Sorting Algorithms

Selection Sort | Insertion Sort | Bubble Sort | Quick Sort | Merge Sort

Introduction

A sorting algorithm is a method for reorganizing a large number of items (i.e. number, object, name, etc...) into a specific order, such as alphabetical, highest-to-lowest value or shortest-to-longest distance. Sorting algorithms take lists of items as input data, perform specific operations on those lists and deliver ordered arrays as output. The many applications of sorting algorithms include organizing items by price on a retail website and determining the order of sites on a search engine results page.

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 sub-arrays in a given array.
  - The sub-array which is already sorted.
  - Remaining sub-array 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

- Quick Sort 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 Quick Sort that pick pivot in different ways.
  - Always pick first element as pivot.
  - Always pick last element as pivot
  - Pick a random element as pivot.
  - 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 of Sorting Algorithms

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 w/o swap	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
Bubble Sort w/ swap	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Quick Sort	$\Omega(n \cdot \log(n))$	$\Theta(n \cdot \log(n))$	$O(n^2)$
Merge Sort	$\Omega(n \cdot \log(n))$	$\Theta(n \cdot \log(n))$	$O(n \cdot \log(n))$

Algorithms Attributes

Algorithm	Non-Adaptive	In-Place	Stable
Selection Sort	Yes	Yes	No
Insertion Sort	No	Yes	Yes
Bubble Sort w/o swap	Yes	Yes	Yes
Bubble Sort w/ swap	No	Yes	Yes
Quick Sort	No	Yes	No
Merge Sort	Yes	No	Yes

Objective

Run the sorting algorithms for the following types of arrays containing:

- 1000 integers: random numbers, sorted list, almost sorted list.
- 10000 integers: random numbers, sorted list, almost sorted list.
- 100000 integers: random numbers, sorted list, almost sorted list.
- Seed that was use to generate random number: 363366622308613L

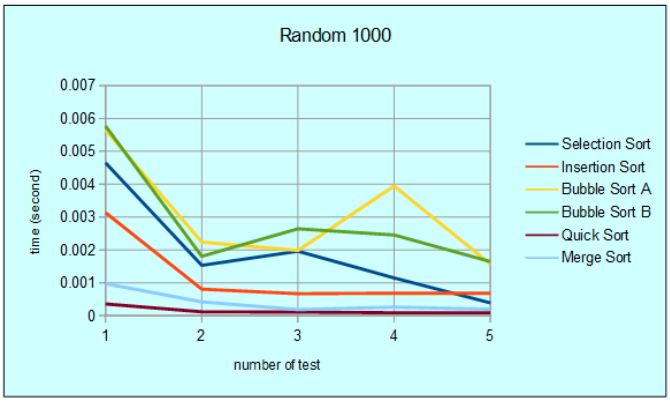
Use the experimental results to determine if it support the theoretical information about the efficiency of each sorting algorithms. In additional, conclude which sorting algorithms work better for specific types of input.

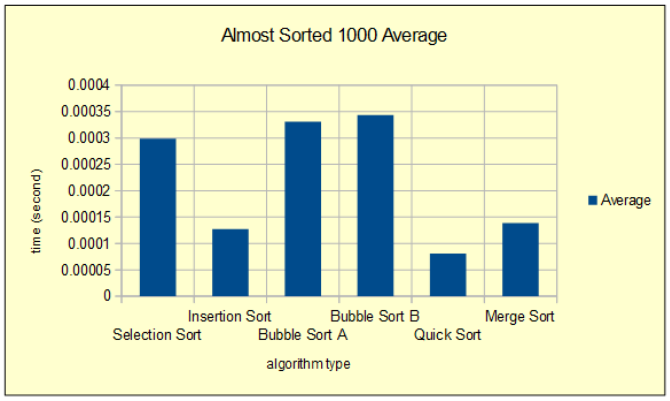
Basic Computer Information (that was use to run sorting algorithms)

