Lab 9 Sorting

Name: ___Supatach Vanichayangkuranont_____

ID: ____6238211921_____

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.

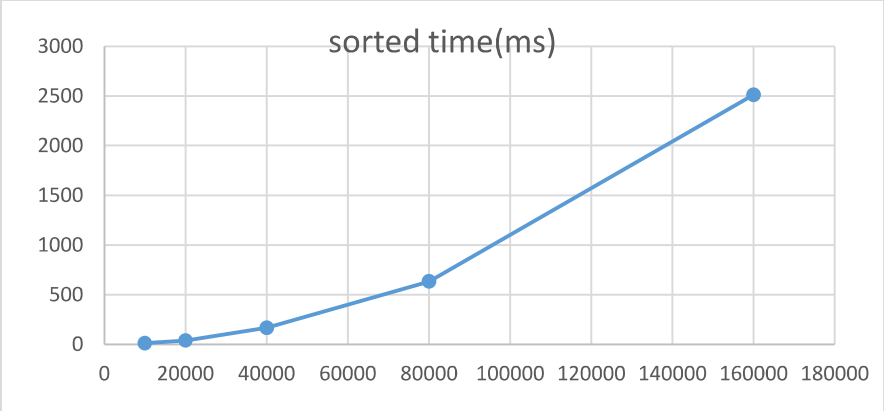
To help you understand how each algorithm works:

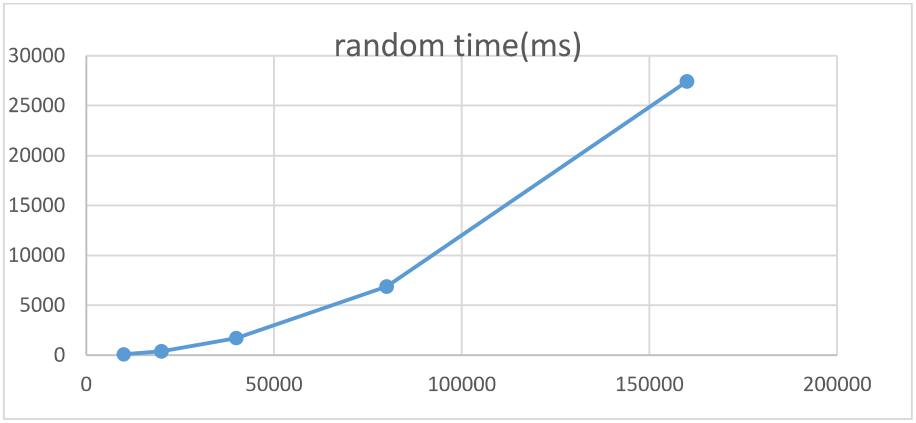
- Watch this video http://img-9gag-fun.9cache.com/photo/aPyoG4P_460sv_v1.mp4

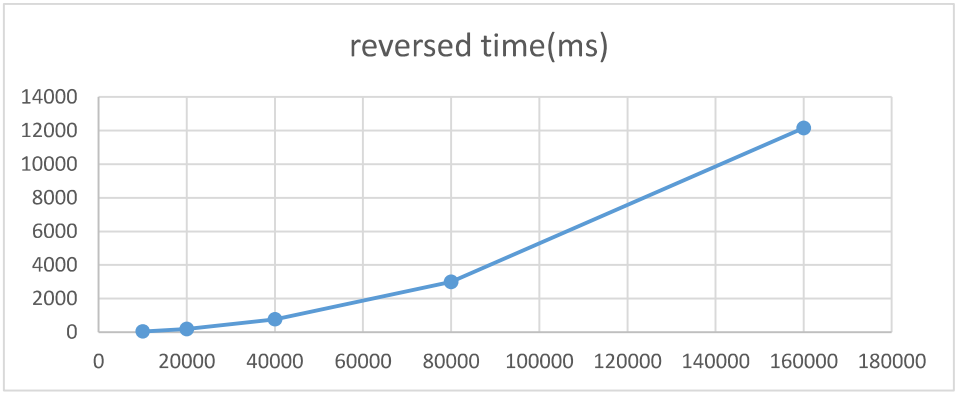
Follow the instructions and answer question 5-8:

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.

