Kübra Okumuş
21600980

Question 1

a) Insertion sort

In Insertion Sort, the array is divided into two parts with an imaginary wall. The first sub array called sorted and the second one is called unsorted. At the beginning, only the first element of the original array is considered in the sorted part and others are in the unsorted array. Then, first element of the unsorted part is selected as the key (yellow highlighted numbers). The key element is switched with the number on the left of the key, until the number on the left of the key is less than the key. In this way, the left of the wall will be sorted. This continues until the whole array is sorted.

Sorted | Unsorted

4 | 8, 3, 7, 6, 2, 1, 5 – Original array

4, 8 | 3, 7, 6, 2, 1, 5

3, 4, 8 | 7, 6, 2, 1, 5

3, 4, 7, 8 | 6, 2, 1, 5

3, 4, 6, 7, 8 | 2, 1, 5

2, 3, 4, 6, 7, 8 | 1, 5

1, 2, 3, 4, 6, 7, 8 | 5

1, 2, 3, 4, 5, 6, 7, 8 – Sorted array

b) Selection sort

In selection sort, an imaginary wall divided the array into two subarrays, unsorted and sorted. At the beginning, sorted part is empty. The greatest number (yellow highlighted numbers) is picked from the unsorted array and it is transferred to the sorted part, right after the imaginary wall. Then, the greatest number of the unsorted part is selected again. These steps are repeated until the full array is sorted.

Unsorted | Sorted

4, 8, 3, 7, 6, 2, 1, 5 |

4, 3, 7, 6, 2, 1, 5 | 8

4, 3, 6, 2, 1, 5 | 7, 8

4, 3, 2, 1, 5 | 6, 7, 8

4, 3, 2, 1 | 5, 6, 7, 8

Kübra Okumuş
21600980

3, 2, 1 | 4, 5, 6, 7, 8

2, 1 | 3, 4, 5, 6, 7, 8

1 | 2, 3, 4, 5, 6, 7, 8

1, 2, 3, 4, 5, 6, 7, 8

c) Bubble sort

In bubble sort, the array is divided into two parts with an imaginary wall. At the beginning, the sorted part is empty. In each step, the two element of the array is compared and the greater one is placed to the right of the smaller one. In this way the greater number is transferred to the sorted part of the array. This means one **pass**. This continues until the whole numbers are at the sorted part. In this case, (n-1) = 7 **passes** (maximum passes) are needed.

