Name: Jared Diehl
Course: CS 3130
Professor: Galina Piatnitskaia
Date: March 16, 2020

Analysis of Sorting Algorithms

Introduction

A sorting algorithm is an algorithm that puts elements of a list in a specific order. This document looks at benchmarks of various sorting algorithms using integer arrays. The data will be presented in tables, line graphs, and bar graphs. Although, quicksort and mergesort may appear obscured on the charts because one is overlapping the other.

Definition

Selection Sort

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1. The subarray which is already sorted.
2. Remaining subarray which is unsorted.

Insertion Sort

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Quick Sort

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

Merge Sort

Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. The merge() function is used for merging two halves. The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one.

Time Complexities

Algorithm	Best	Average	Worst
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Bubble Sort (without swaps)	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
Bubble Sort (with swaps)	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Quick Sort	$\Omega(n \cdot log(n))$	$\Theta(n \cdot log(n))$	$O(n^2)$
Merge Sort	$\Omega(n \cdot log(n))$	$\Theta(n \cdot log(n))$	$O(n \cdot log(n))$

Attributes

Algorithm	Non-Adaptive	In-Place	Stable
Selection Sort	Yes	Yes	No
Insertion Sort	No	Yes	Yes
Bubble Sort (without swaps)	Yes	Yes	Yes
Bubble Sort (with swaps)	No	Yes	Yes
Quick Sort	No	Yes	No
Merge Sort	Yes	No	Yes

Objective

Run the sorting algorithms for the types of array containing:

1. 1000 integers: random numbers, sorted list, almost sorted list.
2. 10000 integers: random numbers, sorted list, almost sorted list.

3. 100000 integers: random numbers, sorted list, almost sorted list.

Using the results from the experiments, determine if they support the theoretical information about the efficiency of each sorting algorithm. Lastly, conclude which sorting algorithms work better for specific types of input.

Computer Specifications

OS Name: Microsoft Windows 10 Home
Version: 10.0.18363 Build 18363
System Manufacturer: TOSHIBA
System Model Satellite: P855
System Type: x64-based PC
Processor: Intel Core i5-3210M CPU @ 2.50GHz, 2501 Mhz, 2 Cores, 4 Logical Processors
Installed Physical Memory (RAM): 8.00 GB

Programming Language Specifications

Java 1.8.0_241
Java(TM) SE Runtime Environment (build 1.8.0_241-b07)
Java HotSpot(TM) 64-Bit Server VM (build 25.241-b07, mixed mode)

Experimental Data

Random Times (1000) (seconds)

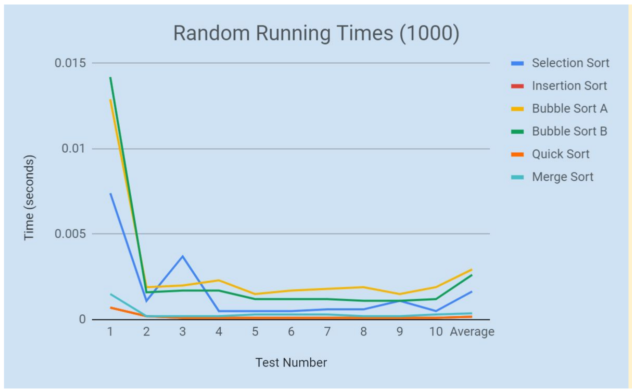
Test Number	Selection Sort	Insertion Sort	Bubble Sort A	Bubble Sort B	Quick Sort	Merge Sort
1	0.0074	0.0007	0.0129	0.0142	0.0007	0.0015
2	0.0011	0.0002	0.0019	0.0016	0.0002	0.0002
3	0.0037	0.0001	0.002	0.0017	0.0001	0.0002
4	0.0005	0.0001	0.0023	0.0017	0.0001	0.0002
5	0.0005	0.0001	0.0015	0.0012	0.0001	0.0003
6	0.0005	0.0001	0.0017	0.0012	0.0001	0.0003
7	0.0006	0.0001	0.0018	0.0012	0.0001	0.0003
8	0.0006	0.0001	0.0019	0.0011	0.0001	0.0002
9	0.0011	0.0001	0.0015	0.0011	0.0001	0.0002
10	0.0005	0.0001	0.0019	0.0012	0.0001	0.0003
Average	**0.00165**	**0.00017**	**0.00294**	**0.00262**	**0.00017**	**0.00037**

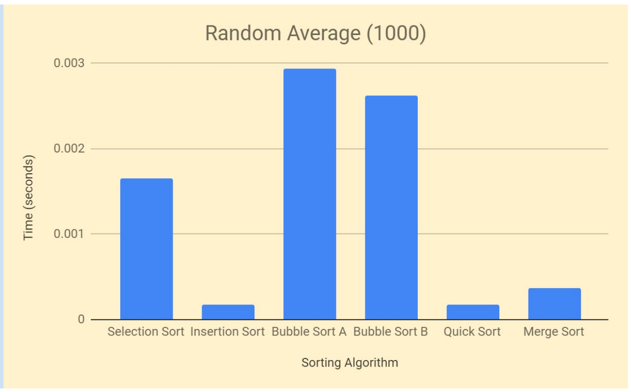
Random Times (10000) (seconds)

