Lab 11 Sorting

Name: ____

ID: ____

No coding in this lab! :-D

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

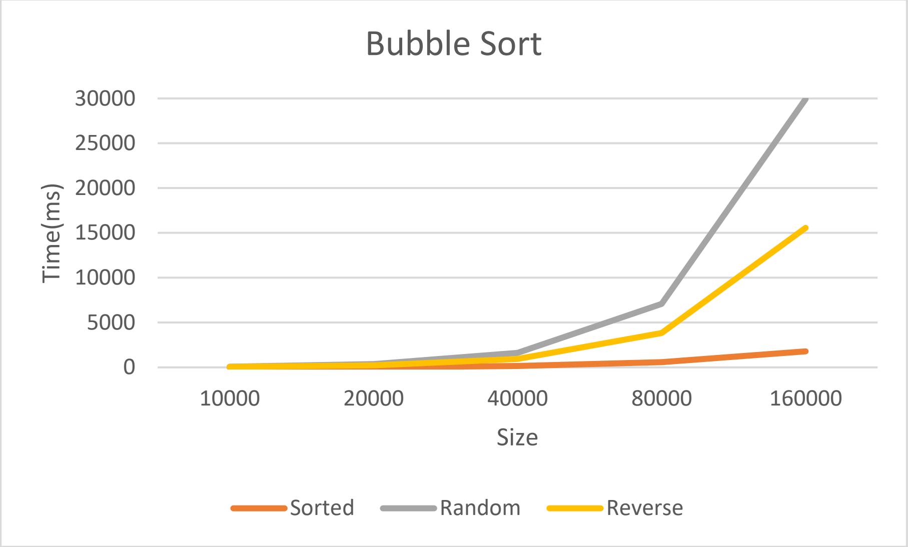
The provided program contains all sorting algorithms. The code in the program may look a little different from the code in lecture slides but they uses the same principles.

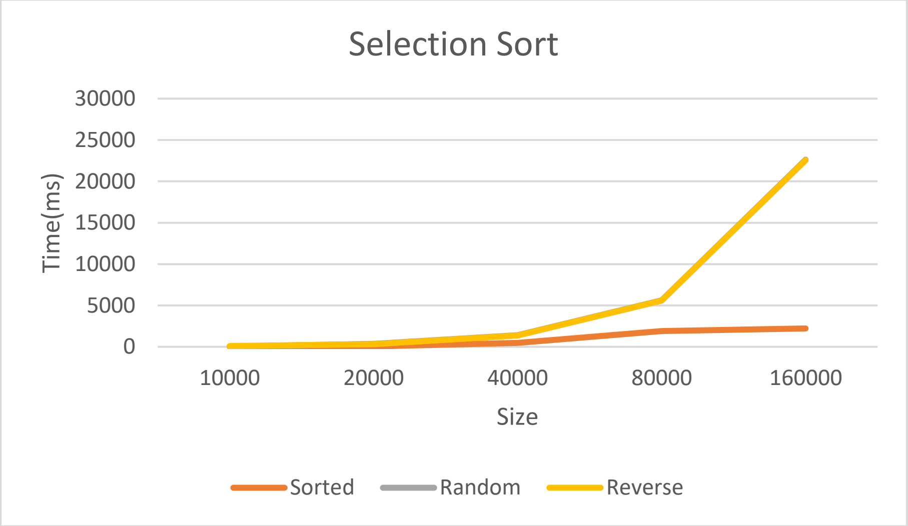
To help you understand how each algorithm works:

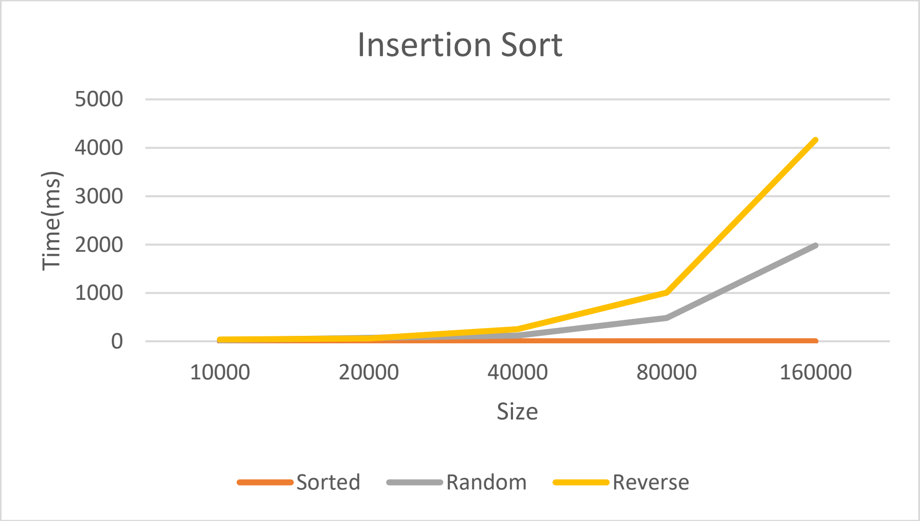
- Watch this video http://img-9gag-fun.9cache.com/photo/aPyoG4P_460sv_v1.mp4