Unsorted | Sorted

Pass 1

4, 8, 3, 7, 6, 2, 1, 5 | – Original array

4, 8, 3, 7, 6, 2, 1, 5 |

4, 3, 8, 7, 6, 2, 1, 5 |

4, 3, 7, 8, 6, 2, 1, 5 |

4, 3, 7, 6, 8, 2, 1, 5 |

4, 3, 7, 6, 2, 8, 1, 5 |

4, 3, 7, 6, 2, 1, 8, 5 |

4, 3, 7, 6, 2, 1, 5 | 8

Pass 2

4, 3, 7, 6, 2, 1, 5 | 8

3, 4, 7, 6, 2, 1, 5 | 8

3, 4, 7, 6, 2, 1, 5 | 8

3, 4, 6, 7, 2, 1, 5 | 8

3, 4, 6, 2, 7, 1, 5 | 8

3, 4, 6, 2, 1, 7, 5 | 8

3, 4, 6, 2, 1, 5 | 7, 8

Pass 3

3, 4, 6, 2, 1, 5 | 7, 8

3, 4, 6, 2, 1, 5 | 7, 8

3, 4, 6, 2, 1, 5 | 7, 8

3, 4, 2, 6, 1, 5 | 7, 8

3, 4, 2, 1, 6, 5 | 7, 8

3, 4, 2, 1, 5 | 6, 7, 8

Pass 4

3, 4, 2, 1, 5 | 6, 7, 8

3, 4, 2, 1, 5 | 6, 7, 8

3, 2, 4, 1, 5 | 6, 7, 8

3, 2, 1, 4, 5 | 6, 7, 8

3, 2, 1, 4 | 5, 6, 7, 8

Kübra Okumuş
21600980

Pass 5

3, 2, 1, 4 | 5, 6, 7, 8

2, 3, 1, 4 | 5, 6, 7, 8

2, 1, 3, 4 | 5, 6, 7, 8

2, 1, 3 | 4, 5, 6, 7, 8

Pass 6

2, 1, 3 | 4, 5, 6, 7, 8

1, 2, 3 | 4, 5, 6, 7, 8

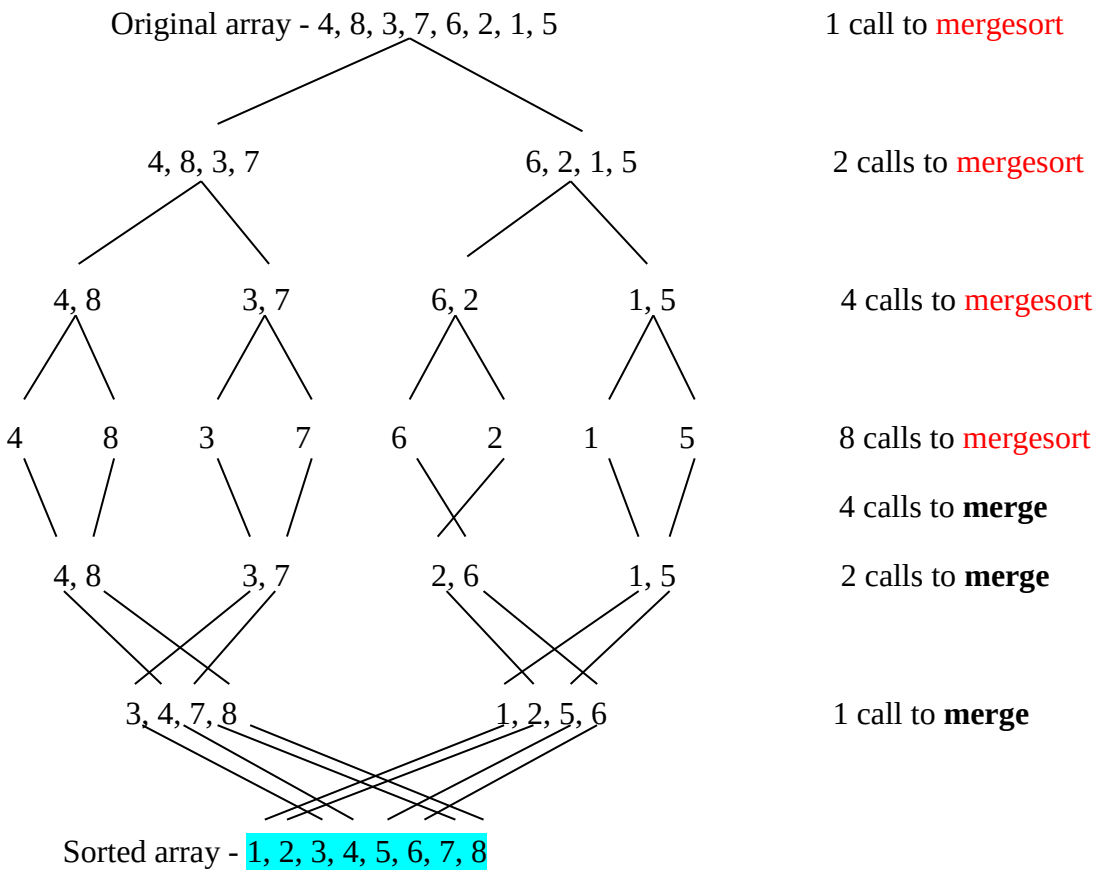
1, 2 | 3, 4, 5, 6, 7, 8

Pass 7

1, 2 | 3, 4, 5, 6, 7, 8

| 1, 2, 3, 4, 5, 6, 7, 8

d) Merge sort



In the order and indentation with respect to recursive calls:

4, 8, 3, 7, 6, 2, 1, 5 mergesort

4, 8, 3, 7 mergesort

4, 8 mergesort

Kübra Okumuş
21600980

4	mergesort
8	mergesort
4, 8	**merge**
3, 7	mergesort
3	mergesort
7	mergesort
3, 7	**merge**
3, 4, 7, 8	**merge**
6, 2, 1, 5	mergesort
6, 2	mergesort
6	mergesort
2	mergesort
2, 6	**merge**
1, 5	mergesort
1	mergesort
5	mergesort
1, 5	**merge**
1, 2, 5, 6	**merge**
1, 2, 3, 4, 5, 6, 7, 8	**merge**

e) Quick sort

Bold numbers: **Pivot**

Yellow highlighted numbers: firstUnknown

Blue highlighted numbers: S_1

Green highlighted numbers: S_2

Kübra Okumuş
21600980

4 8 3 7 6 2 1 5 - Original array. First call to quicksort

4 | 8 3 7 6 2 1 5 - One call to partition. 4 is the pivot and firstUnknown is 8. 8 belongs in S_2.

4 | 8 | 3 7 6 2 1 5 - S_1 is empty. 3 belongs in S_1. Hence, swap 3 and 8.

4 | 3 | 8 | 7 6 2 1 5 - 7 belongs in S_2.

4 | 3 | 8 7 | 6 2 1 5 - 6 belongs in S_2.

4 | 3 | 8 7 6 | 2 1 5 - 2 belongs in S_1. Hence, swap 2 and 8.

4 | 3 2 | 7 6 8 | 1 5 - 1 belongs in S_1. Hence, swap 1 and 7.

4 | 3 2 1 | 6 8 7 | 5 - 5 belongs in S_2.

4 | 3 2 1 | 6 8 7 5 | - Place the pivot between S_1 and S_2 (Swap the pivot and lastS_1).

1 3 2 | **4** | 6 8 7 5 - Two quicksort calls for S_1 and S_2.

For the array [1, 3, 2];

1 | 3 2 - One call to partition.

1 | 3 | 2

1 | 3 2 |

1 | 3 2 - Two quicksort calls for S_1 and S_2. (Although there is no S_1 in this case, quicksort will be called for it too)

For the array [3, 2];

3 | 2 - One call to partition.

3 | 2 |

2 | **3** - Two quicksort calls for S_1 and S_2.

For the array [6, 8, 7, 5];

6 | 8 7 5 - One call to partition.

6 | 8 | 7 5

6 | 8 7 | 5

6 | 5 | 7 8

5 | **6** | 7 8 - Two quicksort calls for S_1 and S_2.

Kübra Okumuş
21600980

For the array [7, 8];

7 | 8 - One call to partition.

7 | 8 |

7 | 8 - Two quicksort calls for S_1 and S_2.

So the array is sorted as [1, 2, 3, 4, 5, 6, 7, 8].

Question 2

mergesort:

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

$$T(1) = \Theta(1)$$

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n), \quad \text{and} \quad T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{2^2}\right) + \Theta\left(\frac{n}{2}\right) \text{ then,}$$

$$T(n) = 2^2T\left(\frac{n}{2^2}\right) + 2\Theta(n) \quad \text{and } T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{2^3}\right) + \Theta\left(\frac{n}{2^2}\right) \text{ then,}$$

$$T(n) = 2^3T\left(\frac{n}{2^3}\right) + 3\Theta(n)$$

After k steps…

$$T(n) = 2^kT\left(\frac{n}{2^k}\right) + k\Theta(n)$$

$$when\ n = 2^k \rightarrow k = log_2 n$$

$$T(n) = nT(1) + \log_2 n\,\Theta(n)$$

$$T(n) = O(n \log n)$$

quicksort:

$$T(n) = T(n-1) + \Theta(n-1)$$

$$T(1) = \Theta(1)$$

$$T(n) = T(n-1) + \Theta(n-1)$$

$$T(n) = T(n-2) + \Theta(n-1) + \Theta(n-2)$$

$$T(n) = T(n-3) + \Theta(n-1) + \Theta(n-2) + \Theta(n-3)$$

After k steps…

$$T(n) = T(n-k) + \Theta(n-1) + \Theta(n-2) + \cdots + \Theta(n-k)$$

Kübra Okumuş
21600980

$$\boldsymbol{when\ k = n - 1\ then,}$$

$$\boldsymbol{T(n) = T(1) + \Theta(n-1) + \Theta(n-2) + \cdots + \Theta(1)}$$

$$\boldsymbol{As\ T(1) = \Theta(1)\ then,}$$

$$\boldsymbol{T(n) = \Theta\left(\frac{(n-1)n}{2}\right) + \Theta(1) = O(n^2)}$$

Question 3

The Sample Output of the Program

```
WITH RANDOM INTEGERS
----Insertion Sort----
Execution took 981 milliseconds.
compCount is 452681700
moveCount is 452711698
----Merge Sort----
Execution took 11 milliseconds.
compCount is 1921892
moveCount is 2227644
----Quick Sort----
Execution took 4 milliseconds.
compCount is 1150625
moveCount is 1138574
WITH INTEGERS THAT ARE SORTED IN DESCENDING ORDER
----Insertion Sort----
Execution took 1750 milliseconds.
compCount is 900027662
moveCount is 900057660
----Merge Sort----
Execution took 9 milliseconds.
compCount is 1543466
moveCount is 2038431
----Quick Sort----
Execution took 1588 milliseconds.
compCount is 712231629
moveCount is 712340474
```

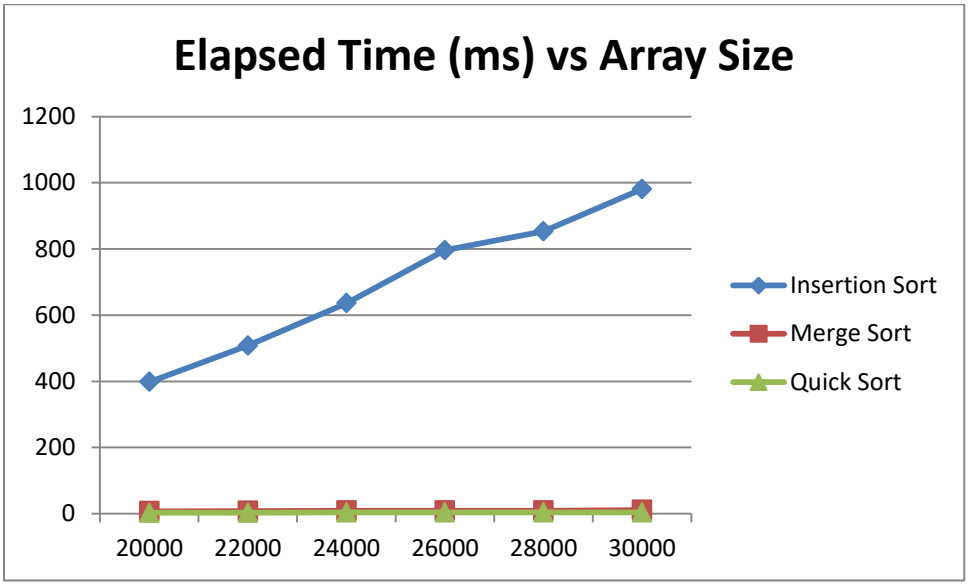
```
WITH INTEGERS THAT ARE SORTED IN ASCENDING ORDER
----Insertion Sort----
Execution took 0.18 milliseconds.
compCount is 89998
moveCount is 119996
----Merge Sort----
Execution took 11 milliseconds.
compCount is 1571478
moveCount is 2052437
----Quick Sort----
Execution took 955 milliseconds.
compCount is 900059998
moveCount is 209993
```

