Name: Brandon London
Course: CS 3130
Professor: Galina Piatnitskaia
Date: October 26, 2020

Analysis of Different Quicksort Implementations

Definition

Time Complexities

Algorithm	Best	Average	Worst
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Quick Sort	$\Omega(n \cdot log(n))$	$\Theta(n \cdot log(n))$	$O(n^2)$

Attributes

Algorithm	Non-Adaptive	In-Place	Stable
Insertion Sort	No	Yes	Yes
Quick Sort	No	Yes	No

Insertion Sort

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

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. (implemented below)
4. Pick median as pivot. (implemented below)
5. Can be combine with Insertion. (implemented below)

Quick Sort Median of Three

The point of using the median-of-3 method is to bring us closer to the average run-time of quick sort: O(nlogn). This is because by avoiding the farthest elements in the list to choose our partition, we can minimize our deviation from the best/average case run-time of O(nlogn). Using the median-of-3 method will allow us to never have to search the entire list for the right partition but rather start somewhere in the middle. Therefore, we can avoid the worst-case run-time of O(n^2) and settle at the average run-time of O(nlogn).

Quick Sort With Insertion Sort

Insertion sort is much faster (O(n) vs O(n log n)) than quicksort IF the data set is already sorted, or nearly so; second, if the data set is very small, the 'start up time" to set up the quicksort, find a pivot point and so on, dominates the rest

Quick Sort Random Pivot

Randomized Quick Sort works well even when the array is sorted/reversely sorted and the complexity is more towards O(n log n). (Yet, there is still a possibility that the randomly picked element is always an extreme.)

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.

Experimental Data

Random Times (1000) (seconds)

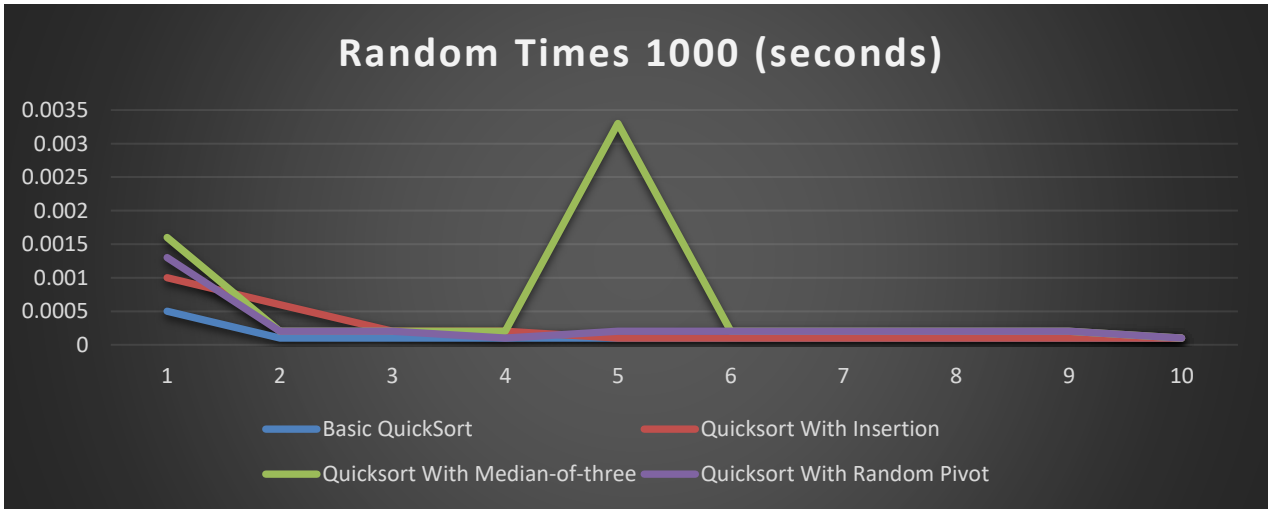
Random Times (1000) (seconds)				
Test Number	**Basic QuickSort**	**Quicksort With Insertion**	**Quicksort With Median-of-three**	**Quicksort With Random Pivot**
1	0.0005	0.001	0.0016	0.0013
2	0.0001	0.0006	0.0002	0.0002
3	0.0001	0.0002	0.0002	0.0002
4	0.0001	0.0002	0.0002	0.0001
5	0.0001	0.0001	0.0033	0.0002
6	0.0001	0.0001	0.0002	0.0002
7	0.0001	0.0001	0.0002	0.0002
8	0.0001	0.0001	0.0002	0.0002
9	0.0001	0.0001	0.0002	0.0002
10	0.0001	0.0001	0.0001	0.0001
Average	**0.00014**	**0.00026**	**0.00064**	**0.00029**

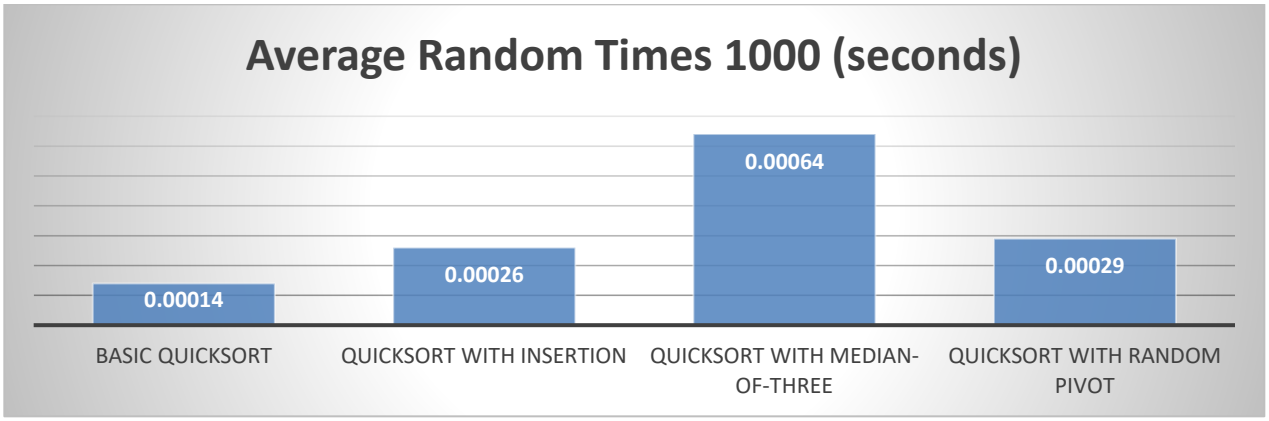
Random Times (10000) (seconds)