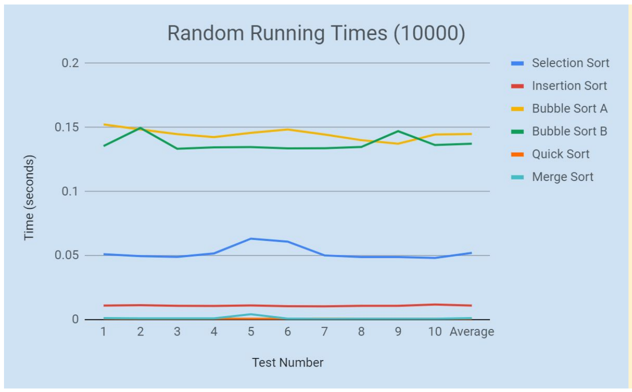
Test Number	Selection Sort	Insertion Sort	Bubble Sort A	Bubble Sort B	Quick Sort	Merge Sort
1	0.0511	0.011	0.1524	0.1355	0.0009	0.0012
2	0.0495	0.0113	0.1484	0.1496	0.0007	0.001
3	0.0489	0.0108	0.1448	0.1334	0.0007	0.001
4	0.0517	0.0107	0.1425	0.1345	0.0007	0.001
5	0.0632	0.0111	0.1458	0.1347	0.0007	0.0042
6	0.0609	0.0105	0.1484	0.1337	0.0007	0.0008
7	0.0501	0.0104	0.1445	0.1338	0.0007	0.0006
8	0.0488	0.0108	0.1401	0.1348	0.0007	0.0007
9	0.0488	0.0108	0.1373	0.1471	0.0007	0.0007
10	0.0481	0.0118	0.1445	0.1363	0.0007	0.0007
Average	**0.05211**	**0.01092**	**0.14487**	**0.13734**	**0.00072**	**0.00119**

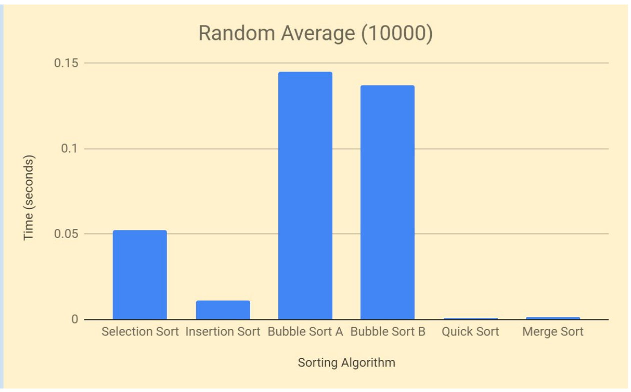
Random Times (100000) (seconds)

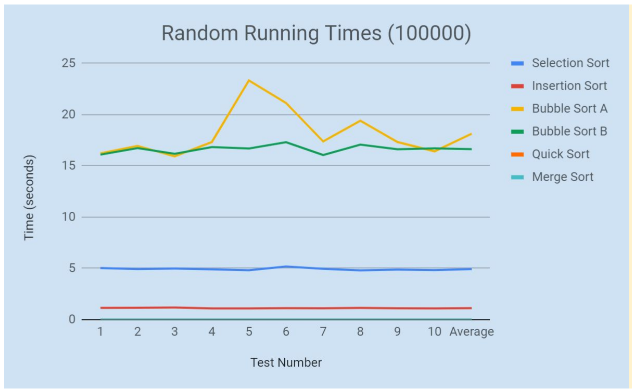
Test Number	Selection Sort	Insertion Sort	Bubble Sort A	Bubble Sort B	Quick Sort	Merge Sort
1	5.0335	1.1428	16.2184	16.0974	0.0084	0.0121
2	4.9345	1.1579	16.9462	16.7333	0.0084	0.0125
3	4.9793	1.1866	15.9324	16.1848	0.0079	0.0126
4	4.9018	1.099	17.3233	16.8344	0.0079	0.0123
5	4.823	1.0972	23.3394	16.6923	0.008	0.0115
6	5.1807	1.1202	21.1394	17.3154	0.0083	0.0128
7	4.9504	1.1022	17.3858	16.0553	0.0102	0.0086
8	4.8027	1.1486	19.4145	17.0752	0.0086	0.0075
9	4.8839	1.1009	17.3276	16.624	0.0083	0.0069
10	4.8302	1.0907	16.4166	16.7108	0.0083	0.0072
Average	**4.932**	**1.12461**	**18.14436**	**16.63229**	**0.00843**	**0.0104**

