/*

* Title : Algorithm Efficiency and Sorting

* Author : Berk Yıldız

* ID : 21502040

* Section : 2

* Assignment : 1

* Description : Solutions for Question 1 and Question 3

*/

Question 1

(a) Let find two positive constants c and n_0

$0 \leq 4n^5 + 3n^2 + 1 \leq cn^5$ for all $n \geq n_0$

$4 + \frac{3}{n^3} + \frac{1}{n^5} \leq c$ for all $n \geq n_0$

For $c = 8$ and $n_0 = 1$, $4n^5 + 3n^2 + 1 \leq 8n^5$ for all $n \geq 1$
For $c = 10$ and $n_0 = 3$, $4n^5 + 3n^2 + 1 \leq 10n^5$ for all $n \geq 3$

(b) $\boldsymbol{T(n) = T(n-1) + n^2}$ $\boldsymbol{T(1) = 1}$

$$T(n-1) = T(n-2) + (n-1)^2$$

$$T(n) = T(n-2) + (n-1)^2 + n^2$$

$$T(n-2) = T(n-3) + (n-2)^2$$

$$T(n) = T(n-3) + (n-2)^2 + (n-1)^2 + n^2$$

$$T(n-3) = T(n-4) + (n-3)^2$$

$$T(n) = T(n-4) + (n-3)^2 + (n-2)^2 + (n-1)^2 + n^2$$

General Formula: $T(n-k) + (n-k+1)^2 + (n-k+2)^2 + \cdots + (n-1)^2 + n^2$

$$= T(1) + \frac{n(n+1)(2n+1)}{6} - 1 = \boldsymbol{O(n^3)}$$

$\boldsymbol{T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{2}} \qquad \boldsymbol{T(1) = 1}$

$$T(n/2) = 2T\left(\frac{n}{4}\right) + \frac{n}{4}$$

$$T(n) = 4T\left(\frac{n}{4}\right) + \frac{2n}{4} + \frac{n}{2}$$

$$T(n/4) = 2T\left(\frac{n}{8}\right) + \frac{n}{8}$$

$$T(n) = 8T\left(\frac{n}{8}\right) + \frac{4n}{8} + \frac{2n}{4} + \frac{n}{2}$$

General Formula: $2^k T\left(\frac{n}{2^k}\right) + \frac{kn}{2}$

$\frac{n}{2k} = 1 \qquad n = 2^k \qquad \log_2 n = k$

$= 2^{log_2 n} T(1) + (log_2 n)(\frac{n}{2})$

$= n + \frac{n}{2}\log_2 n = \boldsymbol{O(n \log n)}$

(c) Selection Sort

Italics and underlined are selected, **bolds** are sorted position.

Initial Array	8	4	5	1	*9*	6	2	3
After 1st swap	*8*	4	5	1	6	2	3	**9**
After 2nd swap	4	5	1	*6*	2	3	**8**	**9**
After 3rd swap	4	*5*	1	2	3	**6**	**8**	**9**
After 4th swap	*4*	1	2	3	**5**	**6**	**8**	**9**
After 5th swap	**1**	**2**	**3**	**4**	**5**	**6**	**8**	**9**

Bubble Sort

Italics and underlined are selected, **bolds** are sorted position.

Pass 1

Initial Array	*8*	*4*	5	1	9	6	2	3
	4	*8*	*5*	1	9	6	2	3
	4	5	*8*	*1*	9	6	2	3
	4	5	1	*8*	*9*	6	2	3
	4	5	1	8	*9*	*6*	2	3
	4	5	1	8	6	*9*	*2*	3
	4	5	1	8	6	2	*9*	*3*
	4	5	1	8	6	2	3	**9**

Pass 2

Initial Array	*4*	*5*	1	8	6	2	3	9
	4	*5*	*1*	8	6	2	3	9
	4	1	*5*	*8*	6	2	3	9
	4	1	5	*8*	*6*	2	3	9
	4	1	5	6	*8*	*2*	3	9
	4	1	5	6	2	*8*	*3*	9
	4	1	5	6	2	3	**8**	**9**