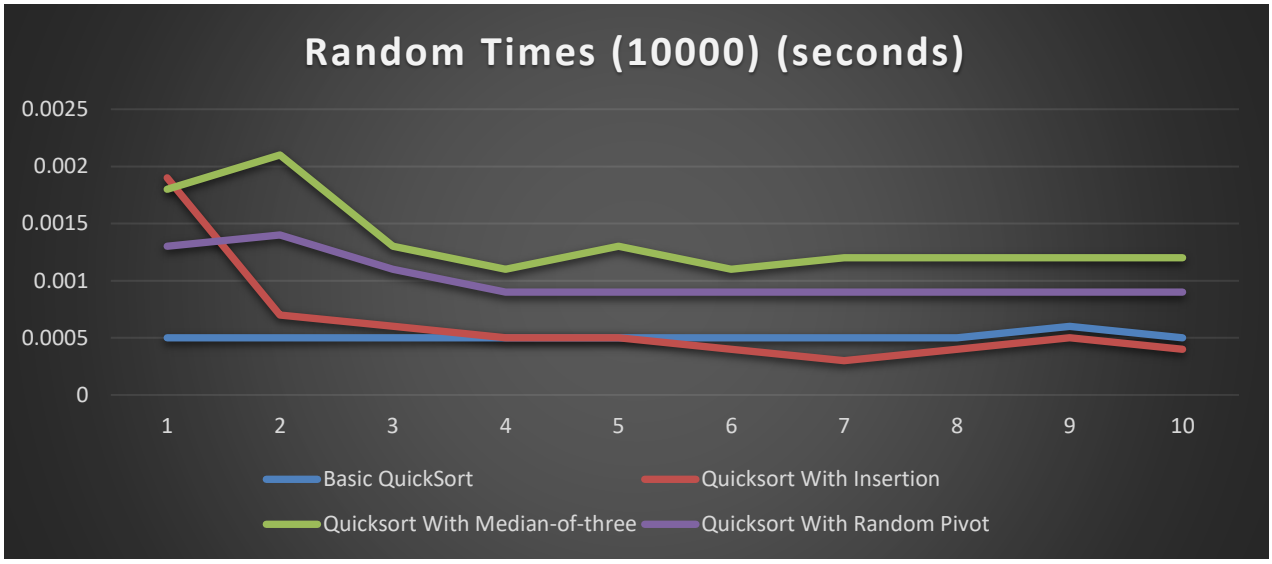
Test Number	Basic QuickSort	Quicksort With Insertion	Quicksort With Median-of-three	Quicksort With Random Pivot
1	0.0005	0.0019	0.0018	0.0013
2	0.0005	0.0007	0.0021	0.0014
3	0.0005	0.0006	0.0013	0.0011
4	0.0005	0.0005	0.0011	0.0009
5	0.0005	0.0005	0.0013	0.0009
6	0.0005	0.0004	0.0011	0.0009
7	0.0005	0.0003	0.0012	0.0009
8	0.0005	0.0004	0.0012	0.0009
9	0.0006	0.0005	0.0012	0.0009
10	0.0005	0.0004	0.0012	0.0009
Average	**0.00051**	**0.00062**	**0.00135**	**0.00101**

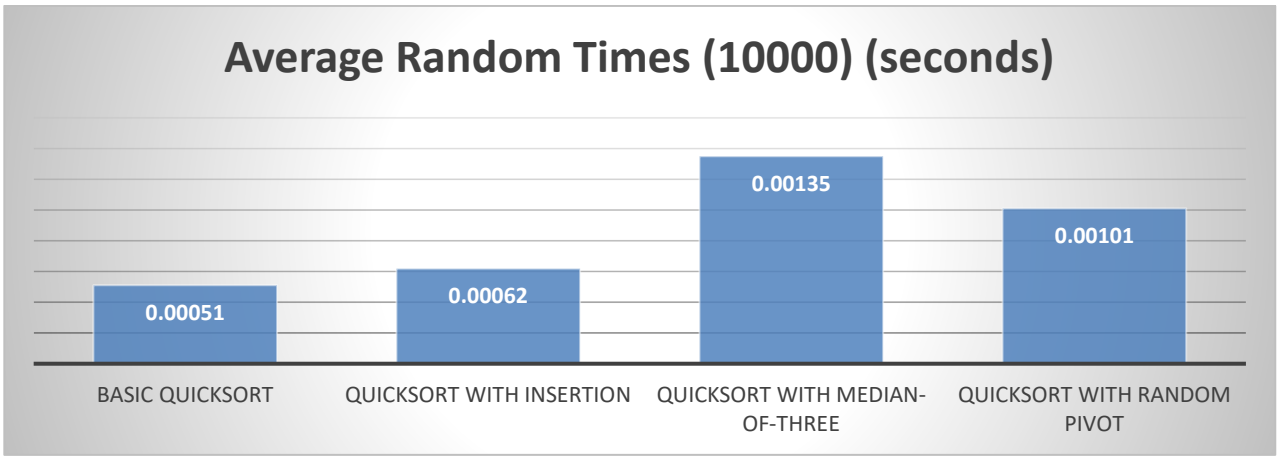
Random Times (100000) (seconds)

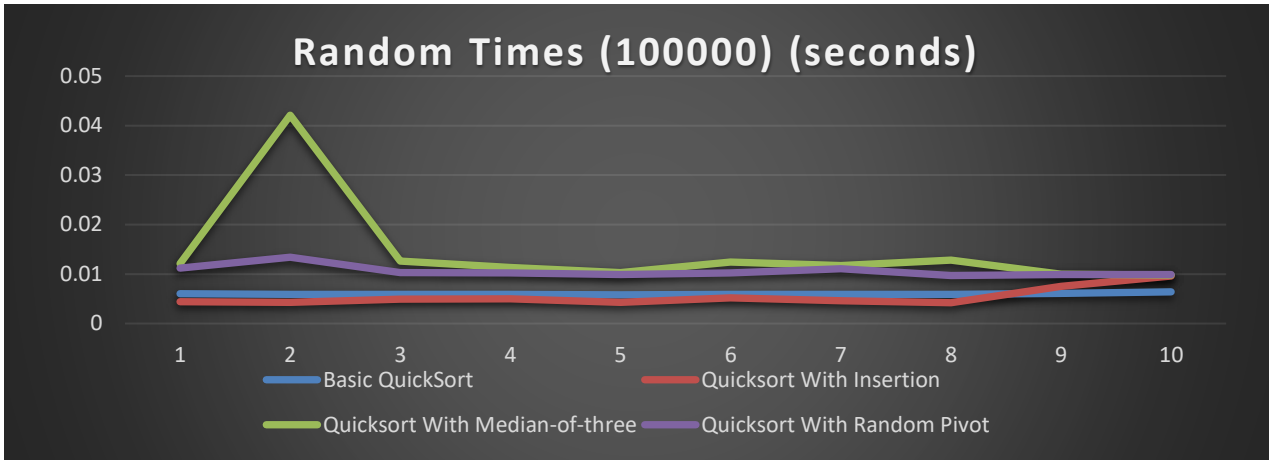
Test Number	Basic QuickSort	Quicksort With Insertion	Quicksort With Median-of-three	Quicksort With Random Pivot
1	0.006	0.0044	0.0121	0.0112
2	0.0059	0.0043	0.0421	0.0134
3	0.0059	0.0049	0.0126	0.0103
4	0.0059	0.005	0.0113	0.0102
5	0.0058	0.0043	0.0103	0.0099
6	0.0059	0.0052	0.0124	0.0102
7	0.0059	0.0046	0.0117	0.0111
8	0.0059	0.0042	0.0128	0.0097
9	0.0061	0.0075	0.01	0.0099
10	0.0064	0.0096	0.0098	0.0099
Average	**0.00597**	**0.0054**	**0.01451**	**0.01058**