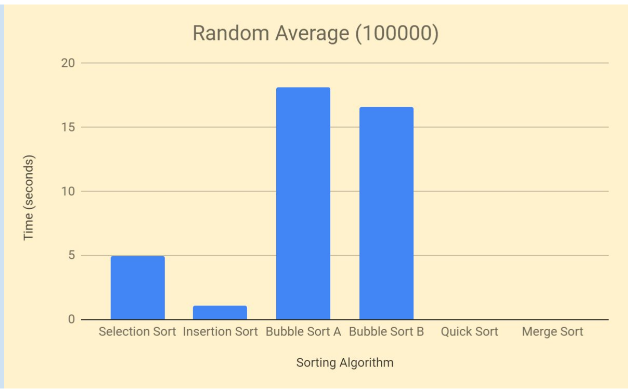












- The bubble sorting algorithm appear to be around the same rate.
- Clearly, quick sort and merge sort also appear to be around the same rate and are the fastest among them all no matter what the input size is so far.

Sorted Times (1000) (seconds)

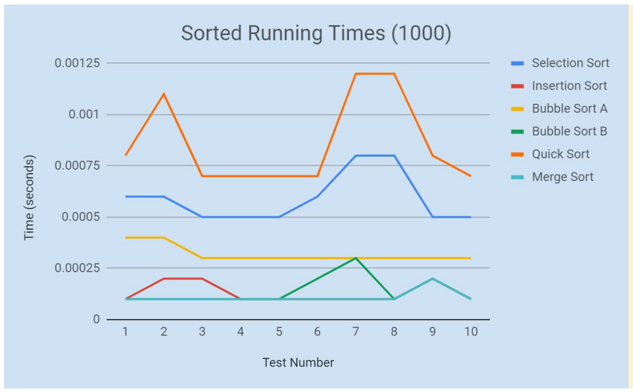
Test Number	Selection Sort	Insertion Sort	Bubble Sort A	Bubble Sort B	Quick Sort	Merge Sort
1	0.0006	0.0001	0.0004	0.0001	0.0008	0.0001
2	0.0006	0.0002	0.0004	0.0001	0.0011	0.0001
3	0.0005	0.0002	0.0003	0.0001	0.0007	0.0001
4	0.0005	0.0001	0.0003	0.0001	0.0007	0.0001
5	0.0005	0.0001	0.0003	0.0001	0.0007	0.0001
6	0.0006	0.0001	0.0003	0.0002	0.0007	0.0001
7	0.0008	0.0001	0.0003	0.0003	0.0012	0.0001
8	0.0008	0.0001	0.0003	0.0001	0.0012	0.0001
9	0.0005	0.0002	0.0003	0.0002	0.0008	0.0002
10	0.0005	0.0001	0.0003	0.0001	0.0007	0.0001
Average	**0.00059**	**0.00013**	**0.00032**	**0.00014**	**0.00086**	**0.00011**

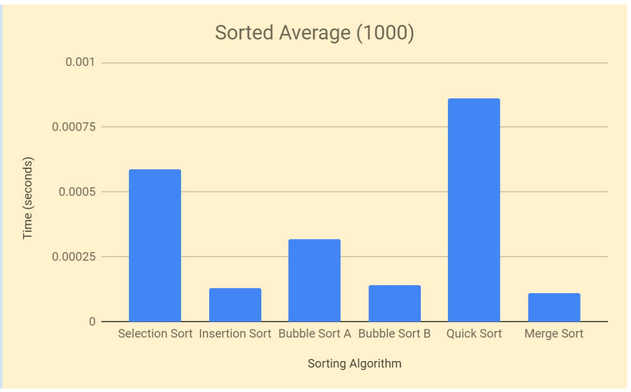
Sorted Times (10000) (seconds)

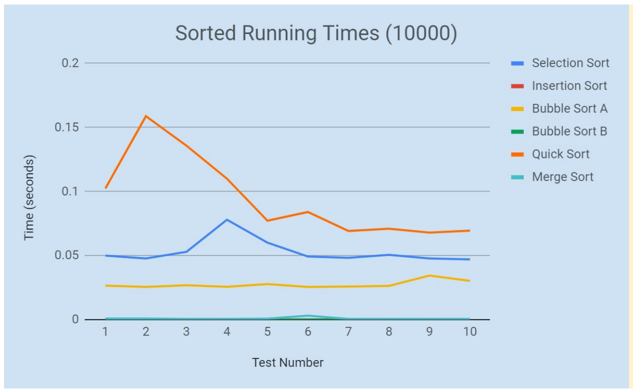
Test Number	Selection Sort	Insertion Sort	Bubble Sort A	Bubble Sort B	Quick Sort	Merge Sort
1	0.05	0.0001	0.0265	0.0001	0.1025	0.0009
2	0.0477	0.0002	0.0255	0.0001	0.1588	0.0009
3	0.0529	0.0002	0.0268	0.0002	0.1358	0.0006
4	0.078	0.0001	0.0256	0.0001	0.11	0.0006
5	0.0601	0.0001	0.0277	0.0001	0.0772	0.0008
6	0.0492	0.0001	0.0255	0.0002	0.084	0.0031
7	0.0482	0.0001	0.0258	0.0003	0.0692	0.0006
8	0.0506	0.0001	0.0263	0.0001	0.071	0.0006
9	0.0477	0.0002	0.0344	0.0002	0.0679	0.0006
10	0.047	0.0001	0.0303	0.0001	0.0694	0.0006
Average	**0.05314**	**0.00013**	**0.02744**	**0.00015**	**0.09458**	**0.00093**