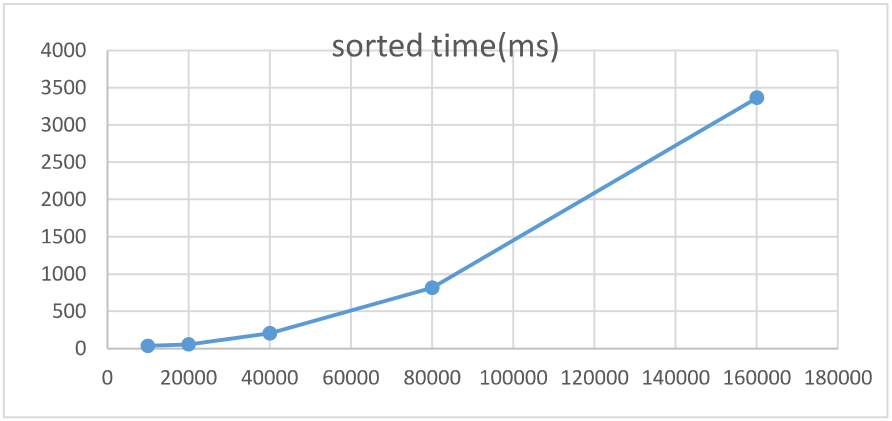
Bubble

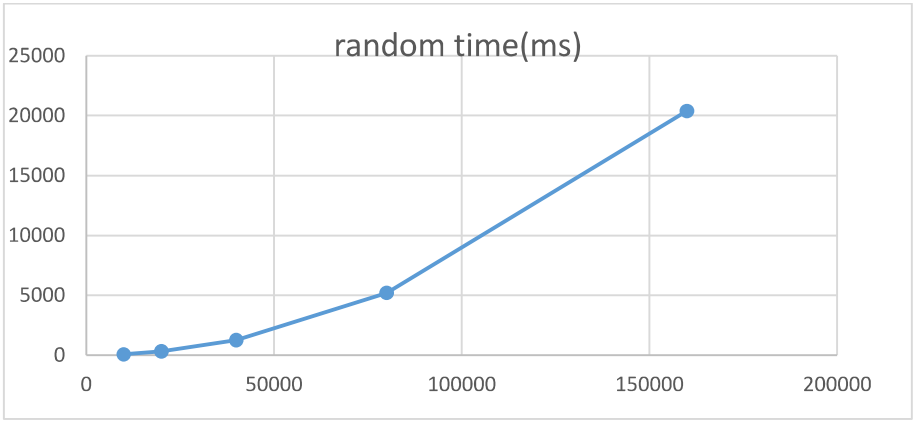


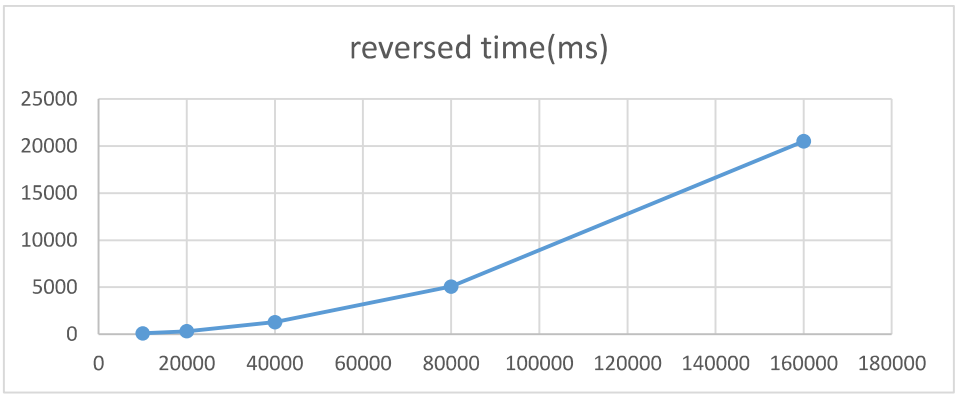

random time(ms)
30000
25000
20000
15000
10000
5000
0
0
50000
100000
150000
200000


reversed time(ms)
14000
12000
10000
8000
6000
4000
2000
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