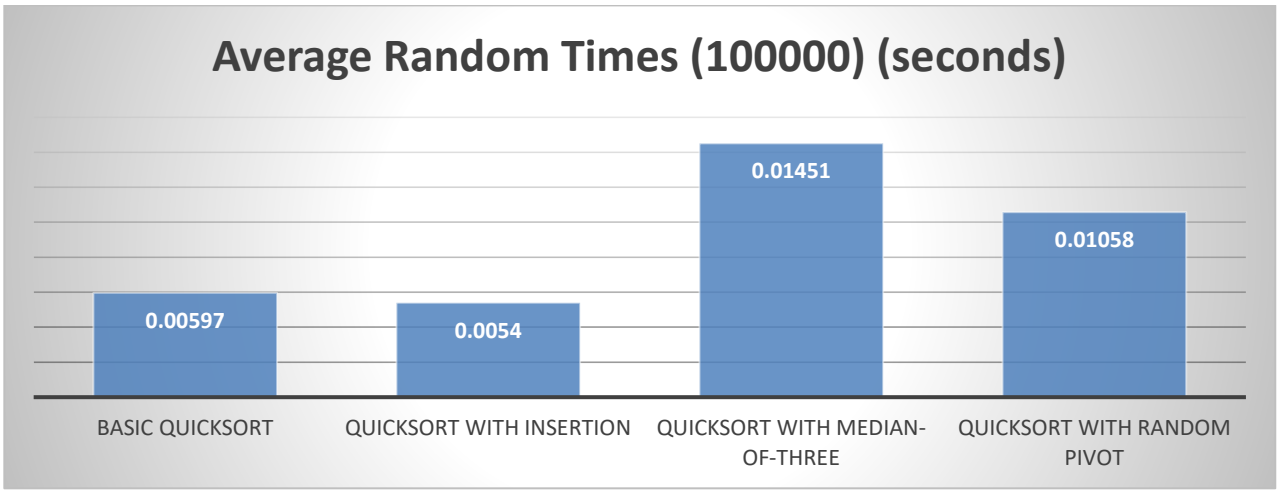



Average Random Times 1000 (seconds)
0.00014
0.00026
0.00064
0.00029
BASIC QUICKSORT
QUICKSORT WITH INSERTION
QUICKSORT WITH MEDIAN-OF-THREE
QUICKSORT WITH RANDOM PIVOT


Random Times (10000) (seconds)
0.0025
0.002
0.0015
0.001
0.0005
0
1
2
3
4
5
6
7
8
9
10
Basic QuickSort
Quicksort With Insertion
Quicksort With Median-of-three
Quicksort With Random Pivot


Average Random Times (10000) (seconds)
0.00051
0.00062
0.00135
0.00101
BASIC QUICKSORT
QUICKSORT WITH INSERTION
QUICKSORT WITH MEDIAN-OF-THREE
QUICKSORT WITH RANDOM PIVOT



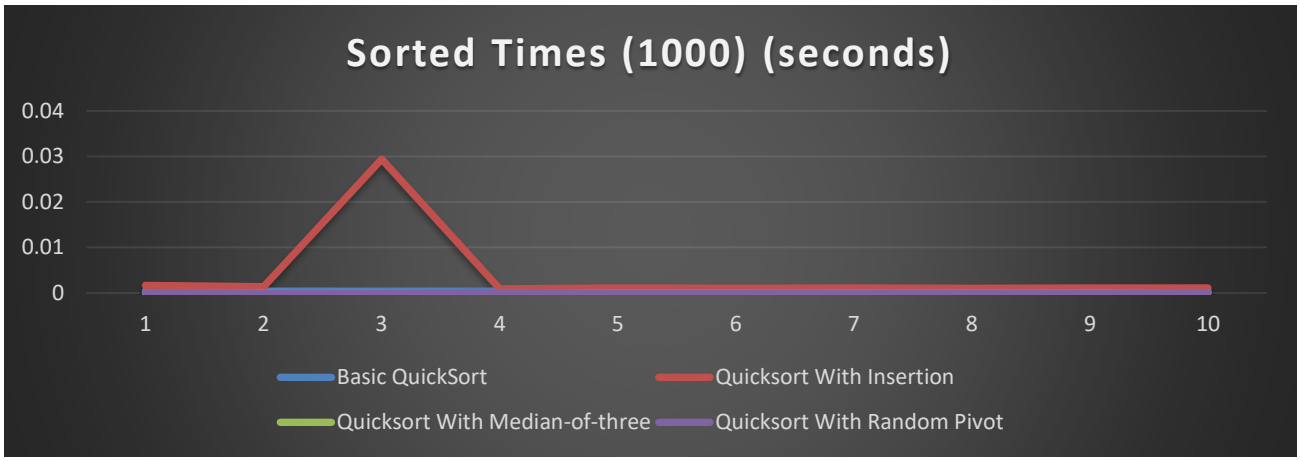


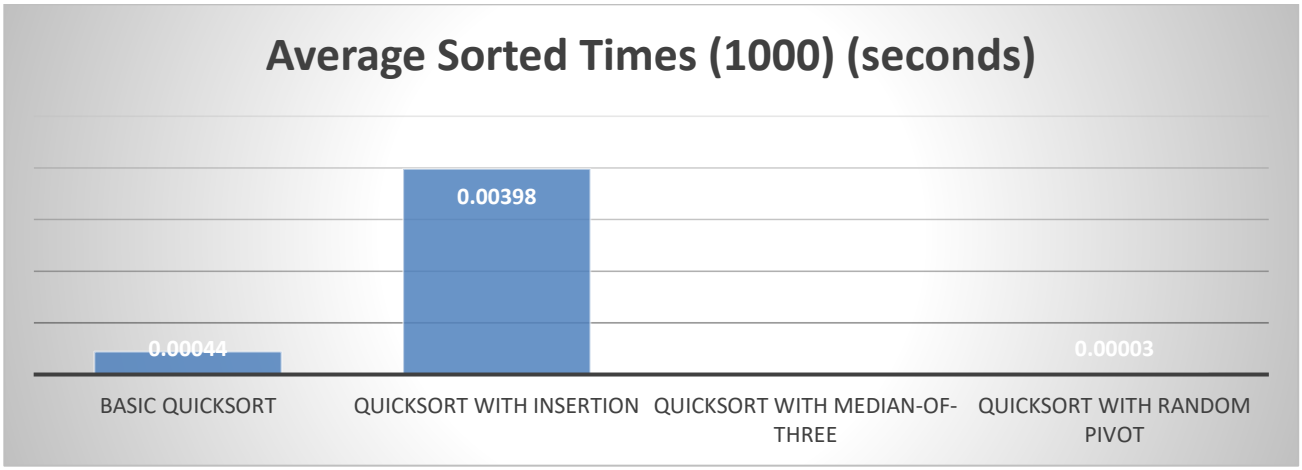
- The basic Quicksort and Quicksort with insertion have very similar but seem the smaller the array the quicker the insertion works.
- It seems the Median of Three is twice as slow