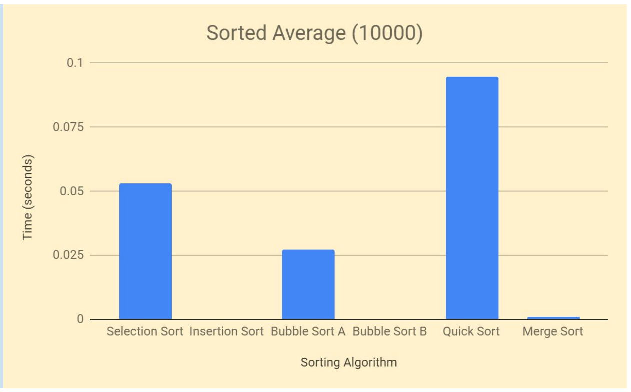
Sorted Times (100000) (seconds)

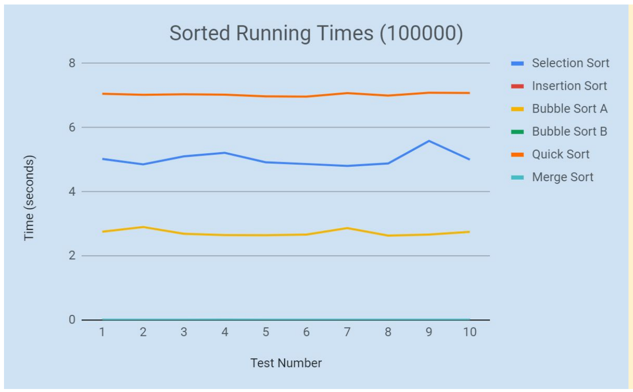
Test Number	**Selection Sort**	**Insertion Sort**	**Bubble Sort A**	**Bubble Sort B**	**Quick Sort**	**Merge Sort**
1	5.0249	0.0003	2.7533	0.0001	7.0546	0.0075
2	4.855	0.0003	2.9007	0.0001	7.0241	0.0068
3	5.1052	0.0003	2.6877	0.0001	7.0398	0.0083
4	5.2135	0.0002	2.6469	0.0001	7.0272	0.0088
5	4.9191	0.0002	2.6457	0.0001	6.9747	0.0071
6	4.8644	0.0002	2.6666	0.0001	6.9662	0.0066
7	4.8055	0.0002	2.865	0.0001	7.0763	0.0072
8	4.8816	0.0002	2.6327	0.0001	6.9985	0.0069
9	5.5858	0.0002	2.6639	0.0001	7.0893	0.0071
10	5.0046	0.0002	2.7511	0.0001	7.0816	0.0069
Average	**5.02596**	**0.00023**	**2.72136**	**0.0001**	**7.03323**	**0.00732**

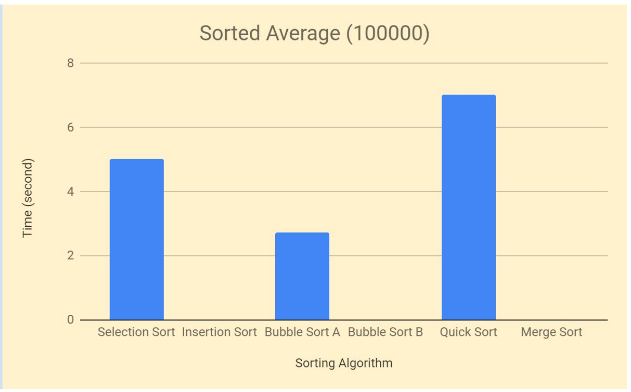












- The algorithms insertion sort, bubble sort without swaps, and merge sort appear to perform at the same rate and faster than the rest of the algorithms.

Almost Sorted Times (1000) (seconds)