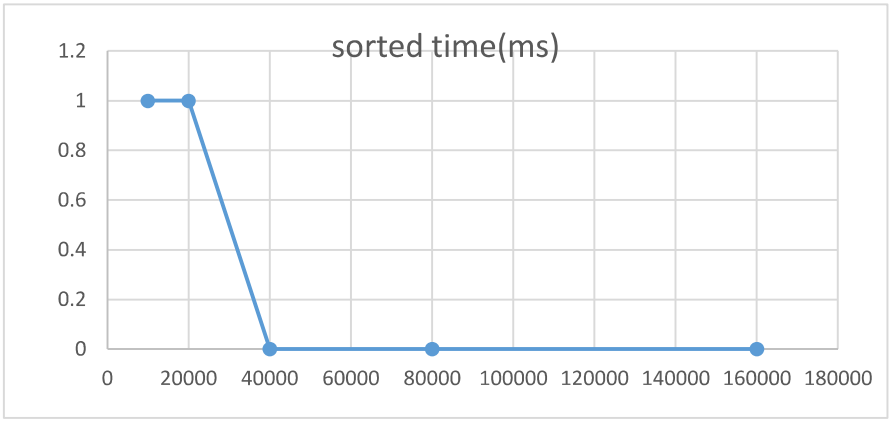
Selection

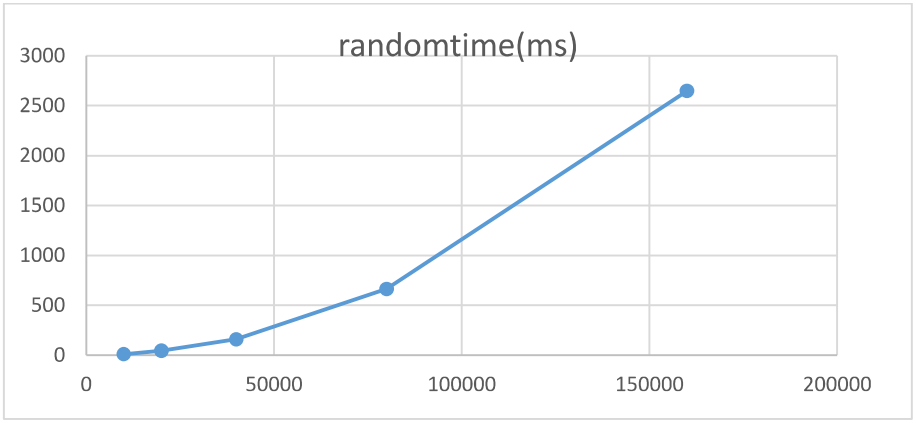

sorted time(ms)
4000
3500
3000
2500
2000
1500
1000
500
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

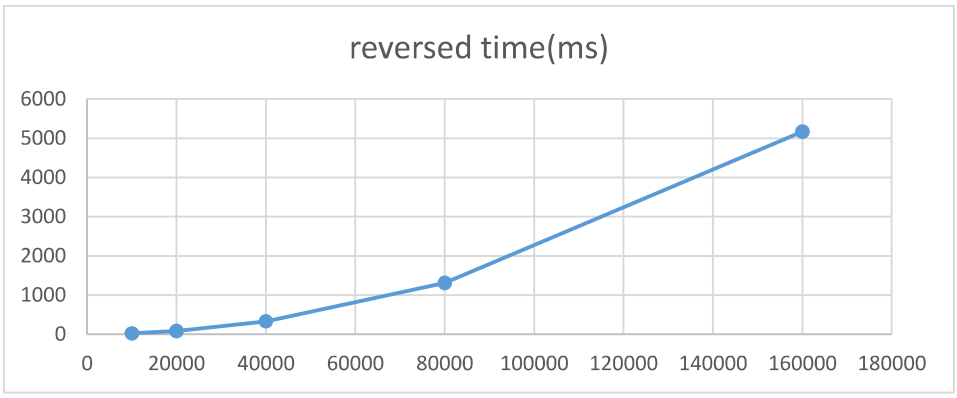

random time(ms)
25000
20000
15000
10000
5000
0
0
50000
100000
150000
200000


reversed time(ms)
25000
20000
15000
10000
5000
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