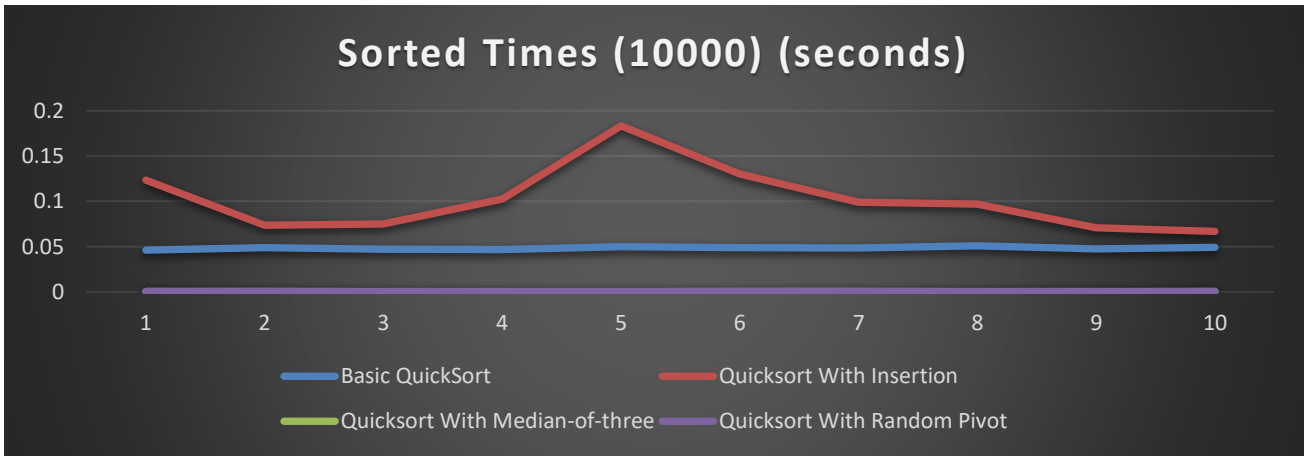
Sorted Times (1000) (seconds)				
Test Number	**Basic QuickSort**	**Quicksort With Insertion**	**Quicksort With Median-of-three**	**Quicksort With Random Pivot**
1	0.0005	0.0017	0	0.0001
2	0.0004	0.0014	0	0
3	0.0004	0.0294	0	0
4	0.0004	0.0009	0	0.0001
5	0.0004	0.0011	0	0
6	0.0004	0.001	0	0
7	0.0004	0.0011	0	0
8	0.0005	0.001	0	0
9	0.0005	0.0011	0	0
10	0.0005	0.0011	0	0.0001
Average	**0.00044**	**0.00398**	**0**	**0.00003**

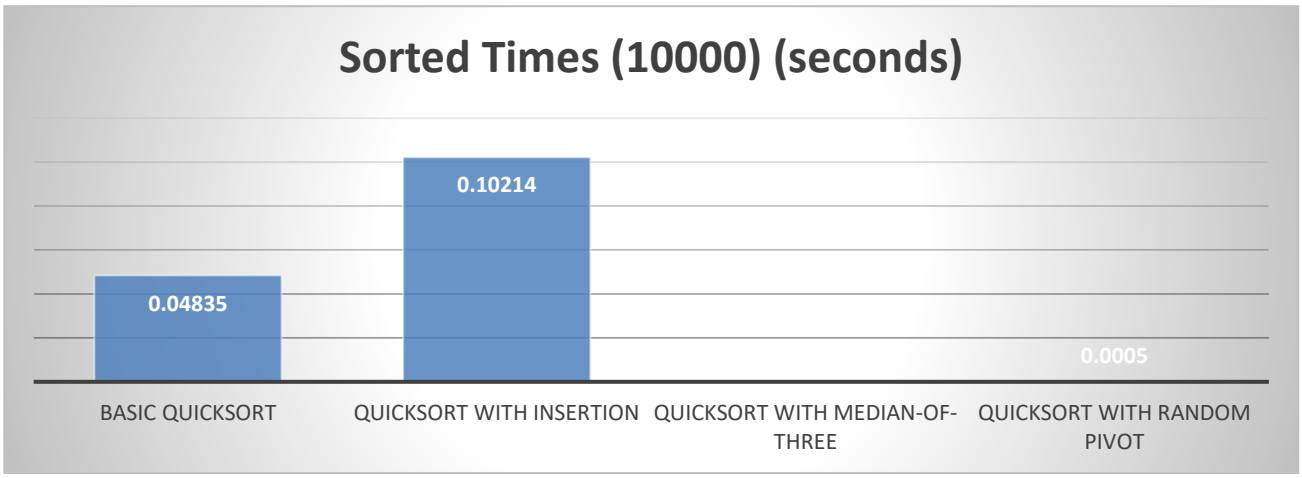
Sorted Times (10000) (seconds)				
Test Number	**Basic QuickSort**	**Quicksort With Insertion**	**Quicksort With Median-of-three**	**Quicksort With Random Pivot**
1	0.0461	0.1233	0.0003	0.0005
2	0.0489	0.0735	0.0004	0.0005
3	0.0472	0.075	0.0002	0.0005
4	0.0466	0.1025	0.0002	0.0005
5	0.0501	0.1834	0.0002	0.0005
6	0.0488	0.13	0.0003	0.0005
7	0.0485	0.0991	0.0005	0.0005
8	0.0509	0.097	0.0002	0.0005
9	0.0473	0.0708	0.0002	0.0005
10	0.0491	0.0668	0.0003	0.0005
Average	**0.04835**	**0.10214**	**0.00028**	**0.0005**

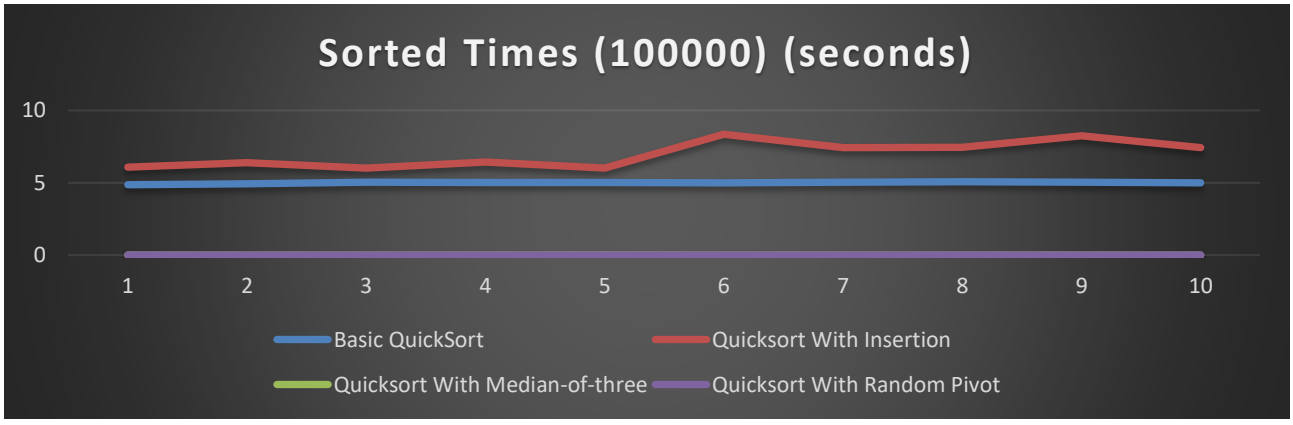
Sorted Times (100000) (seconds)				
Test Number	**Basic QuickSort**	**Quicksort With Insertion**	**Quicksort With Median-of-three**	**Quicksort With Random Pivot**
1	4.8647	6.0713	0.002	0.0061
2	4.9369	6.3853	0.0021	0.0063
3	5.0359	6.0165	0.0021	0.0062
4	5.0116	6.4358	0.0022	0.0062
5	5.0088	6.0153	0.0025	0.0066
6	5.0018	8.3516	0.0023	0.0063
7	5.0368	7.4183	0.0021	0.0062
8	5.0706	7.448	0.0021	0.0061
9	5.0358	8.2405	0.002	0.0062
10	4.9984	7.4283	0.0024	0.0068
Average	**5.00013**	**6.98109**	**0.00218**	**0.0063**