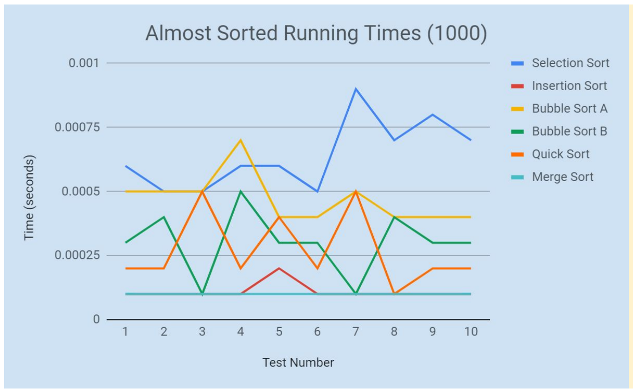
Test Number	Selection Sort	Insertion Sort	Bubble Sort A	Bubble Sort B	Quick Sort	Merge Sort
1	0.0006	0.0001	0.0005	0.0003	0.0002	0.0001
2	0.0005	0.0001	0.0005	0.0004	0.0002	0.0001
3	0.0005	0.0001	0.0005	0.0001	0.0005	0.0001
4	0.0006	0.0001	0.0007	0.0005	0.0002	0.0001
5	0.0006	0.0002	0.0004	0.0003	0.0004	0.0001
6	0.0005	0.0001	0.0004	0.0003	0.0002	0.0001
7	0.0009	0.0001	0.0005	0.0001	0.0005	0.0001
8	0.0007	0.0001	0.0004	0.0004	0.0001	0.0001
9	0.0008	0.0001	0.0004	0.0003	0.0002	0.0001
10	0.0007	0.0001	0.0004	0.0003	0.0002	0.0001
Average	**0.00064**	**0.00011**	**0.00047**	**0.0003**	**0.00027**	**0.0001**

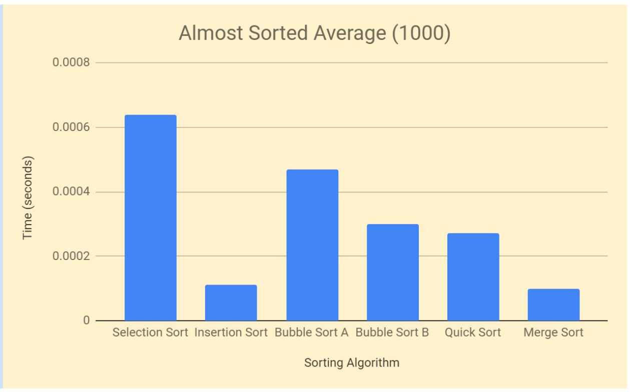
Almost Sorted Times (10000) (seconds)

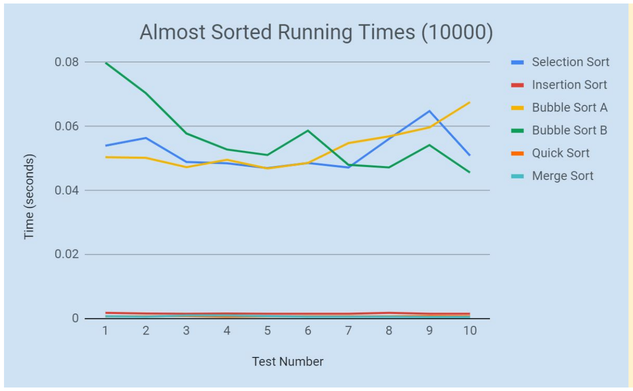
Test Number	Selection Sort	Insertion Sort	Bubble Sort A	Bubble Sort B	Quick Sort	Merge Sort
1	0.054	0.0018	0.0504	0.0799	0.0007	0.0007
2	0.0564	0.0016	0.0502	0.0704	0.0007	0.0006
3	0.0489	0.0015	0.0473	0.0578	0.0008	0.001
4	0.0485	0.0016	0.0496	0.0528	0.0006	0.001
5	0.047	0.0015	0.0469	0.0511	0.0007	0.0008
6	0.0486	0.0015	0.0486	0.0587	0.0007	0.0006
7	0.0472	0.0015	0.0548	0.048	0.0007	0.0006
8	0.0561	0.0018	0.0569	0.0472	0.0007	0.0006
9	0.0648	0.0015	0.0597	0.0542	0.0008	0.0005
10	0.0509	0.0015	0.0676	0.0456	0.0007	0.0005
Average	**0.05224**	**0.00158**	**0.0532**	**0.05657**	**0.00071**	**0.00069**

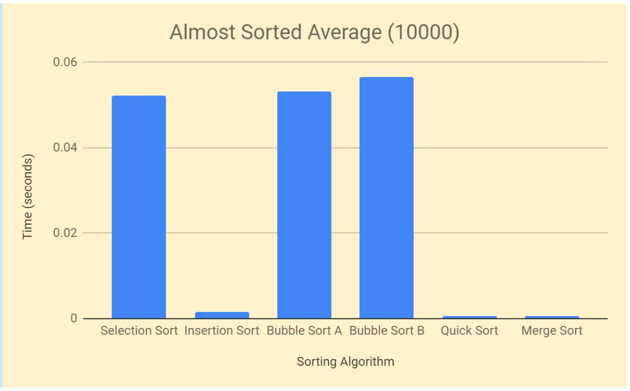
Almost Sorted Times (100000) (seconds)