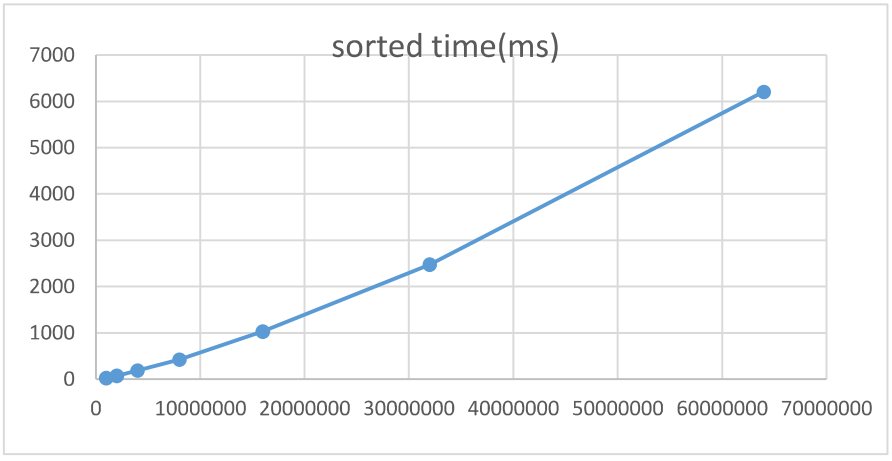
Insertion:

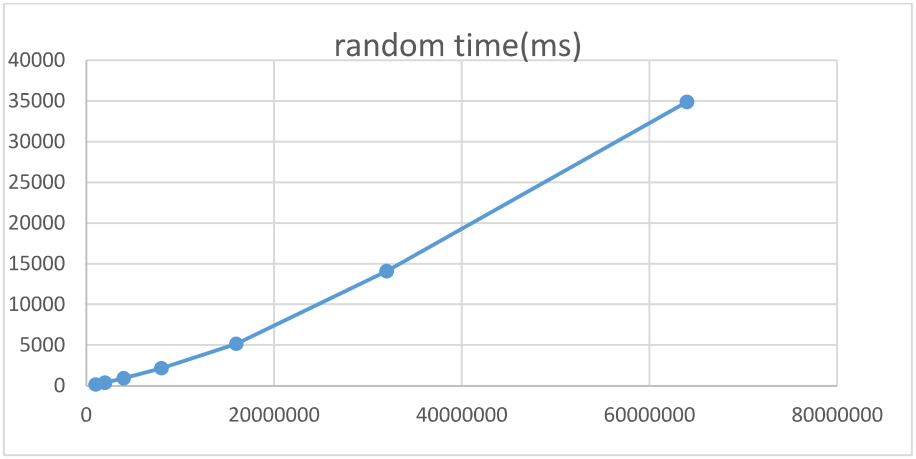

sorted time(ms)
1.2
1
0.8
0.6
0.4
0.2
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