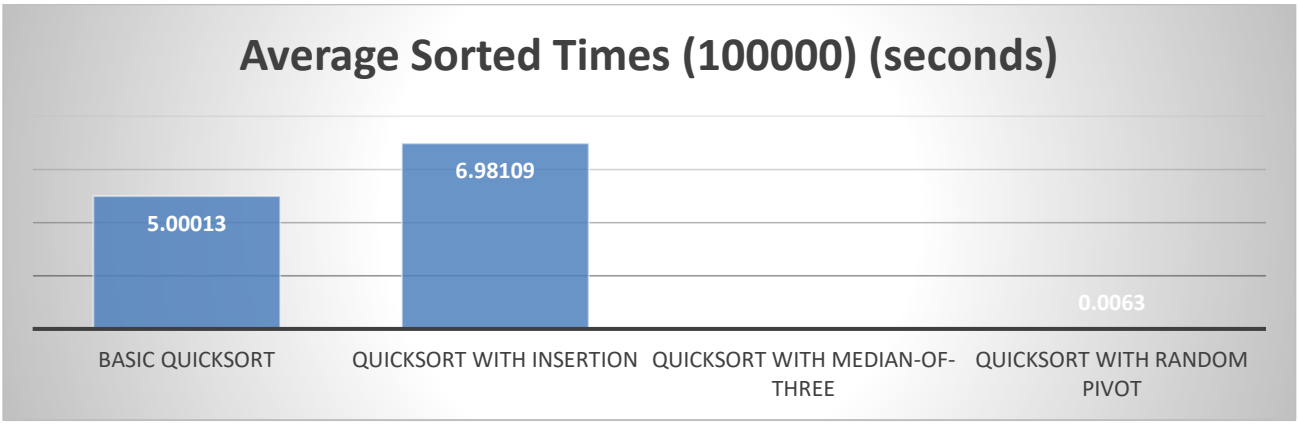



Average Sorted Times (1000) (seconds)
0.00398
0.00044
0.00003
BASIC QUICKSORT
QUICKSORT WITH INSERTION
QUICKSORT WITH MEDIAN-OF-THREE
QUICKSORT WITH RANDOM PIVOT


Sorted Times (10000) (seconds)
0.2
0.15
0.1
0.05
0
1
2
3
4
5
6
7
8
9
10
Basic QuickSort
Quicksort With Insertion
Quicksort With Median-of-three
Quicksort With Random Pivot


Sorted Times (10000) (seconds)
0.10214
0.04835
0.0005
BASIC QUICKSORT
QUICKSORT WITH INSERTION
QUICKSORT WITH MEDIAN-OF-THREE
QUICKSORT WITH RANDOM PIVOT

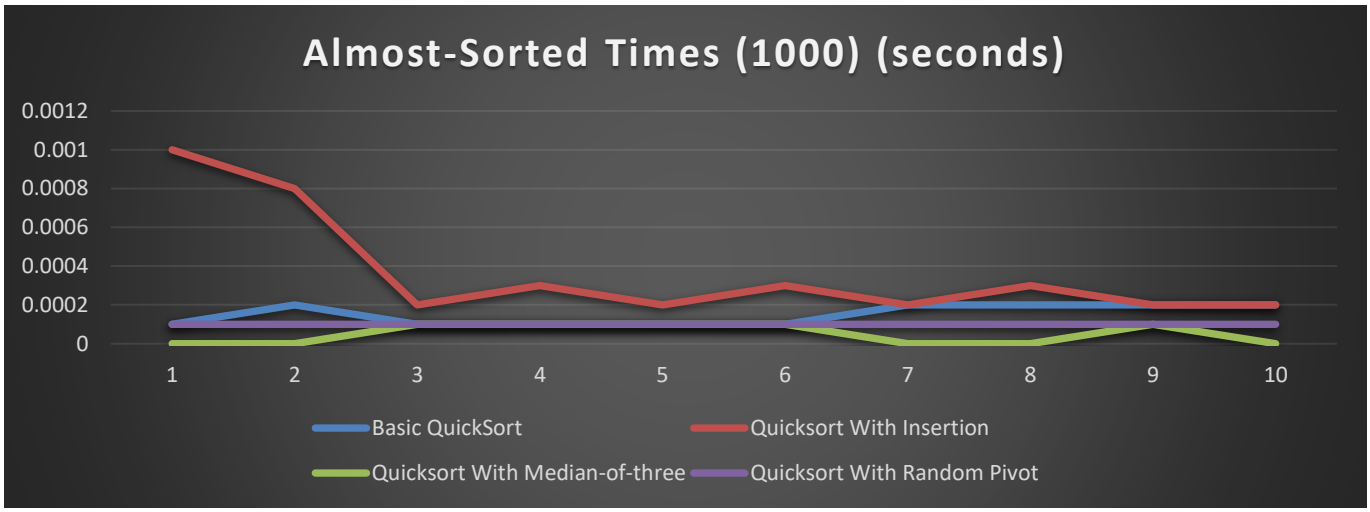

Sorted Times (100000) (seconds)
10
5
0
1
2
3
4
5
6
7
8
9
10
Basic QuickSort
Quicksort With Insertion
Quicksort With Median-of-three
Quicksort With Random Pivot