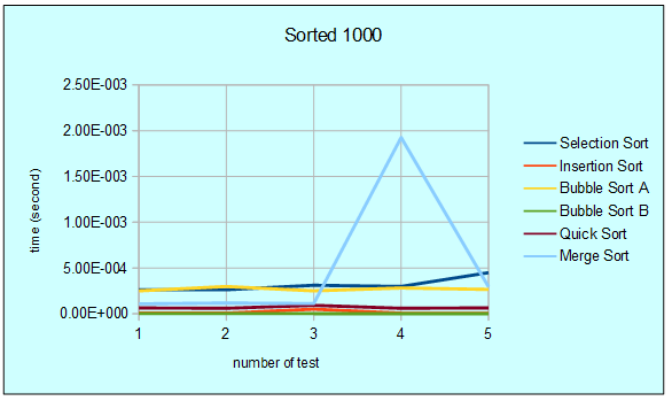
CPU: 2.5 GHz Quad-core Intel Core i5-7300HQ
GPU: NVDIA GeForce GTX 1050 Ti
RAM: 16GB DDR4
Storage: 128GB SSD and 1TB HDD

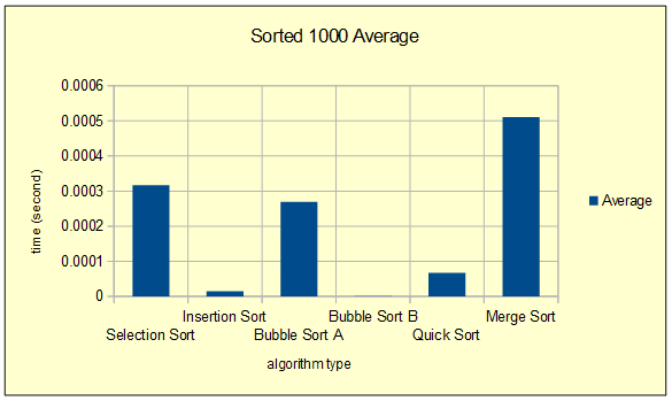
Experimental Data

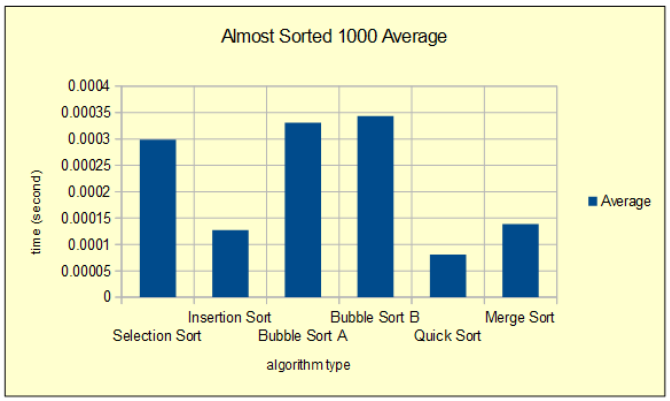
Size: 1000					
--Random--	**--Random--**	**--Random--**	**--Random--**	**--Random--**	**--Random--**
0.004648619	0.003131079	0.005648825	0.005767389	3.56E-004	9.73E-004
0.001529437	8.06E-004	0.00223713	0.001804309	1.17E-004	4.19E-004
1.96E-003	6.67E-004	0.00198195	0.002641643	1.07E-004	1.86E-004
0.001147078	6.84E-004	0.003943388	0.002453335	8.78E-005	2.59E-004
3.90E-004	6.80E-004	0.001570874	0.00164472	9.07E-005	1.96E-004
0.001935017	**0.0011936012**	**0.0030764334**	**0.0028622792**	**1.52E-004**	**4.07E-004**
--Sorted--	**--Sorted--**	**--Sorted--**	**--Sorted--**	**--Sorted--**	**--Sorted--**
2.64E-004	6.15E-006	2.51E-004	1.23E-006	6.36E-005	1.08E-004
2.63E-004	5.74E-006	2.98E-004	1.23E-006	5.91E-005	1.17E-004
3.11E-004	5.05E-005	2.50E-004	1.23E-006	9.19E-005	1.11E-004
2.98E-004	5.74E-006	2.81E-004	8.21E-007	5.91E-005	0.001926155
4.49E-004	5.33E-006	2.66E-004	8.21E-007	6.40E-005	2.94E-004
3.17E-004	**1.47E-005**	**2.69E-004**	**1.07E-006**	**6.75E-005**	**5.11E-004**
--Almost Sorted--	**--Almost Sorted--**	**--Almost Sorted--**	**--Almost Sorted--**	**--Almost Sorted--**	**--Almost Sorted--**
3.01E-004	1.19E-004	3.15E-004	3.11E-004	9.27E-005	1.55E-004
2.99E-004	1.29E-004	3.77E-004	3.72E-004	7.88E-005	1.54E-004
2.98E-004	1.30E-004	3.42E-004	3.41E-004	7.92E-005	1.38E-004
2.97E-004	1.32E-004	3.11E-004	3.21E-004	7.67E-005	1.23E-004
2.98E-004	1.26E-004	3.08E-004	3.71E-004	7.63E-005	1.23E-004
2.98E-004	**1.27E-004**	**3.31E-004**	**3.43E-004**	**8.07E-005**	**1.39E-004**
***bold is the average time in seconds**					