Kübra Okumuş
21600980

Experiment with Random Integers

Elapsed Time (ms) vs Array Size

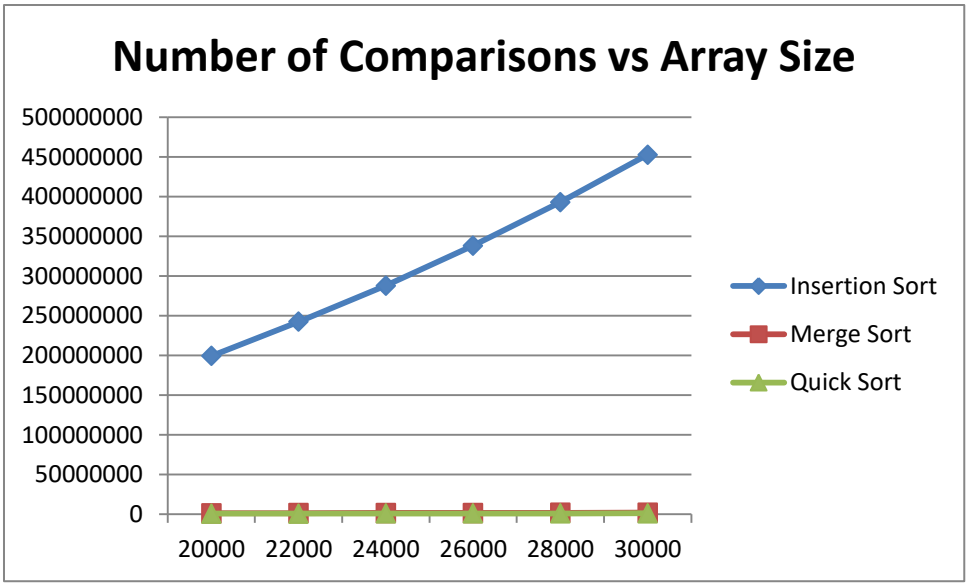
Array Size	Insertion Sort	Merge Sort	Quick Sort
20000	398	7	3
22000	508	8	3
24000	636	9	4
26000	796	9	4
28000	853	9	4
30000	981	11	4



Number of Key Comparisons vs Array Size

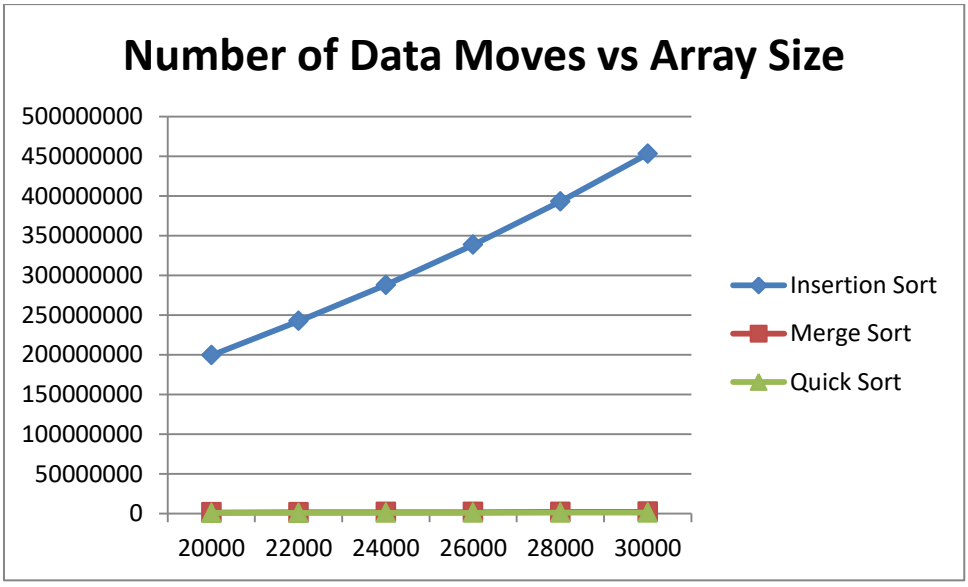
Array Size	Insertion Sort	Merge Sort	Quick Sort
20000	199302732	1236084	738689
22000	242350482	1372588	845170
24000	287765664	1509226	922525
26000	338239418	1646486	991890
28000	393016822	1783858	1062739
30000	452681700	1921892	1150625