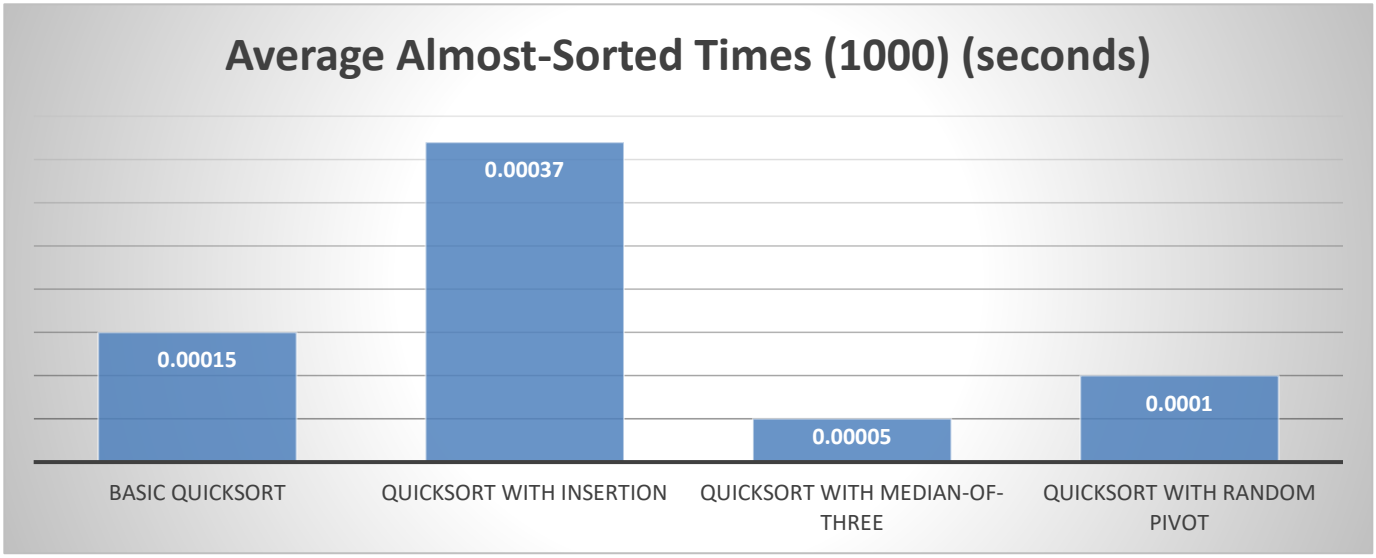


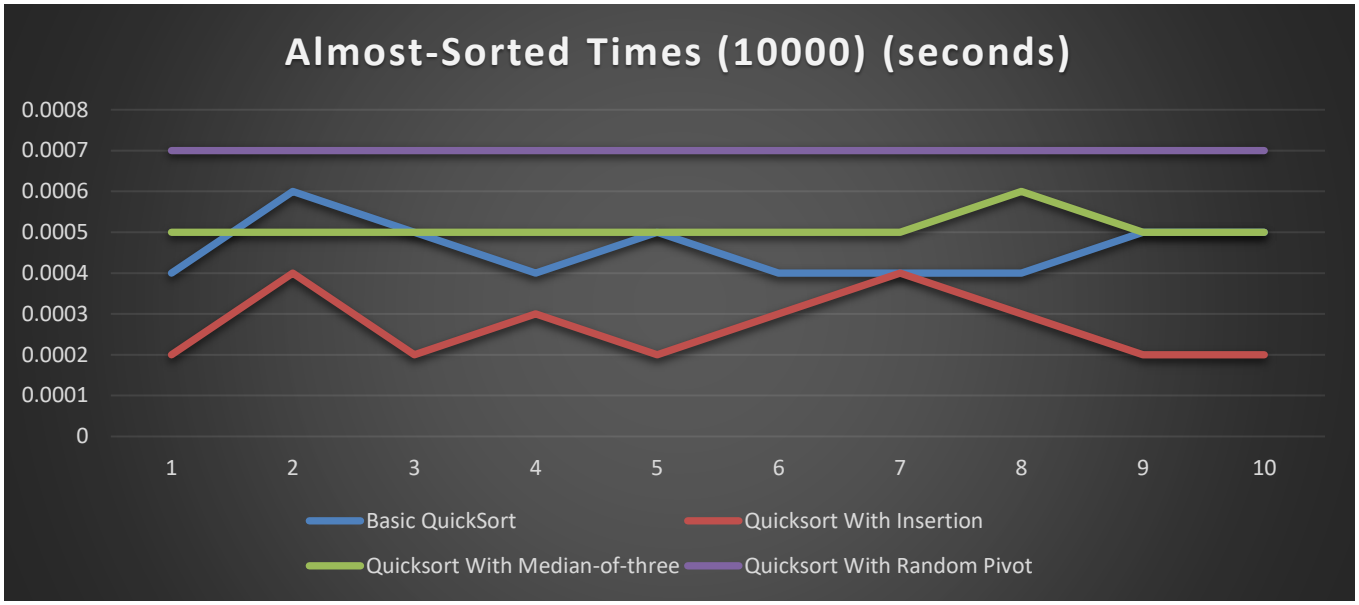
- Median of three seems to be the fastest when the array is already sorted, along with a random pivot point Quicksort with insertion taking the most time.

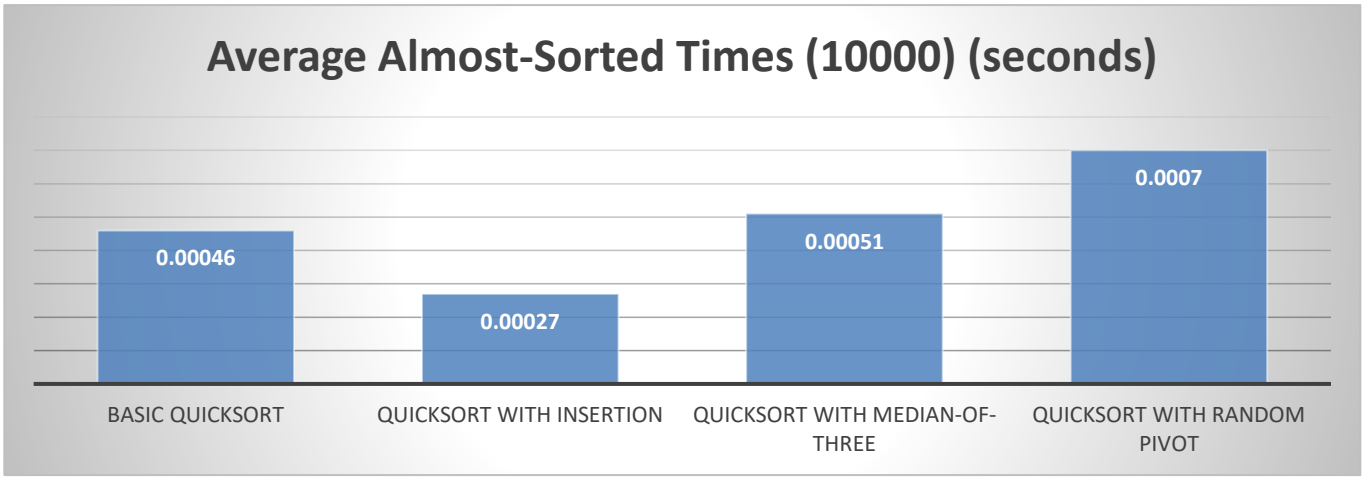
Almost-Sorted Times (1000) (seconds)				
Test Number	**Basic QuickSort**	**Quicksort With Insertion**	**Quicksort With Median-of-three**	**Quicksort With Random Pivot**
1	0.0001	0.001	0	0.0001
2	0.0002	0.0008	0	0.0001
3	0.0001	0.0002	0.0001	0.0001
4	0.0001	0.0003	0.0001	0.0001
5	0.0001	0.0002	0.0001	0.0001
6	0.0001	0.0003	0.0001	0.0001
7	0.0002	0.0002	0	0.0001
8	0.0002	0.0003	0	0.0001
9	0.0002	0.0002	0.0001	0.0001
10	0.0002	0.0002	0	0.0001
Average	**0.00015**	**0.00037**	**0.00005**	**0.0001**
Almost-Sorted Times (10000) (seconds)				
Test Number	**Basic QuickSort**	**Quicksort With Insertion**	**Quicksort With Median-of-three**	**Quicksort With Random Pivot**
1	0.0004	0.0002	0.0005	0.0007
2	0.0006	0.0004	0.0005	0.0007
3	0.0005	0.0002	0.0005	0.0007
4	0.0004	0.0003	0.0005	0.0007
5	0.0005	0.0002	0.0005	0.0007
6	0.0004	0.0003	0.0005	0.0007
7	0.0004	0.0004	0.0005	0.0007
8	0.0004	0.0003	0.0006	0.0007
9	0.0005	0.0002	0.0005	0.0007
10	0.0005	0.0002	0.0005	0.0007
Average	**0.00046**	**0.00027**	**0.00051**	**0.0007**