Pass 3

Initial Array	*4*	*1*	5	6	2	3	8	9
	1	*4*	*5*	6	2	3	8	9
	1	4	*5*	*6*	2	3	8	9
	1	4	5	*6*	*2*	3	8	9
	1	4	5	2	*6*	*3*	8	9
	1	4	5	2	3	**6**	**8**	**9**

Pass 4

Initial Array	*1*	*4*	5	2	3	6	8	9
	1	*4*	*5*	2	3	6	8	9
	1	4	*5*	*2*	3	6	8	9
	1	4	2	*5*	*3*	6	8	9
	1	4	2	3	**5**	**6**	**8**	**9**

Pass 5

Initial Array	*1*	*4*	2	3	5	6	8	9
	1	*4*	*2*	3	5	6	8	9
	1	2	*4*	*3*	5	6	8	9
	1	**2**	**3**	**4**	**5**	**6**	**8**	**9**

Question 3

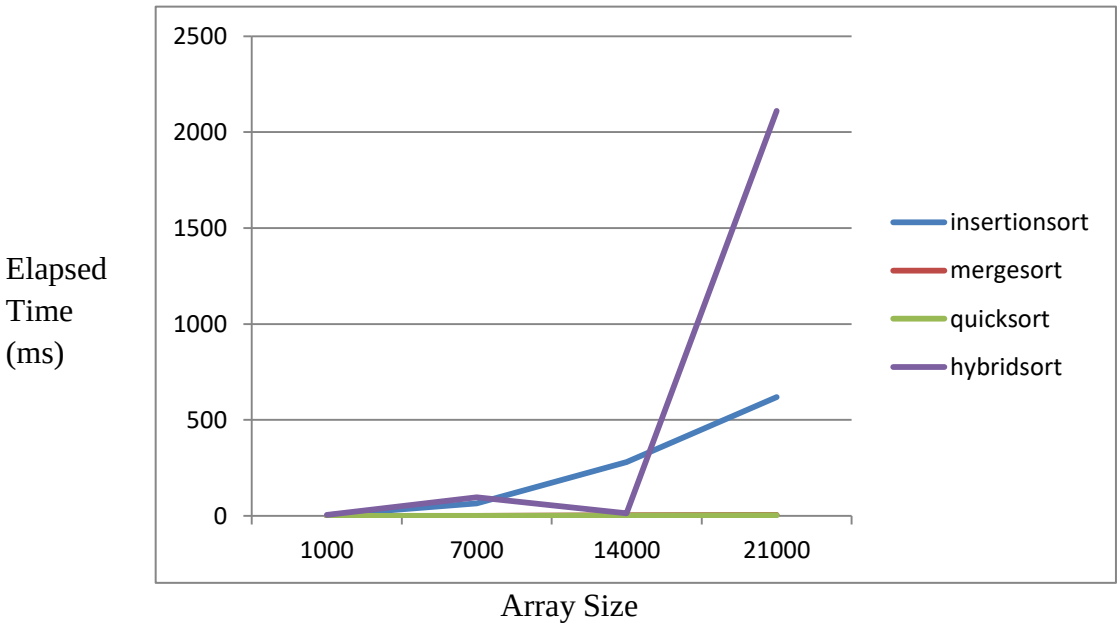
Arrays	Elapsed Time (in milliseconds)			
	Insertion Sort	**Merge Sort**	**Quick Sort**	**Hybrid Sort**
R1K	3	0	0	5
R7K	66	1	1	97
R14K	281	3	2	13
R21K	619	6	3	2111
A1K	0	1	14	6
A7K	0	2	244	235
A14K	1	2	969	972
A21K	0	4	2236	2144
D1K	3	0	6	9
D7K	133	1	133	363
D14K	558	2	510	1464
D21K	1235	3	1059	3217