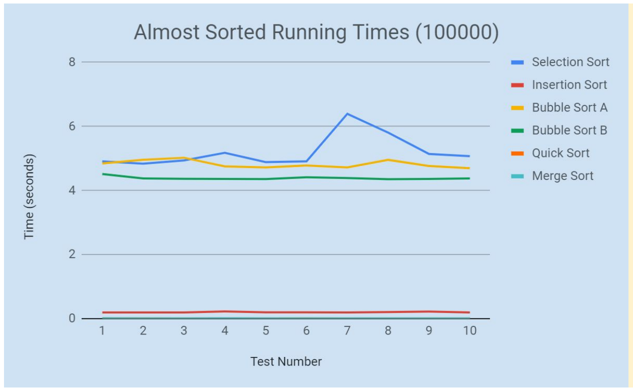
Test Number	**Selection Sort**	**Insertion Sort**	**Bubble Sort A**	**Bubble Sort B**	**Quick Sort**	**Merge Sort**
1	4.911	0.1921	4.8447	4.5137	0.0137	0.0093
2	4.8402	0.1934	4.9612	4.3791	0.0089	0.0081
3	4.9389	0.1916	5.0213	4.367	0.0066	0.0084
4	5.1757	0.224	4.7552	4.362	0.0118	0.0088
5	4.8876	0.1958	4.7218	4.3594	0.008	0.0066
6	4.9105	0.1972	4.7811	4.4153	0.012	0.0073
7	6.3958	0.1925	4.7219	4.3885	0.0068	0.0073
8	5.8087	0.2056	4.9569	4.3521	0.0069	0.0067
9	5.1393	0.2191	4.7638	4.3619	0.0075	0.0069
10	5.0715	0.1936	4.6982	4.3776	0.0067	0.0066
Average	**5.20792**	**0.20049**	**4.82261**	**4.38766**	**0.00889**	**0.0076**

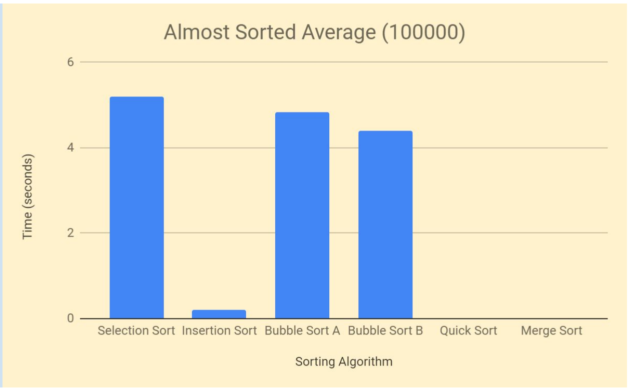












- Clearly, selection sort, and bubble sort (with and without swaps) are taking the longest to finish.
- The algorithms quick sort and merge still appear to perform at the same rate and are the fastest among the others.

Random Averages (seconds)

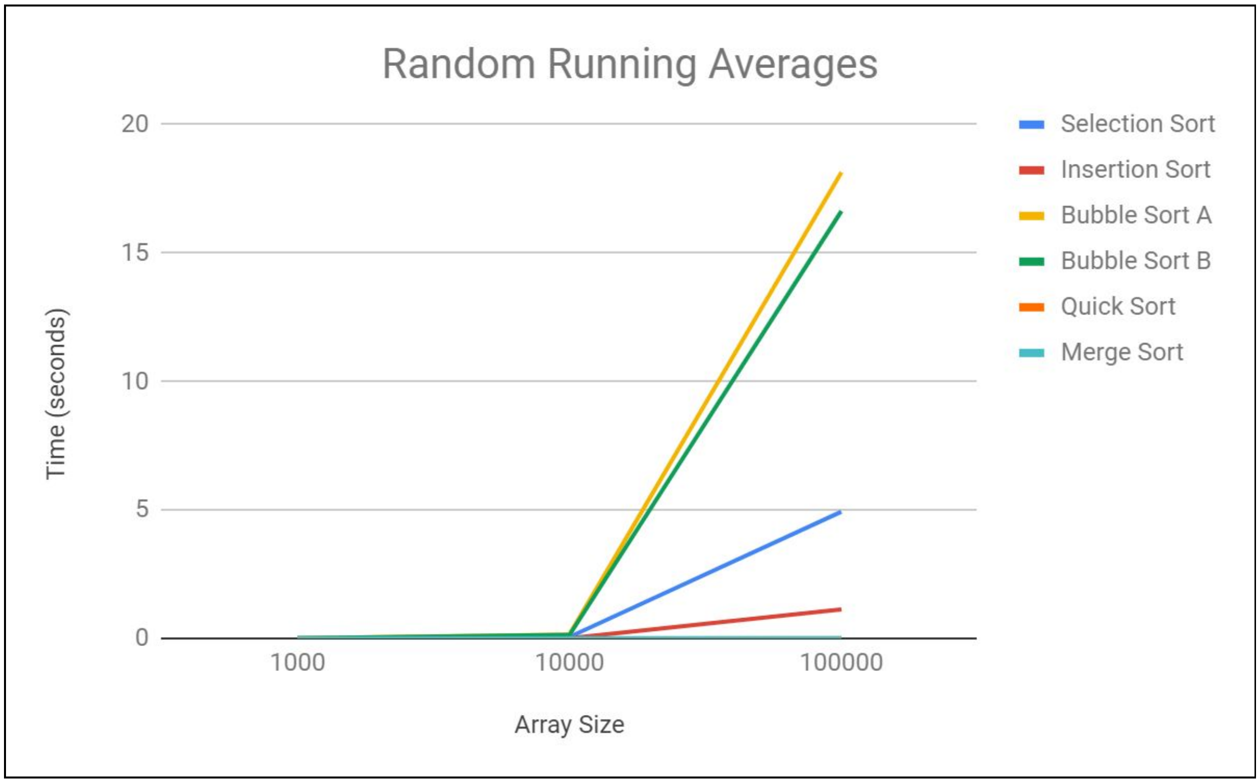
Array Size	Selection Sort	Insertion Sort	Bubble Sort A	Bubble Sort B	Quick Sort	Merge Sort
1000	0.0017	0.0015	0.0029	0.0026	0.0002	0.0004
10000	0.0521	0.0109	0.1449	0.1374	0.0007	0.0012
100000	4.932	1.1246	18.1444	16.6323	0.0084	0.0104