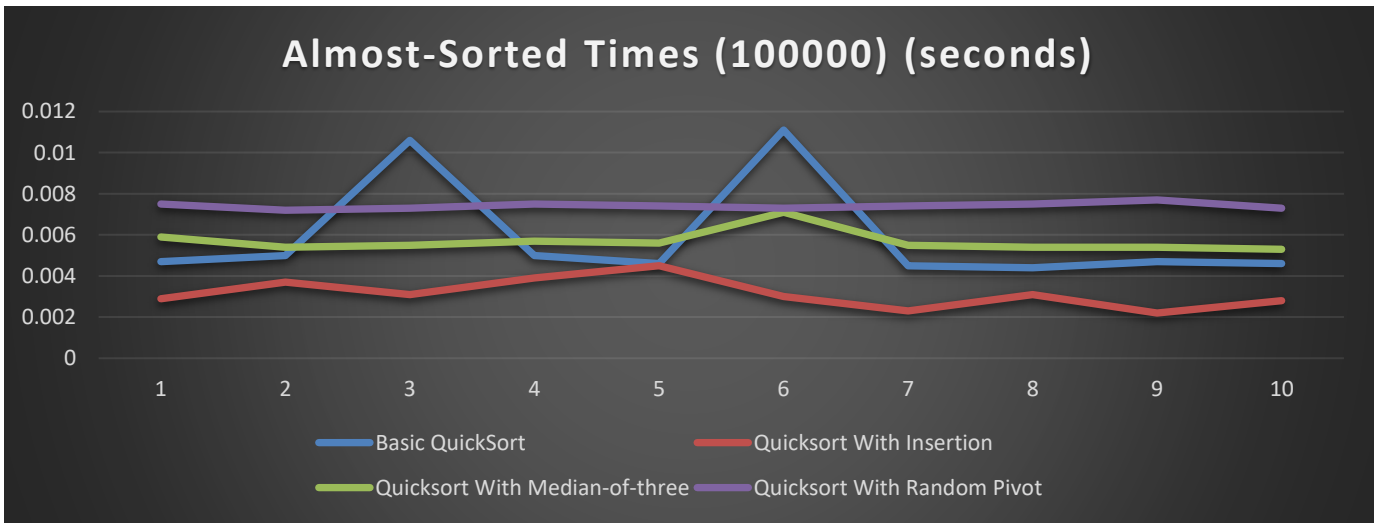
Almost-Sorted Times (100000) (seconds)				
Test Number	**Basic QuickSort**	**Quicksort With Insertion**	**Quicksort With Median-of-three**	**Quicksort With Random Pivot**
1	0.0047	0.0029	0.0059	0.0075
2	0.005	0.0037	0.0054	0.0072
3	0.0106	0.0031	0.0055	0.0073
4	0.005	0.0039	0.0057	0.0075
5	0.0046	0.0045	0.0056	0.0074
6	0.0111	0.003	0.0071	0.0073
7	0.0045	0.0023	0.0055	0.0074
8	0.0044	0.0031	0.0054	0.0075
9	0.0047	0.0022	0.0054	0.0077
10	0.0046	0.0028	0.0053	0.0073
Average	**0.00592**	**0.00315**	**0.00568**	**0.00741**

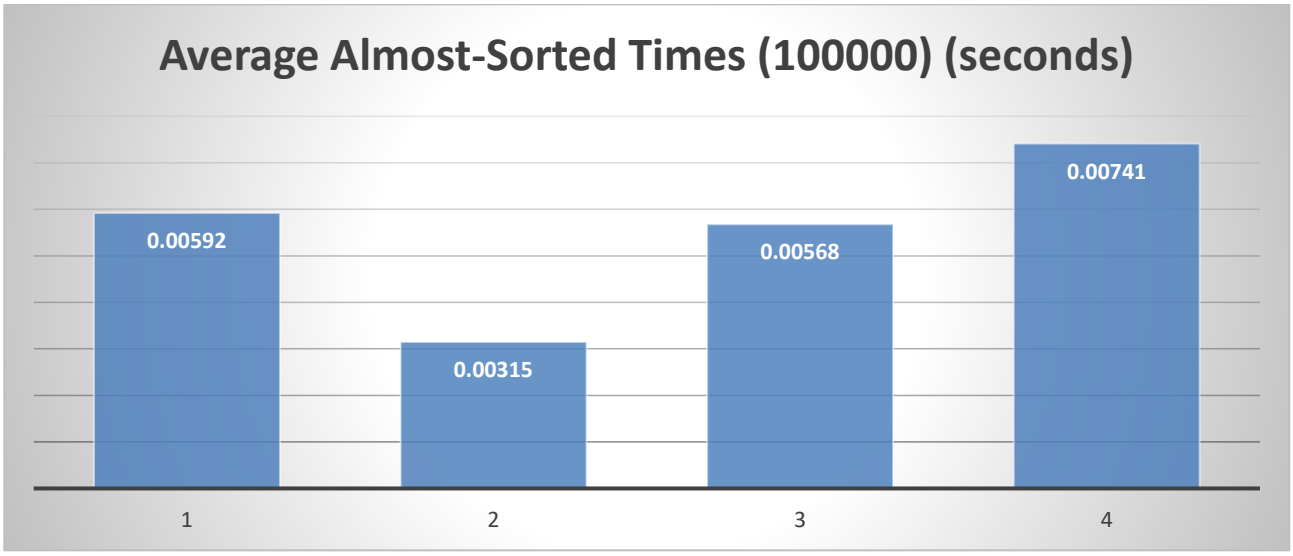




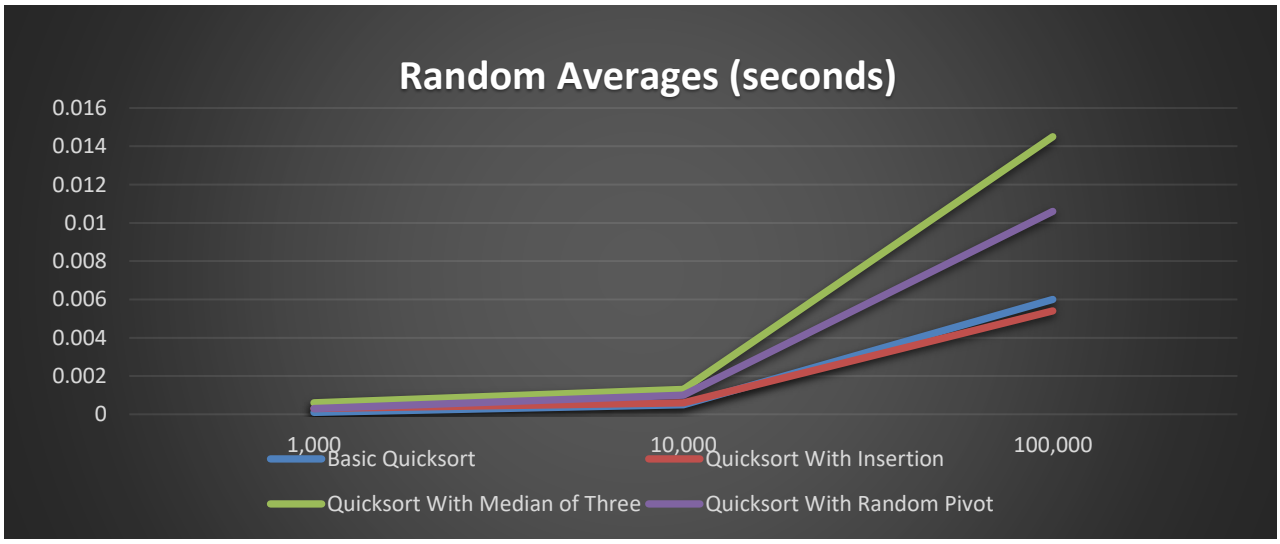

Almost-Sorted Times (10000) (seconds)
0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0
1
2
3
4
5
6
7
8
9
10
Basic QuickSort
Quicksort With Insertion
Quicksort With Median-of-three
Quicksort With Random Pivot