Kübra Okumuş
21600980



Number of Data Moves vs Array Size

Array Size	Insertion Sort	Merge Sort	Quick Sort
20000	199322730	1429740	712462
22000	242372480	1588992	841657
24000	287789662	1748311	901604
26000	338265416	1907941	968003
28000	393044820	2067627	1103936
30000	452711698	2227644	1138574

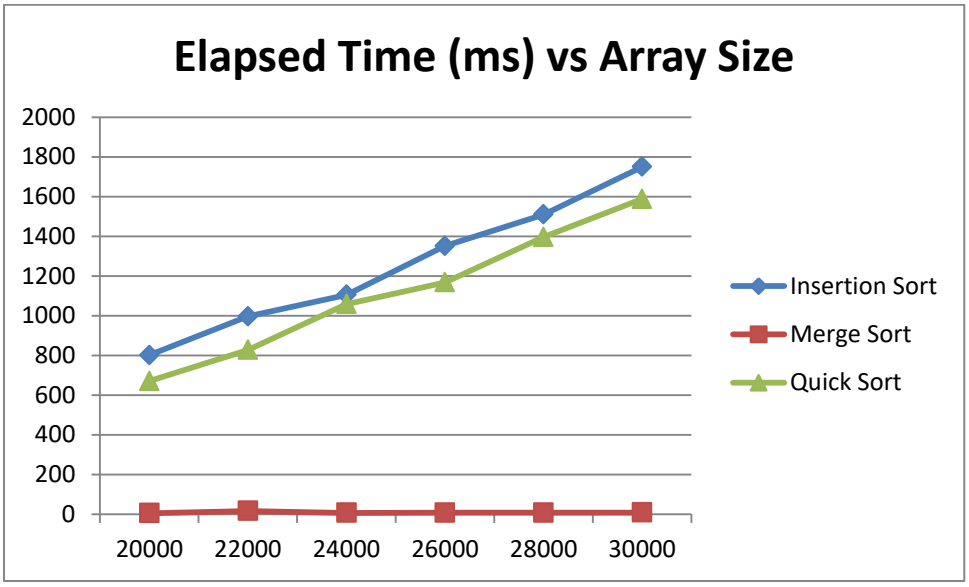


Kübra Okumuş
21600980

Experiment with Integers That are Sorted in Descending Order

Elapsed Time (ms) vs Array Size

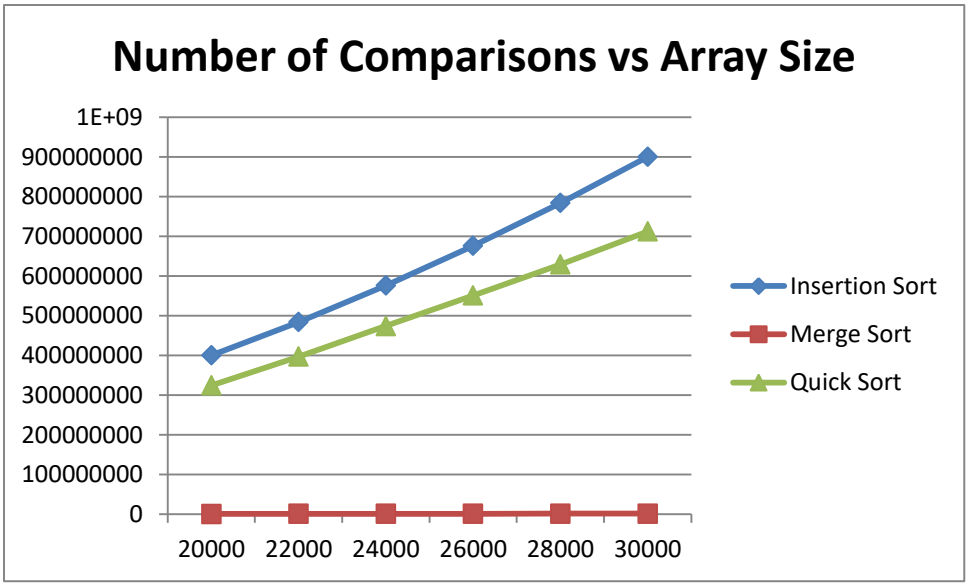
Array Size	Insertion Sort	Merge Sort	Quick Sort
20000	801	6	671
22000	996	17	828
24000	1105	7	1058
26000	1351	8	1168
28000	1510	8	1396
30000	1750	9	1588