Arrays	Number of Key Comparisons			
	Insertion Sort	**Merge Sort**	**Quick Sort**	**Hybrid Sort**
R1K	497289	9714	10807	219165
R7K	24327835	87686	104609	9786379
R14K	97707499	189310	237620	1226890
R21K	219821327	296527	370290	217660264
A1K	999	6043	500499	501498
A7K	6999	53179	24503499	24510498
A14K	13999	113359	98006999	98020998
A21K	20999	177507	220510499	220531498
D1K	999973	5942	491559	990064
D7K	48998469	51045	22189797	46686297
D14K	195994137	109858	81660503	179653503
D21K	440986533	171503	169313613	389803113

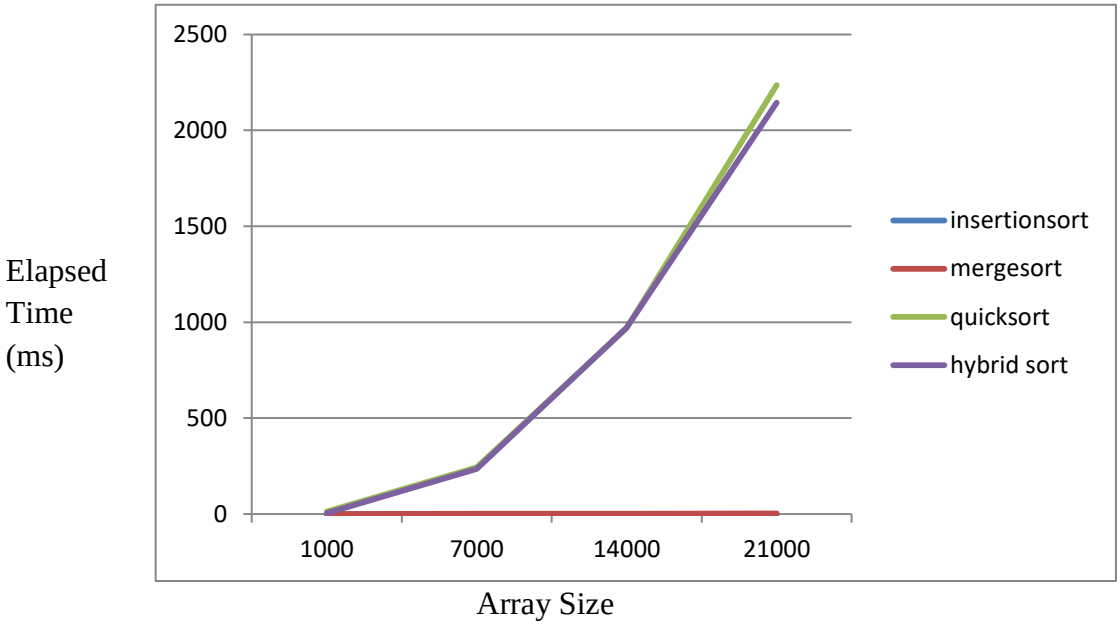
Arrays	Number of Data Moves			
	Insertion Sort	**Merge Sort**	**Quick Sort**	**Hybrid Sort**
R1K	251142	22949	19547	645904
R7K	12181415	200613	172794	29226891
R14K	48888747	429229	432288	3402629
R21K	109963161	669461	546342	652457961
A1K	2997	22949	1503495	1506497
A7K	20997	200613	73524495	73545497
A14K	41997	429229	294048995	294090997
A21K	62997	669461	661573495	661636497
D1K	502484	22949	740036	2237550
D7K	24516732	200613	33478941	106982440
D14K	98032066	429229	124847936	418854935
D21K	220545764	669461	263579412	925089911