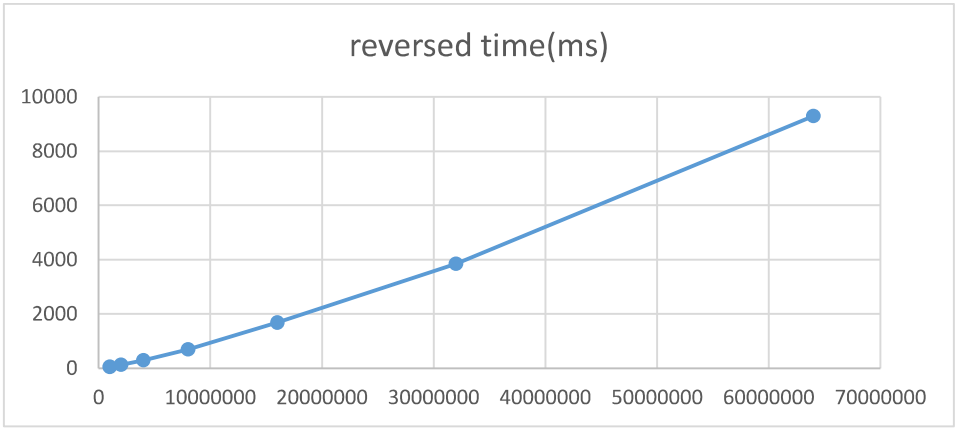

randomtime(ms)
3000
2500
2000
1500
1000
500
0
0
50000
100000
150000
200000


reversed time(ms)
6000
5000
4000
3000
2000
1000
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

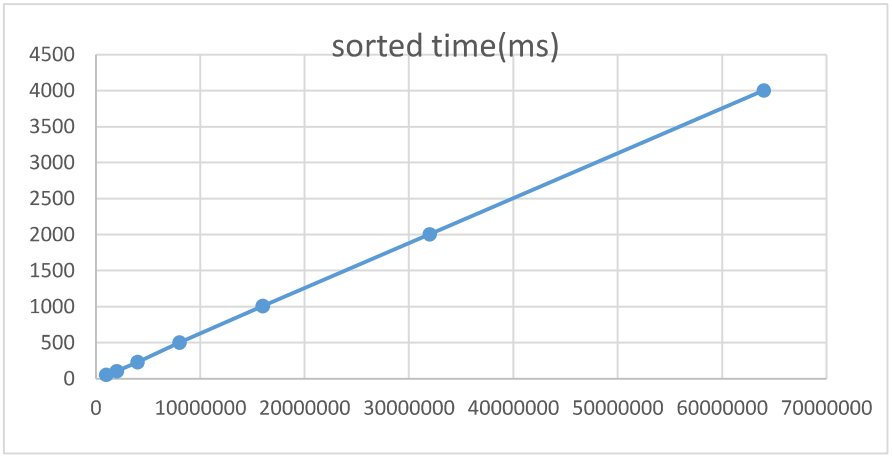
Shell:

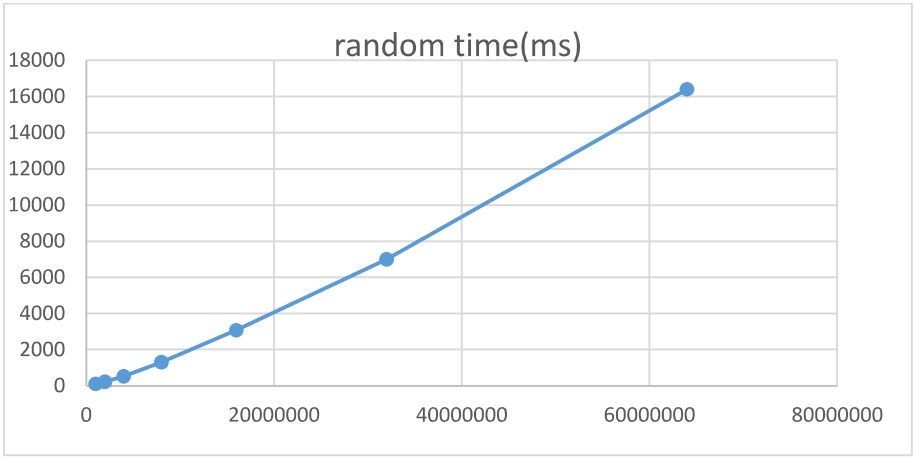

sorted time(ms)
7000
6000
5000
4000
3000
2000
1000
0
0
10000000
20000000
30000000
40000000
50000000
60000000
70000000