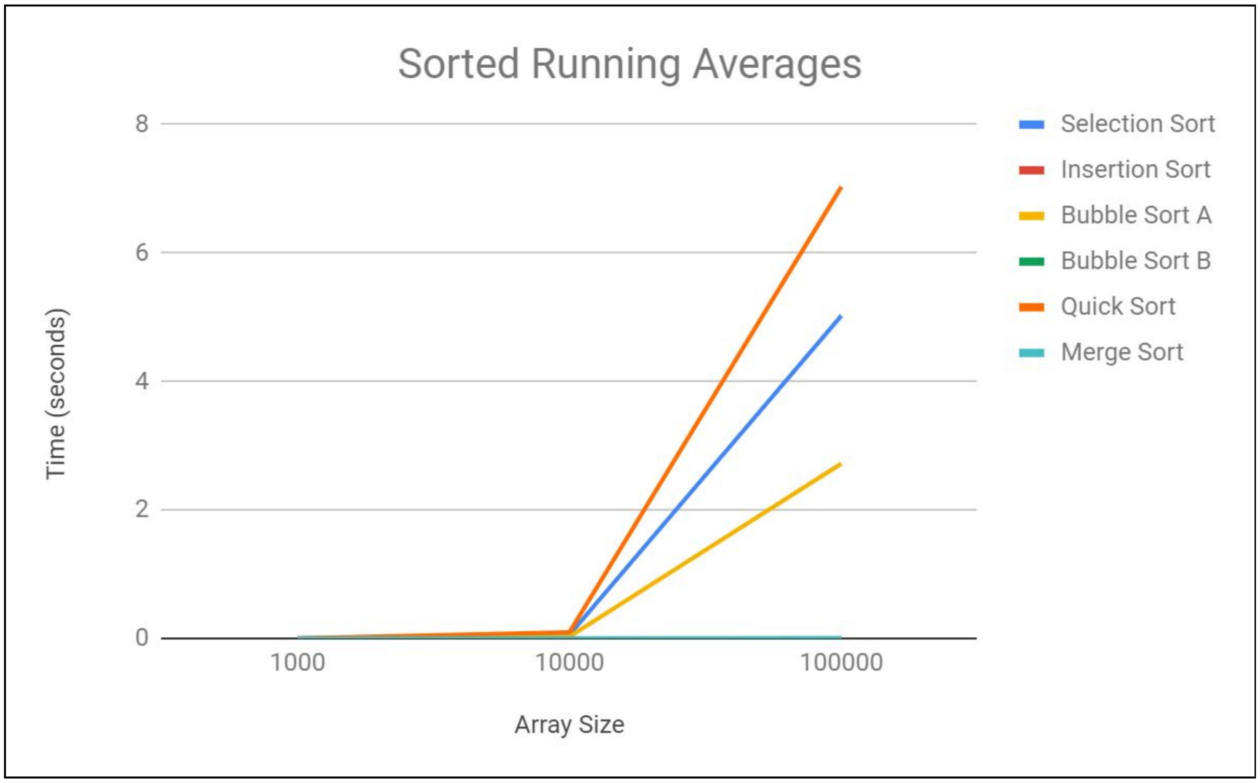
Sorted Averages (seconds)