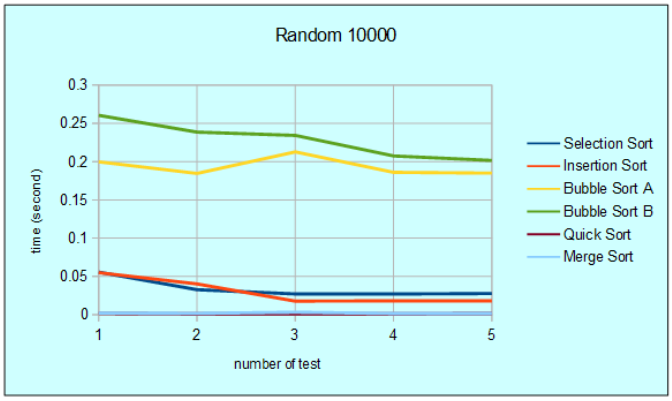


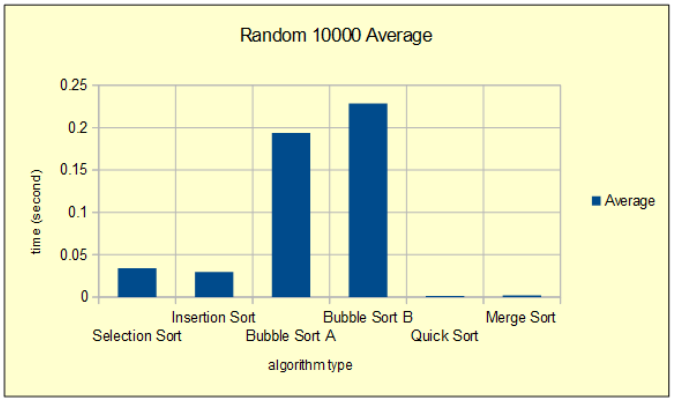


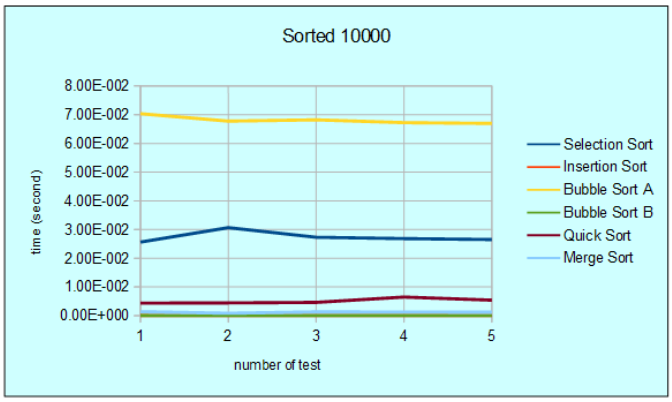
At array size 1000

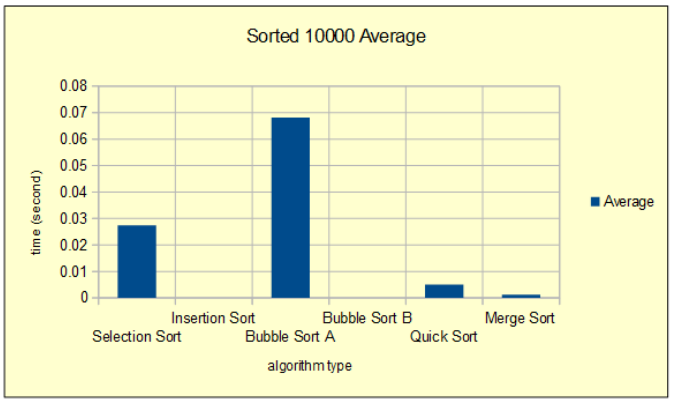
→ All the sorting algorithm are performing the same rate (they all below 1 second).

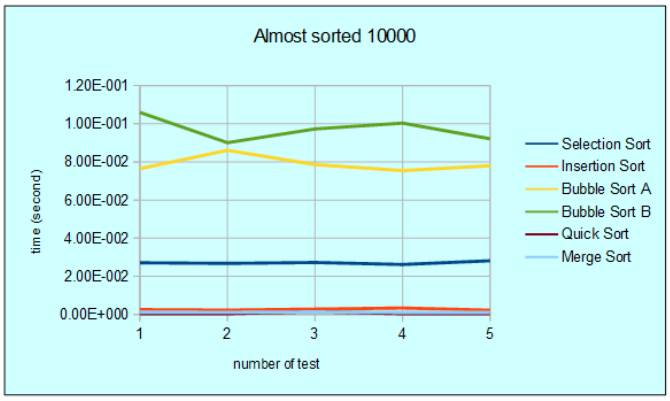
Size: 10000					
Selection Sort	**Insertion Sort**	**Bubble Sort A**	**Bubble Sort B**	**Quick Sort**	**Merge Sort**
--Random--	**--Random--**	**--Random--**	**--Random--**	**--Random--**	**--Random--**
0.055576661	0.054916558	0.199742112	0.260488829	1.68E-003	2.10E-003
0.032621976	4.02E-002	0.184601177	0.238584401	1.11E-003	1.64E-003
2.71E-002	1.76E-002	0.212774329	0.234243885	1.07E-003	3.21E-003
0.02694156	1.82E-002	0.186025999	0.207412683	1.04E-003	1.50E-003
2.76E-002	1.81E-002	0.18503851	0.201580473	1.58E-003	1.39E-003
0.0339755766	**0.0297967834**	**0.1936364254**	**0.2284620542**	**1.30E-003**	**1.97E-003**
--Sorted--	**--Sorted--**	**--Sorted--**	**--Sorted--**	**--Sorted--**	**--Sorted--**