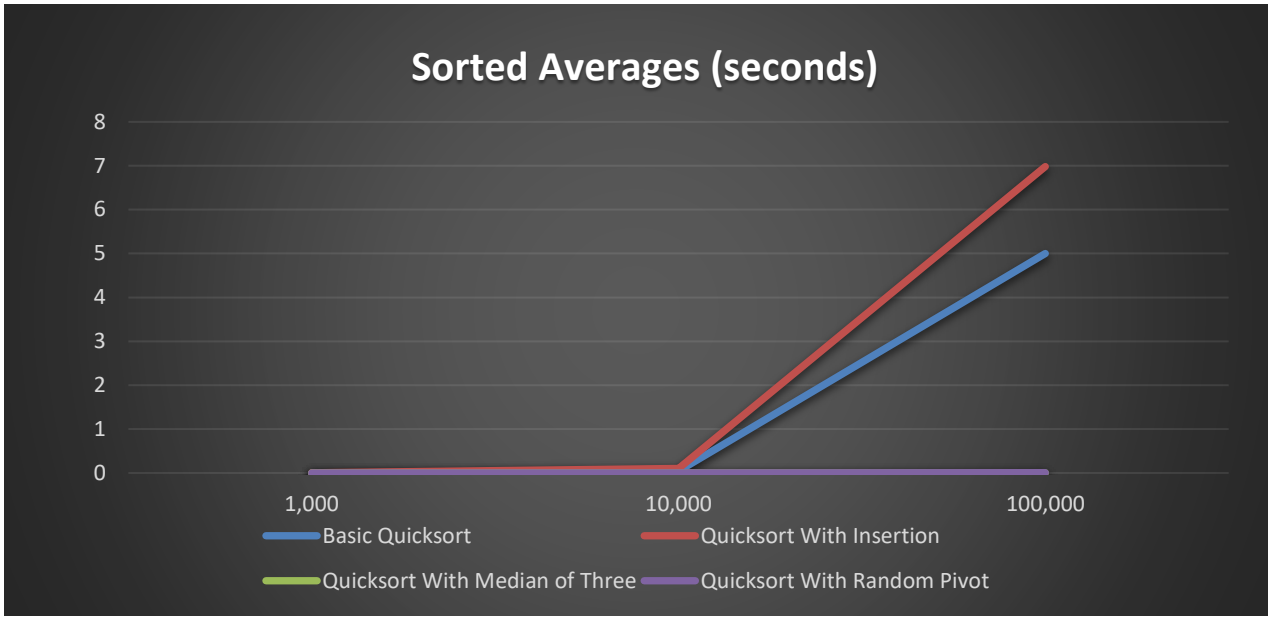

Average Almost-Sorted Times (10000) (seconds)
0.00046
0.00027
0.00051
0.0007
BASIC QUICKSORT
QUICKSORT WITH INSERTION
QUICKSORT WITH MEDIAN-OF-THREE
QUICKSORT WITH RANDOM PIVOT

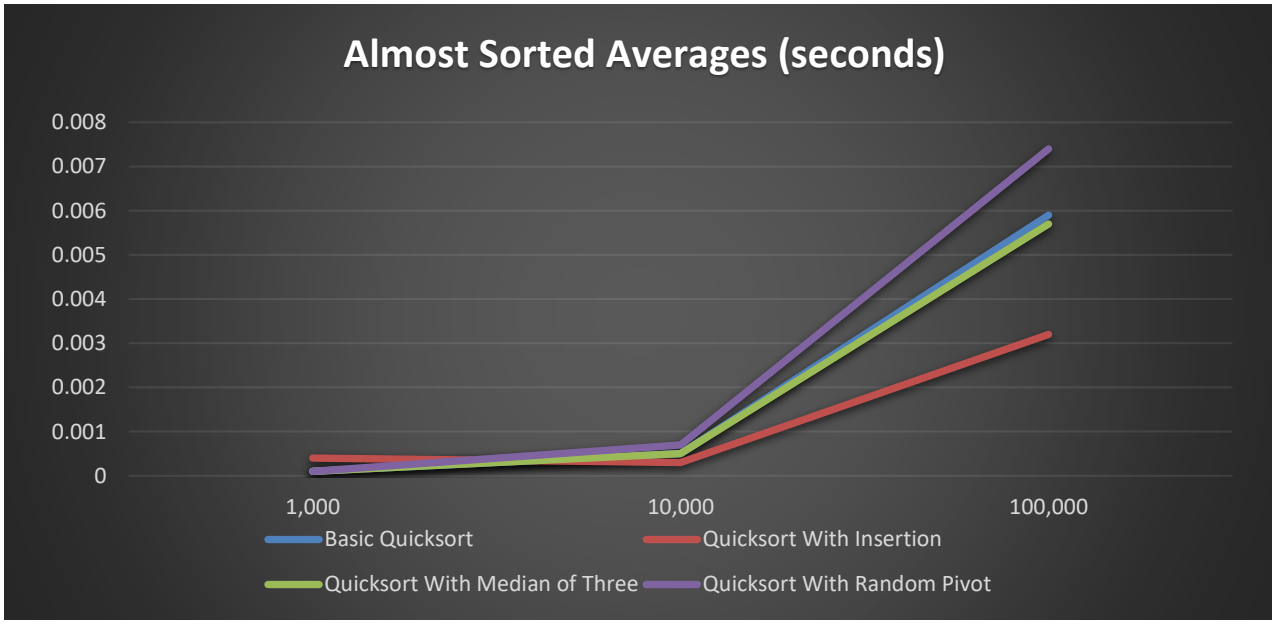

Almost-Sorted Times (100000) (seconds)
0.012
0.01
0.008
0.006
0.004
0.002
0
1
2
3
4
5
6
7
8
9
10
Basic QuickSort
Quicksort With Insertion
Quicksort With Median-of-three
Quicksort With Random Pivot



- Quicksort with insertion is great for large sums that are almost sorted unless it's a small amount if data in which quicksort takes a bit longer to start.
- The algorithms quick sort and quicksort with median of three seem to be close in time for large data sets that are almost sorted.







Conclusion

Time complexity is not the only consideration when designing and evaluating algorithms. All of this implement quicksort in some way, because of this, start time should be relatively similar. The algorithms quick sort with random pivot values and Quicksort with medium of three seem to perform the fastest in sorted arrays, meanwhile as the data gets bigger Quicksort with insertion works the fastest. This is because Quicksort is really inefficient when it comes to the end of the sort. So switching to a faster algorithm for nearly sorted data (insertion) makes sense. A stack overflow error could occur using the quick sort algorithms with higher array sizes. While testing the quick sort implementations, 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. This allowed me to completely run all my tests

Random Sorted Arrays

- Quicksort with insertion has a slower start up speed than basic quicksort but overtakes it with larger data. Random value arrays seem to be the base case scenario for this .
- Quicksort with median of three seem to be worse case and quicksort with random pivot has an average case. The time complexities are listed at the top.

Sorted Arrays

- Quicksort with median of three and Quicksort with a random pivot have the best case scenario to handle already sorted arrays.
- I would still say that Basic Quicksort has an average case while Quicksort with insertion has a worst case.

Almost Sorted Arrays

- Quicksort with Insertion is best used on almost sorted array. This agrees with the theoretical information as insertion is much faster as finshing almost sorted array, but is slower on start up which is why pivot over takes it for the 1000.
- Basic and median of three have very are going to be average cases while quicksort with random pivot seems to be worst case. However for smaller values they are faster on start.

In conclusion, the experimental results appear to agree with the theoretical information. All explanations of each type of implementation can be found at the top under definition and what each one does for time complexity.