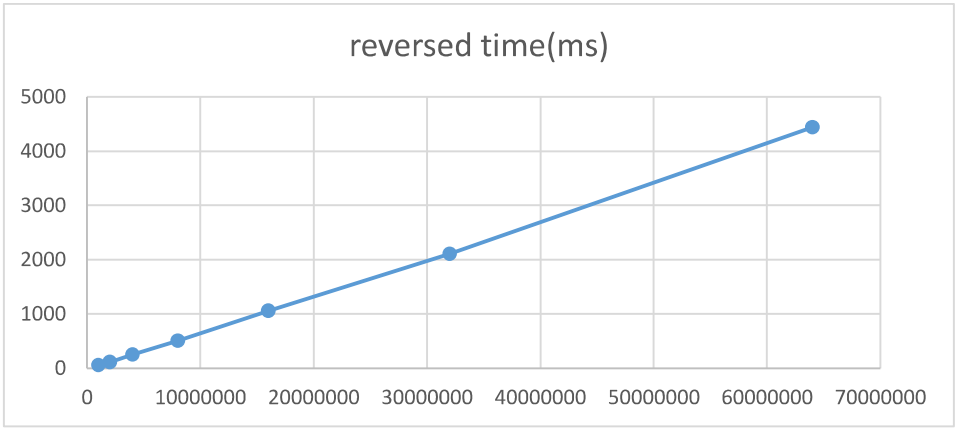

random time(ms)
40000
35000
30000
25000
20000
15000
10000
5000
0
0
20000000
40000000
60000000
80000000


reversed time(ms)
10000
8000
6000
4000
2000
0
0
10000000
20000000
30000000
40000000
50000000
60000000
70000000

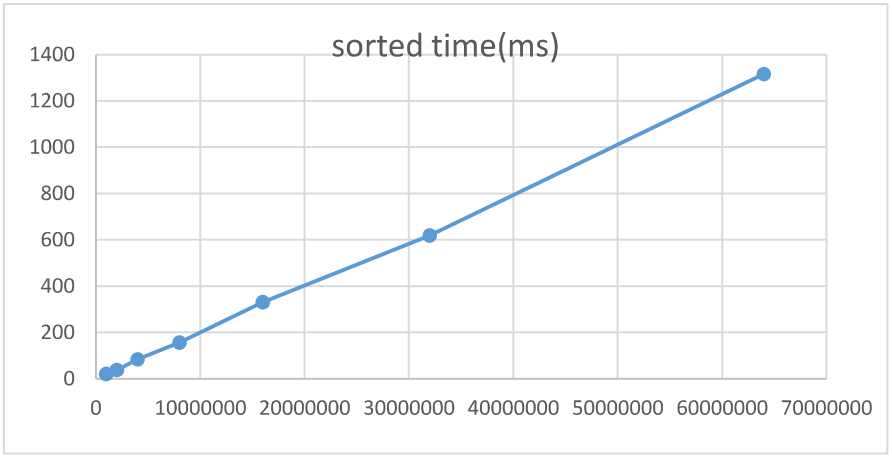
Heap:

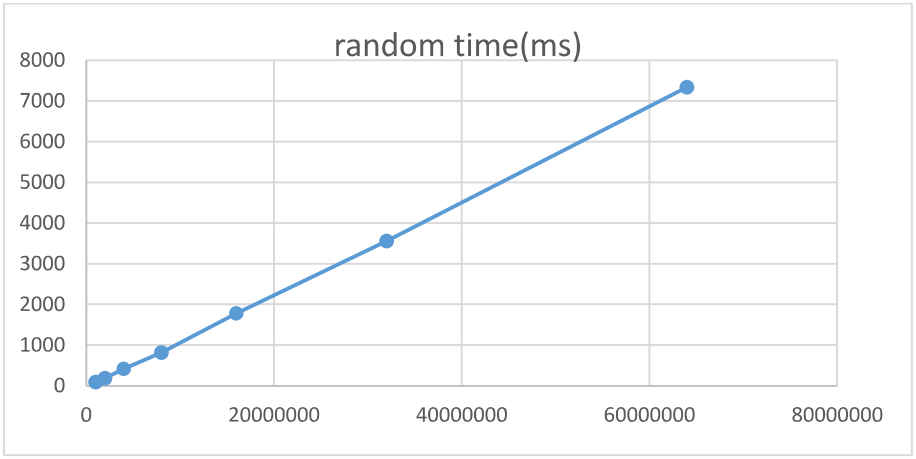

sorted time(ms)
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
10000000
20000000
30000000
40000000
50000000
60000000
70000000