2.56E-002	1.72E-005	7.04E-002	1.56E-005	4.38E-003	1.40E-003
3.07E-002	1.64E-005	6.78E-002	1.64E-005	4.46E-003	8.42E-004
2.73E-002	1.64E-005	6.82E-002	1.64E-005	4.61E-003	1.31E-003
2.68E-002	1.64E-005	6.72E-002	1.64E-005	6.49E-003	0.001258667
2.65E-002	1.68E-005	6.70E-002	1.52E-005	5.40E-003	1.22E-003
2.74E-002	**1.67E-005**	**6.81E-002**	**1.60E-005**	**5.07E-003**	**1.20E-003**
--Almost Sorted--	**--Almost Sorted--**	**--Almost Sorted--**	**--Almost Sorted--**	**--Almost Sorted--**	**--Almost Sorted--**
2.71E-002	2.55E-003	7.64E-002	1.06E-001	4.45E-004	1.26E-003
2.68E-002	2.31E-003	8.61E-002	9.01E-002	4.35E-004	1.11E-003
2.73E-002	2.84E-003	7.86E-002	9.73E-002	1.46E-003	1.37E-003
2.62E-002	3.41E-003	7.54E-002	1.00E-001	4.29E-004	1.05E-003
2.82E-002	2.28E-003	7.80E-002	9.21E-002	4.35E-004	1.07E-003
2.71E-002	**2.68E-003**	**7.89E-002**	**9.72E-002**	**6.40E-004**	**1.17E-003**
***bold is the average time in seconds**					