Performance of Sorting Algorithms

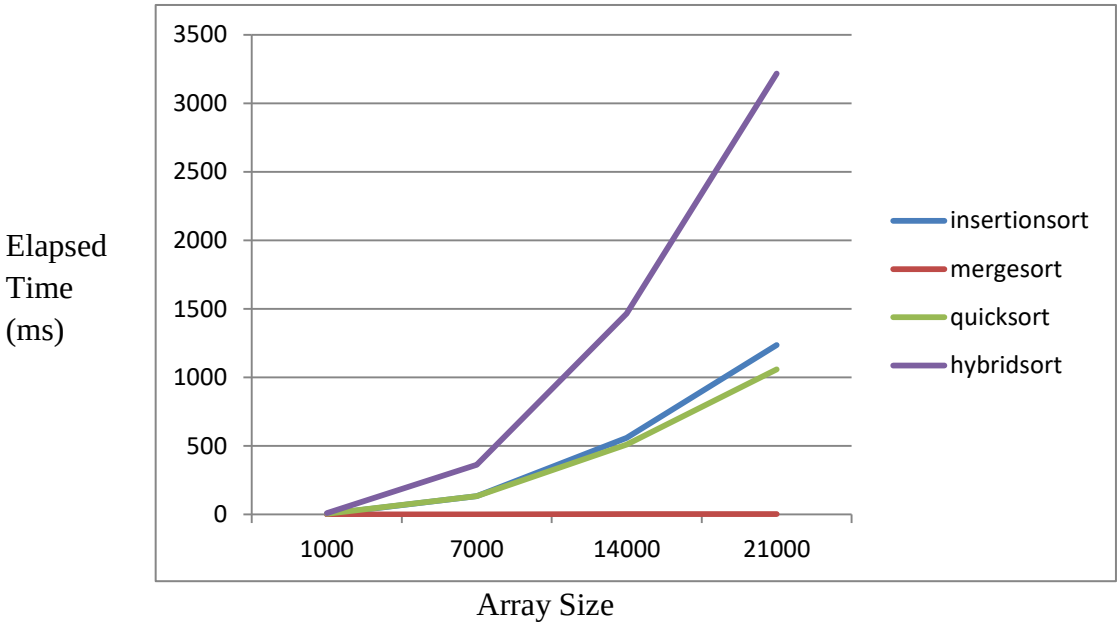
(Random)



(Ascending)



(Descending)



Interpret and compare your empirical results with the theoretical ones for each sorting algorithm. Explain any discrepancies between the empirical and theoretical results, if any.

For the random arrays, which we can define as the average case, results of the elapsed times are consistent with the theoretical results. As expected quick sort and merge sort executed in very short time differently from the insertion sort and hybrid sort.

For the ascending array, results of the elapsed times are mostly consistent with the theoretical results. As expected, insertion sort executed in very short time like merge sort. The execution time for quick sort and hybrid sort are extremely long than the other algorithms.

For the descending arrays, which we can define as the worst case, results of the elapsed times are consistent with the theoretical results. As expected growth rate of insertion sort, quick sort and merge sort are much more higher than the merge sort. The reason is growth rate of merge sort is O(nlogn) in all cases however O(n^2) for the worst cases of other sorting algorithms.

In general, when should insertion sort algorithm be preferred over merge sort and quick sort algorithms?

If the elements of the input array is sorted or likely to be sorted, insertion sort algorithm can be preferred over merge sort and quick sort algorithm.

In general, when should merge sort algorithm be preferred over quick sort algorithm?

If the elements of the input array is sorted or likely to be sorted in ascending or descending order, merge sort algorithm can be preferred over quick sort algorithm.

Does hybrid sort algorithm have any advantages/disadvantages over the quick sort algorithm? Explain the reasons, if any.

Hybrid sort algorithm does not have any remarkable advantages over the quick sort algorithm in any case, even it has disadvantages over the quick sort algorithm. The reason is, already hybrid sort starts with quick sorti however it returns to insertion sort when the partition becomes less than 10 elements and the growth rate of the insertion sort algorithm is higher than the quick sort algorithm. So the growth rate of the hybrid sort grows like the insertion sort. The only little advantage can be defined for the hybrid sort is ascending arrays case because insertion sort is the best algorithm for the sorted arrays in ascending order.