**Lab 11 Sorting**

Name: \_\_\_\_\_\_Theerawit Plukmontol\_\_\_\_\_\_\_\_\_\_

ID: \_\_6538090521\_­\_\_\_\_

No coding in this lab! :-D

The objective of this lab is to analyze time complexity of sorting algorithms by experiments.

The provided program contains all sorting algorithms. The code in the program may look a little different from the code in lecture slides but they uses the same principles.

To help you understand how each algorithm works:

* Watch this video <http://img-9gag-fun.9cache.com/photo/aPyoG4P_460sv_v1.mp4>

Follow the instructions and answer question 5-8:

1. Select an algorithm from the list in line 34-40 of the given code by putting // in front of other algorithms. The algorithms to be used in this lab are bubble sort, selection sort, insertion sort, merge sort and quicksort.
2. Select one of the for loops in line 15 and 16 based on the selected algorithm.
3. Run the Sorting program. Do not run other applications while the Sorting program is running.
4. The program will create an array of size n, populate the array with data, sort the array, check the results, and print out the execution time for sorting. It will vary the size of the array and also vary the initial order of data (sorted, random, and reversed order).
5. For each algorithm and each initial order, create a line graph between data size and execution time. There will be 15 lines in total but you can put the graphs of the same algorithm in one plot. You can copy the output from Eclipse into the Excel file to create graphs.

1. Based on the experimental result, determine the time complexity of each algorithm in terms of Big O and fill in the table.

**Time Complexity**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Ordered | Random | Reverse |
| Bubble Sort | n | n^2 | n^2 |
| Selection Sort | n^2 | n^2 | n^2 |
| Insertion Sort | n | n^2 | n^2 |
| Merge Sort | nlogn | nlogn | nlogn |
| Quicksort | nlogn | nlogn | nlogn |

1. Which algorithm in each group is the fastest? What is the reason?

7.1) Bubble, Selection, Insertion

Insertion Sort

Insertion sort works by building the sorted part of the array one by one. It only needs to shift elements to make space for a new element at the correct position. This shifting typically involves fewer swaps compared to bubble and selection sort, especially when the data is partially sorted.

Insertion sort only needs to compare the new element with a smaller number of elements (already sorted part of the array) to find the correct position. This reduces the total number of comparisons compared to bubble and selection sort, especially for data that is partially sorted.

7.2) Merge, Quick

Quick Sort

due to lower constant factors involved in the operation. This is because Quick Sort typically partitions the data efficiently, leading to fewer comparisons and swaps on average.

1. For each algorithm, how is it sensitive to the initial order of data? (Does it run much faster or slower when the data is initially sorted, random, or reversed?) Why?

Bubble Sort:

Will run as fast as possible for sorted data, because no swaps are needed after the first pass.

Will run as slow as possible in reversed data because it needs to perform the maximum number of swaps to rearrange all elements.

Selection Sort:

Performance is unaffected by the initial order of data, because it involves the same number of comparisons and swaps regardless of the arrangement of elements.

Insertion Sort:

Will run as fast as possible for sorted data, because it only needs to insert each element into its already correct position.

Will run as slow as possible (O(n^2)) because each insertion involves shifting many elements to make space.

Merge Sort:

Performance is unaffected by the initial order of data. Always runs in O(n log n) time due to its consistent divide-and-conquer approach.

Quicksort:

Will run as slow as possible in sorted and reversed data, because it will chooses a bad pivot element, creating uneven partitions.

1. Submit this file. Name it YourID\_Lab08\_Sorting, where YourID is your student ID.

NOTE: A program may take a long time to run!!!