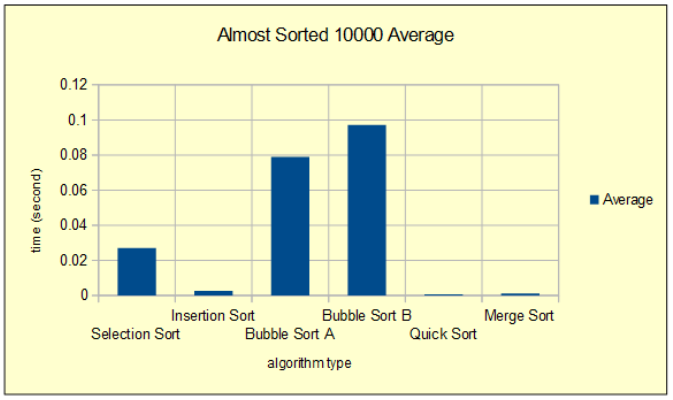










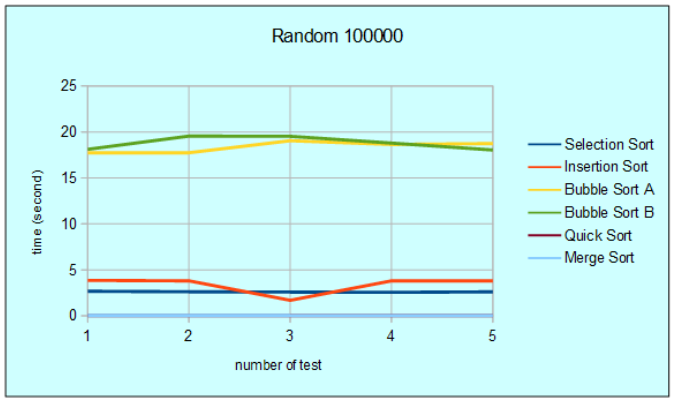


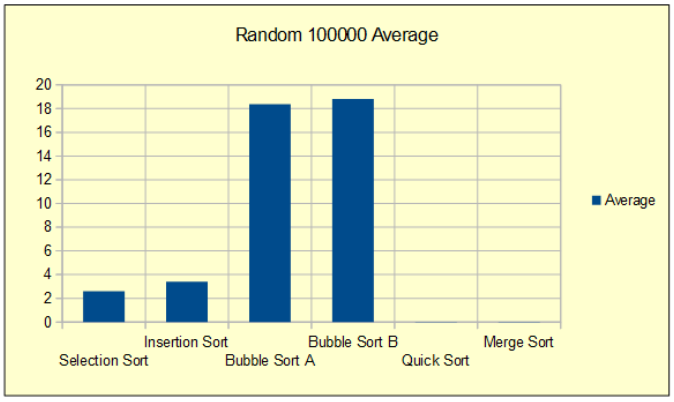
At array size 10000

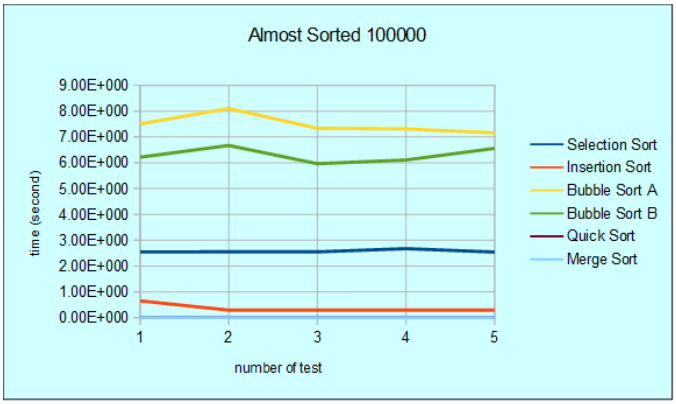
→ All algorithm still perform under 1 second.

