Heaven's Light is Our Guide

Computer Science & Engineering
Rajshahi University of Engineering & Technology

Lab Manual

Module-03

Course Title : Sessional based on CSE 2201

Course No. : CSE 2202

Experiment No. 3

Name of the Experiment: Complexity analysis of Finding Maximum and Minimum Value [Max-Min Algorithm] and various sorting algorithms [Quick Sort and Merge Sort].

Date: 3rd Cycle

Algorithms (Divide-and-Conquer method)**:**

- **Finding the Maximum and Minimum**
- **Quick Sort**
- **Merge Sort**

Finding the Maximum and Minimum:

Study from page 139 to 144 [Book: Fundamentals of Computer Algorithms
Author: Sahni]

Report should contain:

1. Machine configuration.
2. Complete Table 1.
3. Plot graphical output [x-axis=number of data, y-axis=Required Time][Using any Language].
4. Plot bar graph [using Microsoft Office Excel].
5. Properly analysis of **table** and **graph**.
6. Derive Time complexity of the these algorithms and prepare proper analysis of those complexities.

[**N.B.** Pick randomly 1000 numbers ranging from 1 to 30,000 then write them in a file. Read them from the file and search a particular data and find maximum and minimum data.

Example:

Generate 5 numbers randomly ranging 1 to 20.

2 4 1 12 18

Write them in a file [named input.txt]. Then search a particular data and find maximum and minimum data.

	Required Time	
No. of Data	**Brute-force method**	**Max/Min**
1000		0
2500		0.001
5000		
7500		
10000		
12500		
15000		
17500		
20000 etc.		

Table 1

Quick Sort:

Quick sort sorts by employing a divide and conquer strategy to divide a list into two sub-lists.

The steps are:

7. Pick an element, called a pivot, from the list.
8. Reorder the list so that all elements which are less than the pivot come before the pivot and so that all elements greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.
9. Recursively sort the sub-list of lesser elements and the sub-list of greater elements.

The base case of the recursion is lists of size zero or one, which are always sorted.

A Step through Example:

This is the initial array that you are starting the sort with

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

The array is pivoted about its first element p =3

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

Find first element larger than pivot (underlined) and the last element not larger than pivot (italicized)

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

Swap those elements

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

Scan again in both directions

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

Swap

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

Scan

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

The pointers have crossed: swap pivot with italicized.

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

Pivoting is now complete. Recursively sort sub arrays on each side of pivot.

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

The array is now sorted.

Pseudo-code:

A simple way to express Quick sort in pseudocode is as follows:

```
  function partition(array, left, right, pivotIndex)
     pivotValue := array[pivotIndex]
     swap array[pivotIndex] and array[right] // Move pivot to
     end storeIndex := left
     for i from left to right − 1
        if array[i] ≤ pivotValue
           swap array[i] and array[storeIndex]
           storeIndex := storeIndex + 1
     swap array[storeIndex] and array[right] // Move pivot to its final
     place return storeIndex

procedure quicksort(array, left,
     right) if right > left
        select a pivot index (e.g. pivotIndex := left)
        pivotNewIndex := partition(array, left, right,
        pivotIndex) quicksort(array, left, pivotNewIndex - 1)
        quicksort(array, pivotNewIndex + 1, right)
```

Complexity: O(nlogn)

Merge Sort:

Conceptually, a merge sort works as follows:

1. If the list is of length 0 or 1, then it is already sorted. Otherwise:
2. Divide the unsorted list into two sub lists of about half the size.
3. Sort each sub list recursively by re-applying merge sort.
4. Merge the two sub lists back into one sorted list.

Example:

Here is an example of this sort algorithm:

34 56 78 12 45 3 99 23

34 56 78 12 | 45 3 99 23

34 56 | 78 12 | 45 3 | 99 23

34 | 56 | 78 | 12 | 45 | 3 | 99 | 23

34 56 | 12 78 | 3 45 | 23 99

12 34 56 78 | 3 23 45 99

3 12 23 34 45 56 78 99

Pseudo-code:

In a simple pseudocode form, the algorithm could look something like this:

```
function mergesort(m)
    var list left, right,
    result if length(m) ≤ 1
        return m

    var middle = length(m) / 2
    for each x in m up to middle
         add x to left
    for each x in m after middle
         add x to right
    left = mergesort(left)
    right = mergesort(right)
    result = merge(left, right)
    return result
```

There are several variants for the merge() function, the simplest variant could look like this:

```
function merge(left,right)
    var list result
    while length(left) > 0 and length(right) >
        0 if first(left) ≤ first(right)
            append first(left) to result
            left = rest(left)
        else
            append first(right) to result
            right = rest(right)
    end while
    if length(left) > 0
        append rest(left) to result
    if length(right) > 0
        append rest(right) to
    result return result
```

Complexity: O(nlogn)

Report should contain:

1. Machine configuration.
2. Complete Table 1.
3. Plot graphical output [x-axis=number of data, y-axis=Time complexity][Using any software or Language].
4. Plot bar graph [using Microsoft Office Excel].
5. Properly analysis of **table** and **graph**.

[**N.B.** Pick randomly 1000 numbers ranging from 1 to 30,000 then write them in a file. Read them from the file and sort them by the described two sorting algorithms. Then write the sorted numbers in that file.

Example:

Generate 5 numbers randomly ranging 1 to 20.

2 4 1 12 18

Write them in a file. Then sort them by those two sorting algorithms and write the sorted numbers in that file.

File content should like as follows:

Input [5] numbers [Ranging from 1 to 20]

2 4 1 12 18

After sorting by Quick

sort 1 2 4 12 18

After sorting by Merge sort

1 2 4 12 18

]

	Required Time	
No. of Data	**Quick Sort**	**Merge Sort**
1000		
2500		
5000		
7500		
10000		
12500		
15000		
17500		
20000 and etc		

Table 2