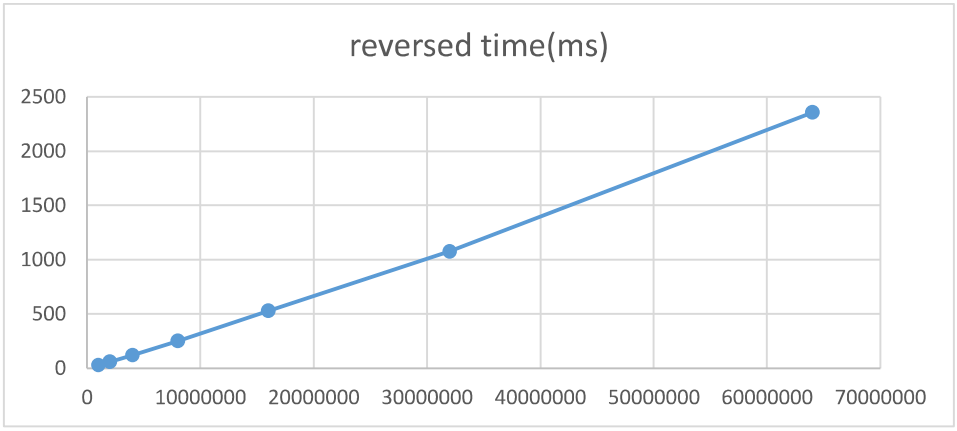

random time(ms)
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0
20000000
40000000
60000000
80000000


reversed time(ms)
5000
4000
3000
2000
1000
0
0
10000000
20000000
30000000
40000000
50000000
60000000
70000000

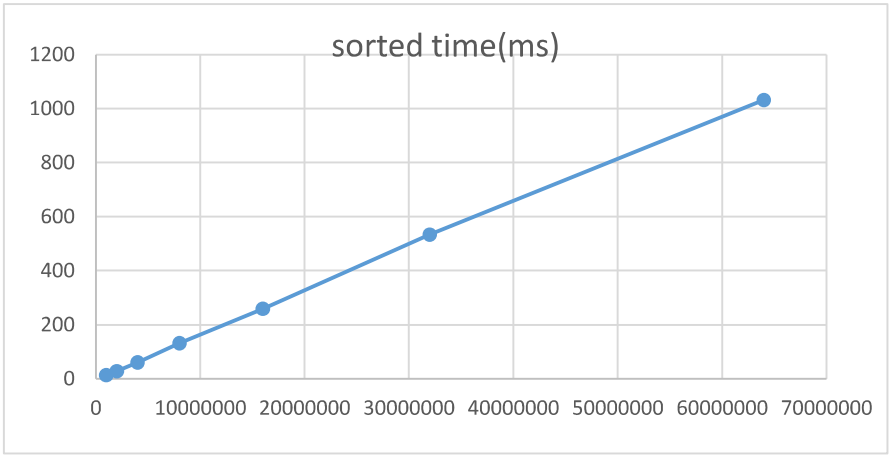
Merge:

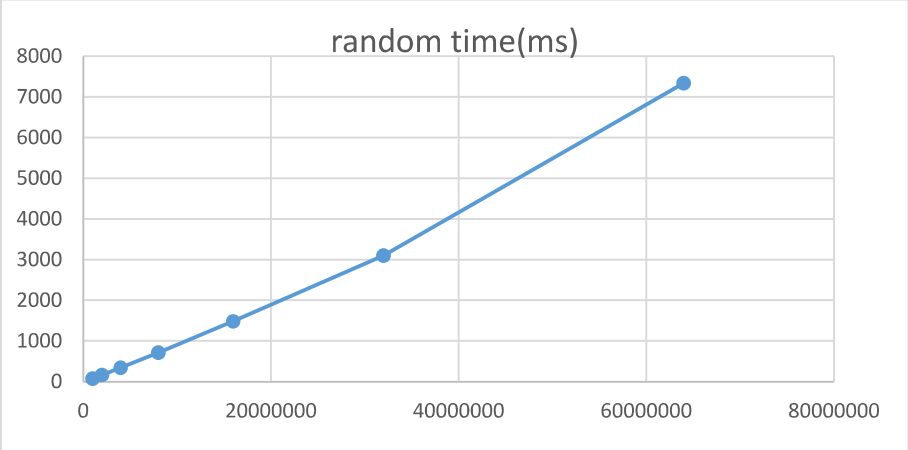

sorted time(ms)
1400
1200
1000
800
600
400
200
0
0
10000000
20000000
30000000
40000000
50000000
60000000
70000000