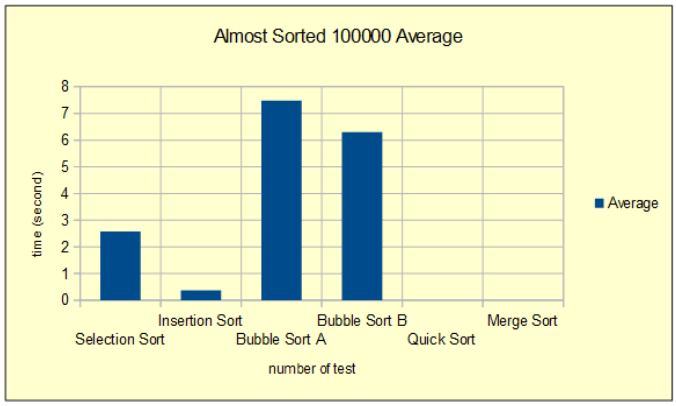
→ However, Bubble Sort starting to take more time to finish sorting the inputs.

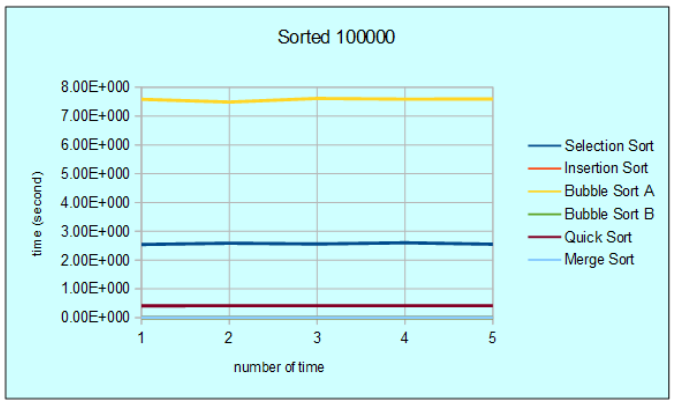
Size: 100000					
Selection Sort	**Insertion Sort**	**Bubble Sort A**	**Bubble Sort B**	**Quick Sort**	**Merge Sort**
--Random--	**--Random--**	**--Random--**	**--Random--**	**--Random--**	**--Random--**
2.66958537	3.85236993	17.722131464	18.113584504	1.41E-002	2.02E-002
2.615961942	3.80E+000	17.740748095	19.553554095	6.55E-003	2.12E-002
2.59E+000	1.68E+000	19.051023631	19.526574791	6.36E-003	2.06E-002
2.55143635	3.80E+000	18.645472529	18.78984393	6.24E-003	1.73E-002
2.60E+000	3.81E+000	18.744245944	18.038603108	6.33E-003	1.38E-002
2.6050417272	**3.3893999504**	**18.3807243326**	**18.8044320856**	**7.93E-003**	**1.86E-002**
--Sorted--	**--Sorted--**	**--Sorted--**	**--Sorted--**	**--Sorted--**	**--Sorted--**
2.54E+000	6.83E-004	7.58E+000	1.62E-004	4.11E-001	5.35E-003
2.59E+000	1.21E-003	7.49E+000	1.60E-004	4.22E-001	5.41E-003
2.56E+000	6.25E-004	7.61E+000	1.30E-004	4.15E-001	5.49E-003
2.60E+000	6.53E-004	7.59E+000	1.43E-004	4.14E-001	0.005531902
2.55E+000	8.72E-004	7.60E+000	1.44E-004	4.15E-001	5.34E-003
2.57E+000	**8.08E-004**	**7.57E+000**	**1.48E-004**	**4.15E-001**	**5.42E-003**
--Almost Sorted--	**--Almost Sorted--**	**--Almost Sorted--**	**--Almost Sorted--**	**--Almost Sorted--**	**--Almost Sorted--**
2.54E+000	6.54E-001	7.51E+000	6.22E+000	4.43E-003	6.25E-003
2.56E+000	2.94E-001	8.10E+000	6.67E+000	3.71E-003	6.27E-003
2.55E+000	2.97E-001	7.34E+000	5.97E+000	4.27E-003	6.33E-003
2.68E+000	2.93E-001	7.31E+000	6.11E+000	3.91E-003	6.25E-003
2.54E+000	2.95E-001	7.16E+000	6.56E+000	3.77E-003	6.53E-003
2.57E+000	**3.67E-001**	**7.48E+000**	**6.31E+000**	**4.02E-003**	**6.33E-003**
***bold is the average time in seconds**					