Number of Key Comparisons vs Array Size

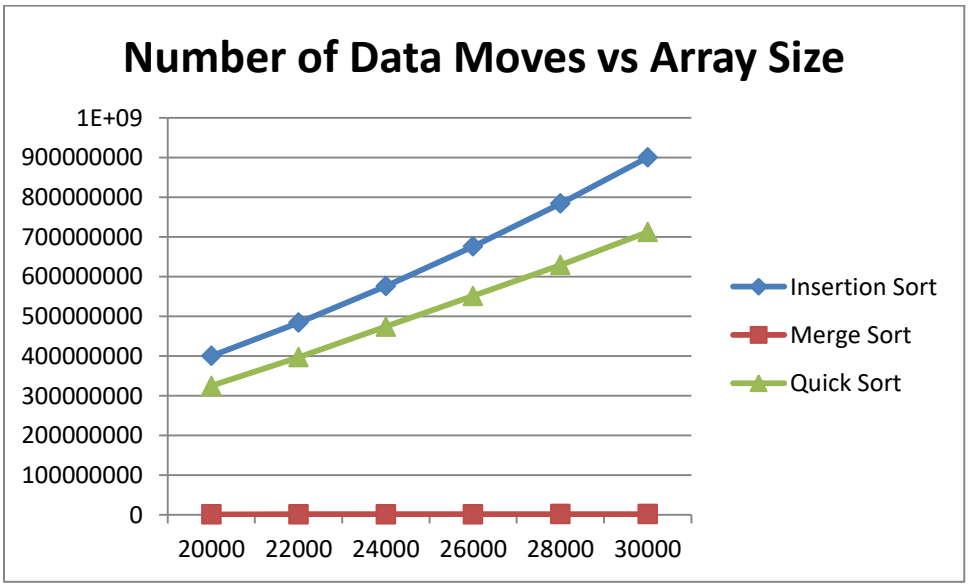
Array Size	Insertion Sort	Merge Sort	Quick Sort
20000	400018196	992890	324605585
22000	484019772	1100874	396991485
24000	576020944	1211706	474047030
26000	676022644	1320778	551063069
28000	784024912	1431482	629218774
30000	900027662	1543466	712231629

Kübra Okumuş
21600980



Number of Data Moves vs Array Size

Array Size	Insertion Sort	Merge Sort	Quick Sort
20000	400038194	1308143	324676856
22000	484041770	1453135	397071122
24000	576044942	1599551	474135347
26000	676048642	1745087	551161236
28000	784052910	1891439	629322651
30000	900057660	2038431	712340474

