Lab 10 –Algorithms and Sorting and Time Complexity

Objectives

- Comparing the efficiency of algorithms in terms of time complexity
- Problem solving, algorithm design and recursive thinking
- Design algorithms and write Java programs

Preparation

Chapter 14 and lecture notes-20, 21, 22

Problem Description

Exercise 1: In lecture, we learned how to program binary search and linear search. We also discussed about time complexity. Now in this lab, you are asked to compare these two algorithms for searching and find out which one is more efficient. In order to compare them, counting the number of comparisons to find a given target can be a good metric. However, since the location of target in array can dramatically change the number of comparisons in the linear search, we better calculate the average number of comparisons to find out all elements in the list.

Part A – [30 marks] Use the following main method and write a java program that reports the average number of comparisons in both algorithms, having arrays with different size.

```
public static void main (String[] args)  {
  System.out.printf("\n%10s%-25s\n", " ", "Average Number of Comparisons");
  System.out.printf("%-10s%12s%20s\n", "n", "Linear Search", "Binary Search");
  System.out.println("-----------------------------------------");
  for (int size=10; size <1000000; size*=10) {
     int [] list = genArray(size);  //generates an array of random
                                    //integers with the given size.
     // Range of random integers is [0, size).
     // develop the code to find out the average comparisons for
     // searching every element in the list using linear search and binary search
     // To sort your data use Arrays.sort(list)

     System.out.printf("%-10d%-20.2f%-20.2f\n", size, avgLnS, avgBS);
  }
}
```

The run of your program looks like the following output (? stands for the data that your program is producing):

```
          Average Number of Comparisons
n         Linear Search       Binary Search
-----------------------------------------
10        ?                   ?
100       ?                   ?
1000      ?                   ?
10000     ?                   ?
100000    ?                   ?
```

Part B - [2 marks] According to your results, which algorithm needs less comparisons to find out the solution. **(answers.pdf)**

Part C - [4 marks] Provide an approximate function (such as n, n^2, 2^n or log n) that gives you the number of comparisons for each search algorithm. **(answers.pdf)**

Part D - [4 marks] if you are to choose between linear search and binary search, which one would you choose? **(answers.pdf)**

Exercise 2 [10 marks] : Merge Sort is a **sorting** algorithm, which is commonly used in computer science. **Merge Sort** is a divide and conquer algorithm. It **works** by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem. So, Merge Sort first divides the array into two halves and then combines them in a sorted manner.

The following program is an implementation of merge sort. In order to complete this algorithm, you must write a method named **merge** that given two sorted list in ascending order, it returns another list which is merging both lists into a sorted list.

```
public class MergeSort{
    public static void main(String[] args)  {
        int [] list1 = genArray(20);
        printArray("The array before merge sort: ", list1);
        mergeSort(list1);
        printArray("The array After merge sort: ", list1);
    }

    public static void mergeSort(int [] list){
        if (list.length > 1 ){
            int mid = list.length / 2 ;

            int [] firstHalf = new int[mid];
            copyArray(list, firstHalf, 0, mid);
            mergeSort(firstHalf);  //sorts firstHalf in ascending order by recursion

            int [] secondHalf = new int[list.length - mid];
            copyArray(list, secondHalf, mid, list.length);
            mergeSort(secondHalf); //sorts secondHalf in ascending order by recursion

            // merges to lists ie. firstHalf and second half into a sorted list
            merge(firstHalf, secondHalf, list);
        } //base case
    }

    // generates and returns an array of random numbers of the given size
    public static int[] genArray(int size){
        int [] list = new int [size];
        for (int i = 0; i< size; ++i)
            list[i] = (int)(Math.random() * 100 );
        return list;
    }
```

```
    // prints an array followed by the header
    public static void printArray(String header, int [] arr) {
        System.out.println(header);
        for (int i = 0; i < arr.length; i++)
            System.out.print(arr[i] + " " );
        System.out.println();
    }

    //copies from index start to index end of sourceLs to destLs. End is exclusive
    public static void copyArray(int [] sourceLs, int[] destLs, int start, int end){
       for (int i = start; i< end ; i++)
            destLs[i-start] = sourceLs[i];
    }

    // develop merge method to complete the program

}
```

Submission

Make a folder containing your source code (.java files) and pdf file (answers.pdf) and submit the zip file to D2L. No external documentation is needed for this lab.