complexity becomes *O(n)*.

Required Steps Comparison:

K	Brute Force	Divide Conquer (Case 1)	Divide Conquer (Case 2)
50000	49999	32767	49999
100000	99999	65535	99999
200000	199999	131071	199999
300000	299999	168927	299999
400000	399999	262143	399999

Discussion:

Although divide and conquer approach case 1 gives better performance than brute force, divide and conquer approach case 2 provides no such efficiency at all.