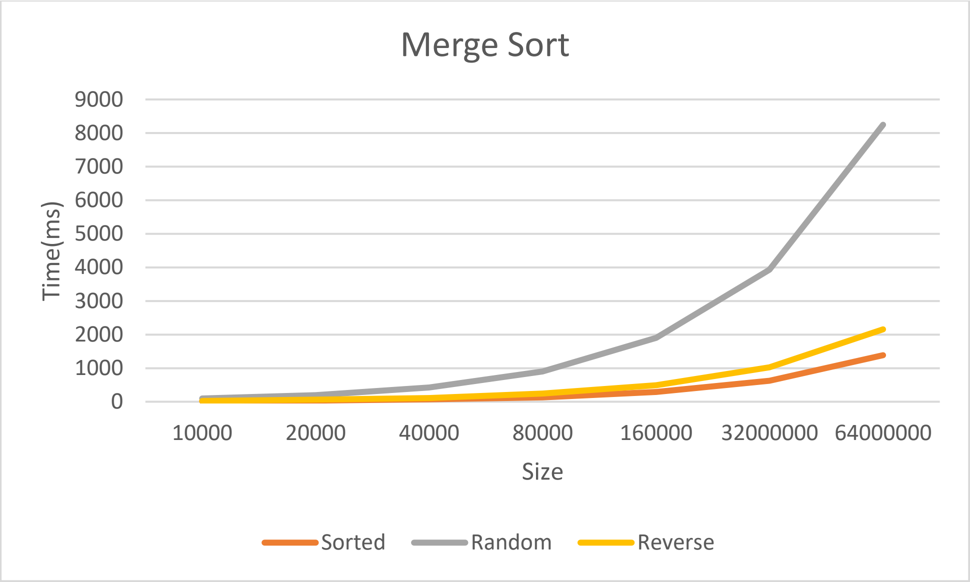
Follow the instructions and answer question 5-8:

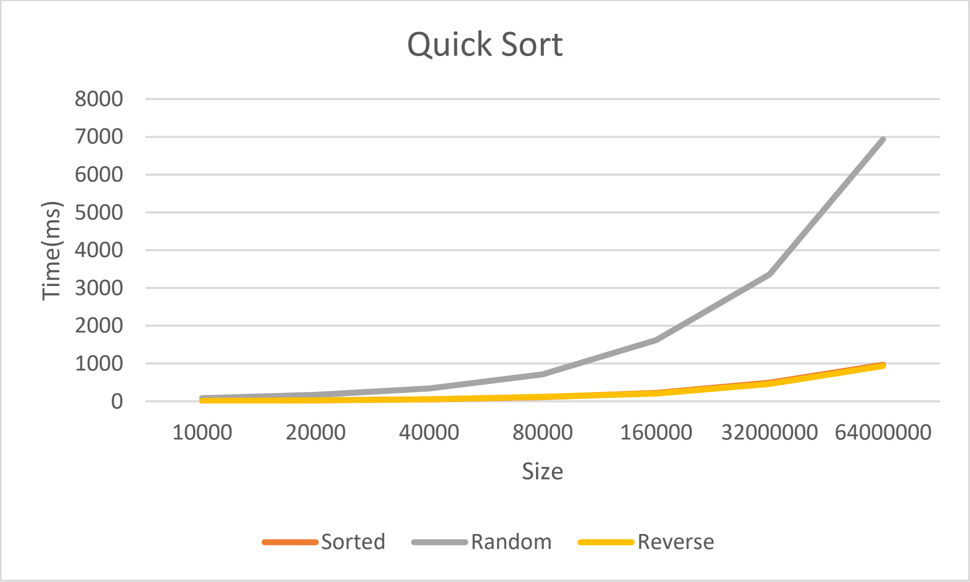
1. Select an algorithm from the list in line 34-40 of the given code by putting // in front of other algorithms. The algorithms to be used in this lab are bubble sort, selection sort, insertion sort, merge sort and quicksort.
2. Select one of the for loops in line 15 and 16 based on the selected algorithm.
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.


Bubble Sort
Time(ms)
30000
25000
20000
15000
10000
5000
0
10000
20000
40000
80000
160000
Size
Sorted
Random
Reverse


Selection Sort
Time(ms)
30000
25000
20000
15000
10000
5000
0
10000
20000
40000
80000
160000
Size
Sorted
Random
Reverse


Insertion Sort
Time(ms)
5000
4000
3000
2000
1000
0
10000
20000
40000
80000
160000
Size
Sorted
Random
Reverse





6. 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	O(n^2)	O(n^2)	O(n^2)
Selection Sort	O(n^2)	O(n^2)	O(n^2)
Insertion Sort	O(1)	O(n^2)	O(n^2)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)
Quicksort	O(nlogn)	O(nlogn)	O(nlogn)

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

   7.1) Bubble, Selection, Insertion

   Insertion sort is faster as it makes less comparison between elements. Insertion sort only compares elements in the sorting part while others compared it with entire list.

   7.2) Merge, Quick

   Quick Sort is generally faster than merge sort because partitioning in quick sort requires less operation than merge sort.

8. 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: Initial order is important since bubble sort needs to perform its full number of swaps to complete the sorting.

   Selection Sort: Initial order is not important for general use case since the algorithm scans the entire list to find the minimum and swap it. For sorted list, it is faster since it doesn't need to swap the element.

   Insertion Sort: Initial order is important because insertion sort needs to shift elements multiple times to insert each element into its correct position. For sorted list, it is very fast since there is no need to shift elements.

   Merge Sort: Initial order doesn't matter since the data is divides into halves recursively and it later get merged.

   Quicksort: Initial order matters since the pivot of this algorithm is chosen from the initial order. A bad selection of pivot may result in unbalanced partitions slowing the algorithm down.

9. Submit this file. Name it YourID_Lab08_Sorting, where YourID is your student ID.

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