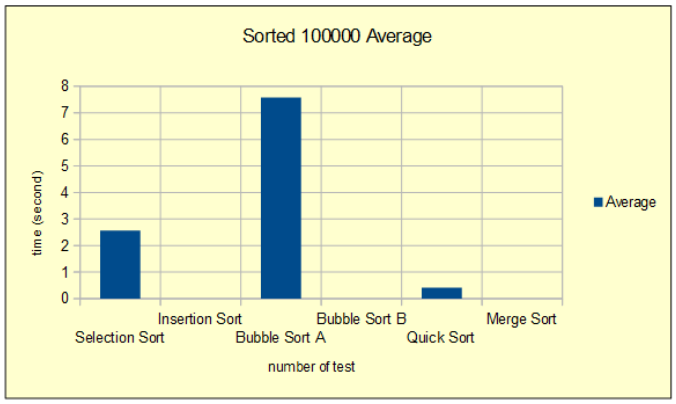












At array size 100000

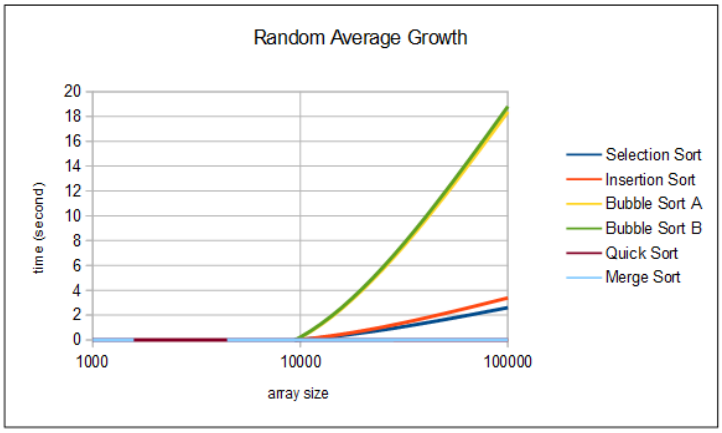
→ Bubble Sort (with and without swap counting) took the longest to finish sorting the inputs.
→ Selection Sort starting to take longer to finish sorting when working with all array types.
→ Insertion Sort starting to take longer to finish sorting when working with random array.
→ Lastly, Quick Sort and Merge Sort still remain under 1 second.