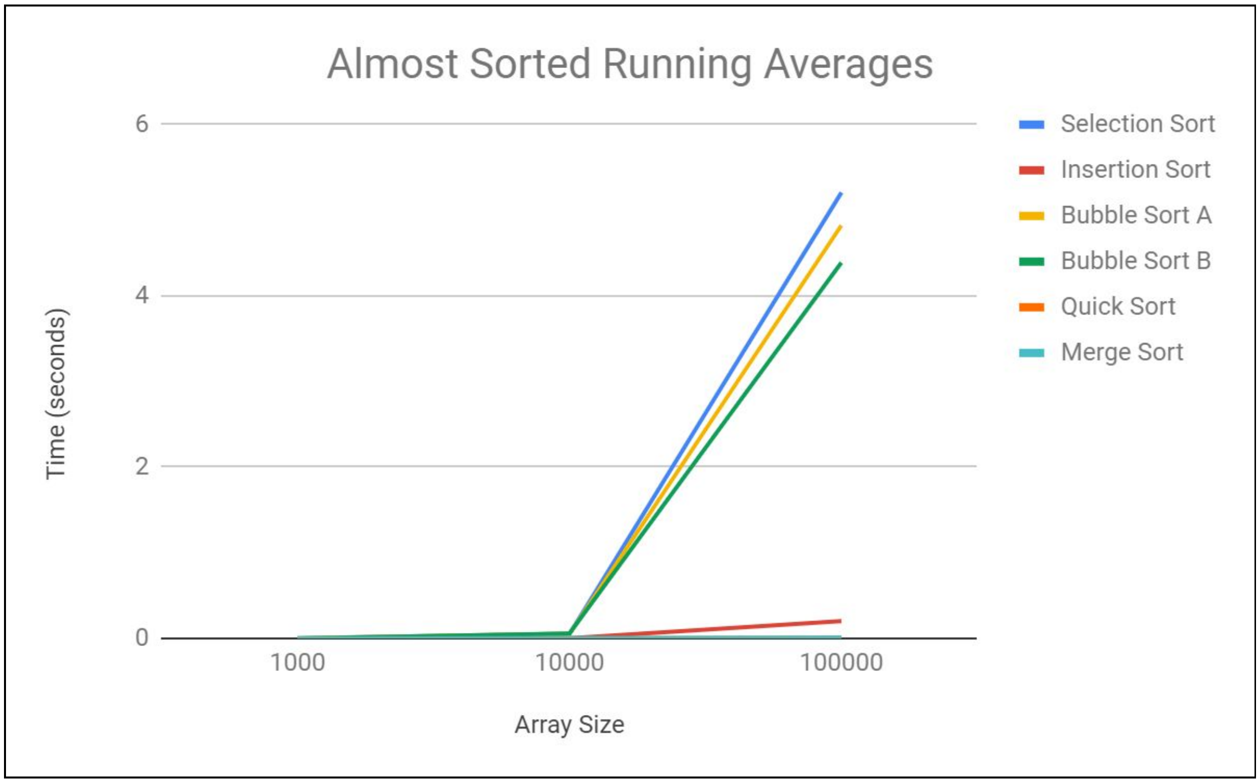
Array Size	Selection Sort	Insertion Sort	Bubble Sort A	Bubble Sort B	Quick Sort	Merge Sort
1000	0.0006	0	0.0003	0	0.0009	0.0001
10000	0.0531	0	0.0274	0	0.0946	0.0009
100000	5.026	0.0002	2.7214	0.0001	7.0332	0.0073

Almost Sorted Averages (seconds)

Array Size	Selection Sort	Insertion Sort	Bubble Sort A	Bubble Sort B	Quick Sort	Merge Sort
1000	0.0006	0	0.0005	0.0003	0.0003	0.0001
10000	0.0522	0.0016	0.0532	0.0566	0.0007	0.0007
100000	5.2079	0.2005	4.8226	4.3877	0.0089	0.0076


Random Running Averages
20
15
10
5
0
Time (seconds)
1000
10000
100000
Array Size
Selection Sort
Insertion Sort
Bubble Sort A
Bubble Sort B
Quick Sort
Merge Sort


Sorted Running Averages
8
6
4
2
0
Time (seconds)
1000
10000
100000
Array Size
Selection Sort
Insertion Sort
Bubble Sort A
Bubble Sort B
Quick Sort
Merge Sort


Almost Sorted Running Averages
Time (seconds)
6
4
2
0
1000
10000
100000
Array Size
Selection Sort
Insertion Sort
Bubble Sort A
Bubble Sort B
Quick Sort
Merge Sort

Conclusion

Time complexity is not the only consideration when designing and evaluating algorithms. The algorithms quick sort and merge sort are both the fastest among the others, but merge sort has Θ(n) space complexity, used to merge two subarrays. A stack overflow error could occur using the quick sort or merge sort algorithms. While testing the quick sort and merge sort algorithms, I began to receive stack overflow errors on inputs of size 100000, so I then allocated 8 megabytes to my stack before running the tests, because my default stack size was 320 kilobytes.

Random Sorted Arrays

- Sorting algorithms selection sort, insertion sort, and bubble sort (with and without swaps) clearly appear to have quadratic growth.
- Sorting algorithms quick sort and merge sort appear to have logarithm growth.

Sorted Arrays

- Sorting algorithms selection sort, bubble sort (with swaps), quick sort, and merge sort clearly appear to have quadratic growth.
- Sorting algorithms insertion sort and bubble sort (without swaps) appear to have logarithm growth.

Almost Sorted Arrays

- Sorting algorithms selection sort, insertion sort, and bubble sort (with and without swaps) clearly appear to have quadratic growth.
- Sorting algorithms quick sort and merge sort appear to have logarithm growth.

In conclusion, the experimental results appear to agree with the theoretical information.