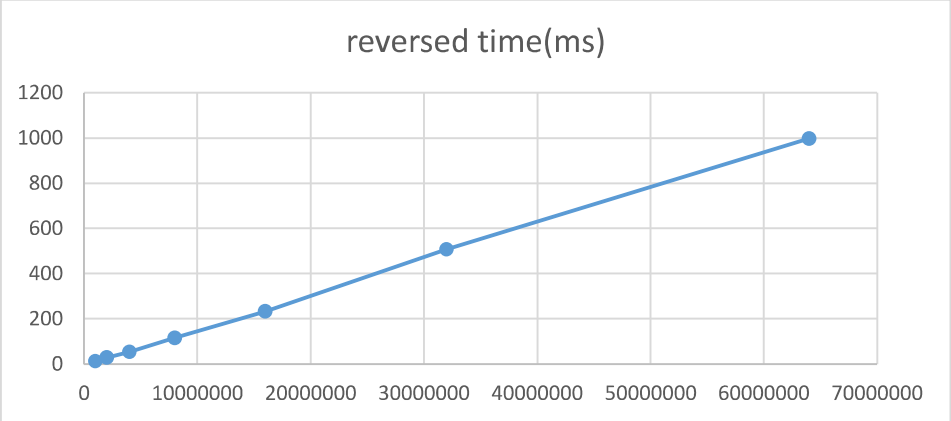

random time(ms)
8000
7000
6000
5000
4000
3000
2000
1000
0
0
20000000
40000000
60000000
80000000


reversed time(ms)
2500
2000
1500
1000
500
0
0
10000000
20000000
30000000
40000000
50000000
60000000
70000000

Quick:


sorted time(ms)
1200
1000
800
600
400
200
0
0
10000000
20000000
30000000
40000000
50000000
60000000
70000000





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	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	n^2	nlogn

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

7.1) Bubble, Selection, Insertion

Insertion sort as for ordered elements it took the fastest for ordered elements and has same time complexity with others for random and reverse elements.

7.2) Merge, Quick

Merge sort as with all cases (ordered random and reverse) it has a time complexity of nlogn while quick sort may have a time complexity of n^2 as its worst case.

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: Run much faster when initial data is sorted then reversed then slowest would be random. This is because when the data is initially ordered, bubble sort comparing adjacent data will pass through all data once without having to swap or come back to swap

   Selection Sort: Run much faster when initial data is ordered. The random and reverse time complexity is similar. This is because selection sort starts by looking for the smallest number that is less than the first index throughout the array then moves on to the next index; however when it is all ordered, this means there are no swaps and selection sort passes through all indices once.

   Insertion Sort: Run very fast when initial data is ordered. While random and reverse took around the same time. This is because for insertion sort, the algorithm checks for swaps if the next number is smaller than previous; so on ordered elements, it always finds a larger number than its previous, so the no swaps happen. While for random and reverse it will need to swap as the next element may not be always larger than its previous elements

   Merge Sort: Merge sort has n^2 time complexity for all cases due to its algorithm of divide and conquering by half each time and iterating through pairs of two for swaps; hence this does not affect time complexity even in ordered cases as it has to divide to compare pairs anyway.

   Quicksort: Has worst time complexity on random elements while sorted and reverse has time complexity of n^2. This is because quick sort does not always partition into equal parts so it may end up partitioning all elements which will be compared again; leading to a worst case of n^2 time complexity. Meanwhile, it's best case and averagely will run on nlogn as it tries to partition into more equal parts.

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!!!