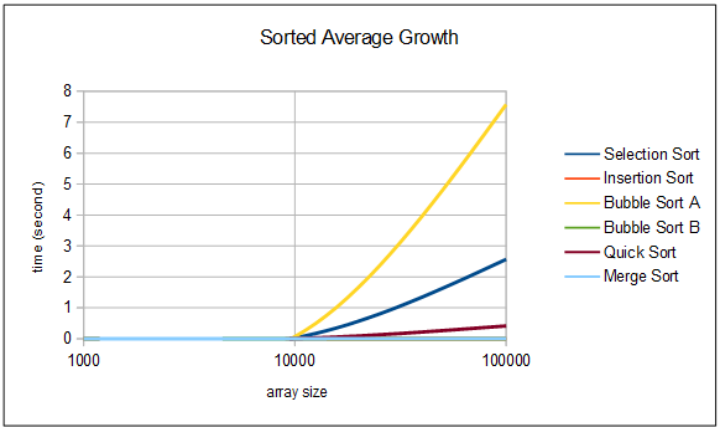
Average Time

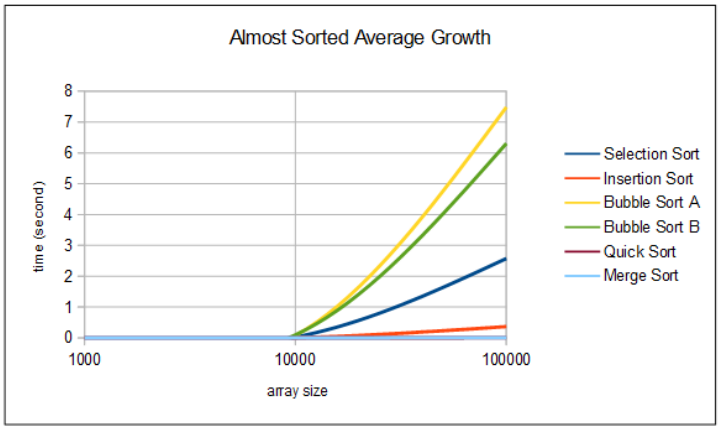
Random						
Array Size	**Selection Sort**	**Insertion Sort**	**Bubble Sort A**	**Bubble Sort B**	**Quick Sort**	**Merge Sort**
1000	0.001935017	0.0011936012	0.0030764334	0.0028622792	0.0001518772	0.0004066462
10000	0.0339755766	0.0297967834	0.1936364254	0.2284620542	0.0012954268	0.0019671812
100000	2.6050417272	3.3893999504	18.3807243326	18.8044320856	0.0079275552	0.0185971844

Sorted						
Array Size	**Selection Sort**	**Insertion Sort**	**Bubble Sort A**	**Bubble Sort B**	**Quick Sort**	**Merge Sort**
1000	0.0003170466	0.000014687	0.0002692106	0.000001067	0.000067528	0.0005113442
10000	0.0273904634	1.66568E-005	0.0681134506	0.000016	0.0050667528	0.0012038576
100000	2.5681772866	0.0008077954	7.5732044088	0.0001478562	0.415325387	0.005423184

Almost Sorted						
Array Size	**Selection Sort**	**Insertion Sort**	**Bubble Sort A**	**Bubble Sort B**	**Quick Sort**	**Merge Sort**
1000	0.0002984206	0.0001270154	0.0003305846	0.000343139	8.07388E-005	0.0001386668
10000	0.0271133762	0.0026784022	0.0788936854	0.0971521618	0.0006399184	0.0011739908
100000	2.5746294764	0.3666663114	7.4837548996	6.305448784	0.0040190394	0.0063264872







Conclusion

Time Complexities of Sorting Algorithms

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 w/o swap	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
Bubble Sort w/ swap	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Quick Sort	$\Omega(n \cdot \log(n))$	$\Theta(n \cdot \log(n))$	$O(n^2)$
Merge Sort	$\Omega(n \cdot \log(n))$	$\Theta(n \cdot \log(n))$	$O(n \cdot \log(n))$

Base on the average time graph (above):

- Selection Sort have:
  - A quadratic growth when working with random, sorted, and almost sorted arrays.
- Insertion Sort have:
  - A quadratic growth when working with random and almost sorted arrays.
- Bubble Sort without swap counting have:
  - A quadratic growth when working with random, sorted, and almost sorted arrays.
- Bubble Sort with swap counting have:
  - A quadratic growth when working with random and almost sorted arrays
  - A linear growth when working with sorted arrays.
- Quick Sort have:
  - A logarithmic growth when working with random and almost sorted arrays.
  - A quadratic growth when working with sorted arrays.
    - Although the sorting time is below 1 second, as array size increasing to million or billion, we will start seeing the time difference (assuming the computer have enough memory).
    - To prove this, I sorted 1,000,000 inputs using Quick Sort.

Quick Sort			
Number of Test	**Random**	**Sorted**	**Almost Sorted**
1	0.093135256	4.424997682	0.08921115
2	0.084679044	4.249132512	0.08714879
3	0.085796583	4.23172778	0.082846017
4	0.084674942	4.237943991	0.084024274
5	0.085320276	4.258103596	0.085327659
Average	**0.0867212202**	**4.2803811122**	**0.085711578**
			***bold is the average time in seconds**

- Merge Sort have:
  - A logarithmic growth when working with random, sorted, and almost sorted arrays.
  - Why Quick Sort is preferred over Merge Sort for sorting Arrays?
    - Quick Sort in its general form is an in-place sort (it doesn't require any extra storage). Whereas merge sort requires O(N) extra storage.
    - Allocating and de-allocating the extra space used for merge sort increases the running time of the algorithm. For arrays, merge sort loses due to the use of extra O(N) storage space...

In conclusion, the experimental data agreed with the theoretical information.