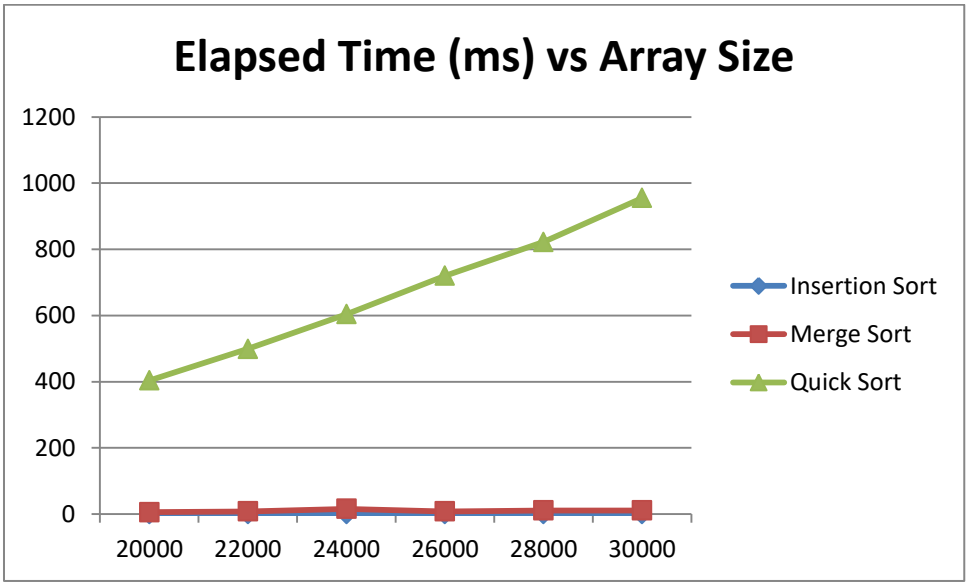


Kübra Okumuş
21600980

Experiment with Integers That are Sorted in Ascending Order

Elapsed Time (ms) vs Array Size

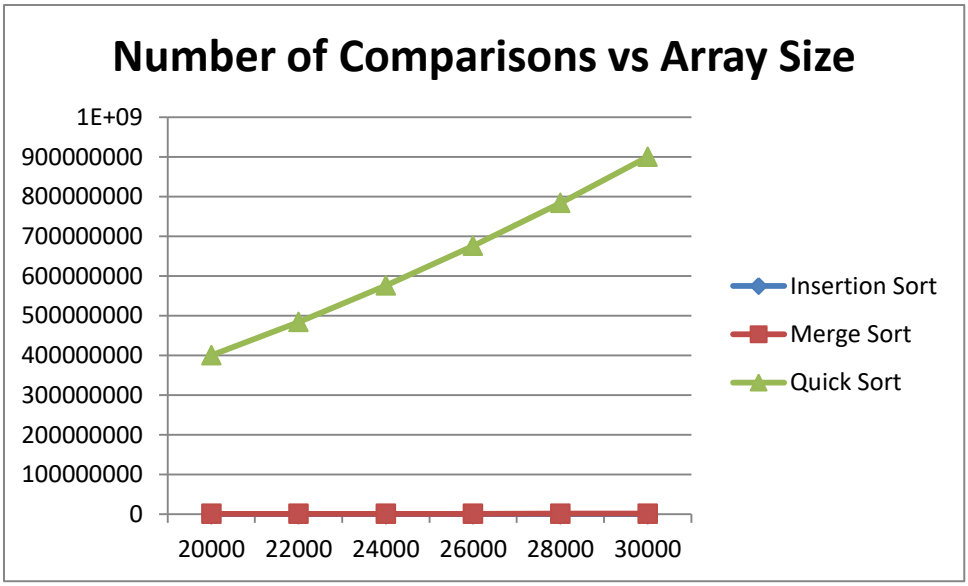
Array Size	Insertion Sort	Merge Sort	Quick Sort
20000	0.113	6	404
22000	0.128	8	499
24000	0.143	16	604
26000	0.152	8	720
28000	0.166	11	822
30000	0.181	11	955



Number of Key Comparisons vs Array Size

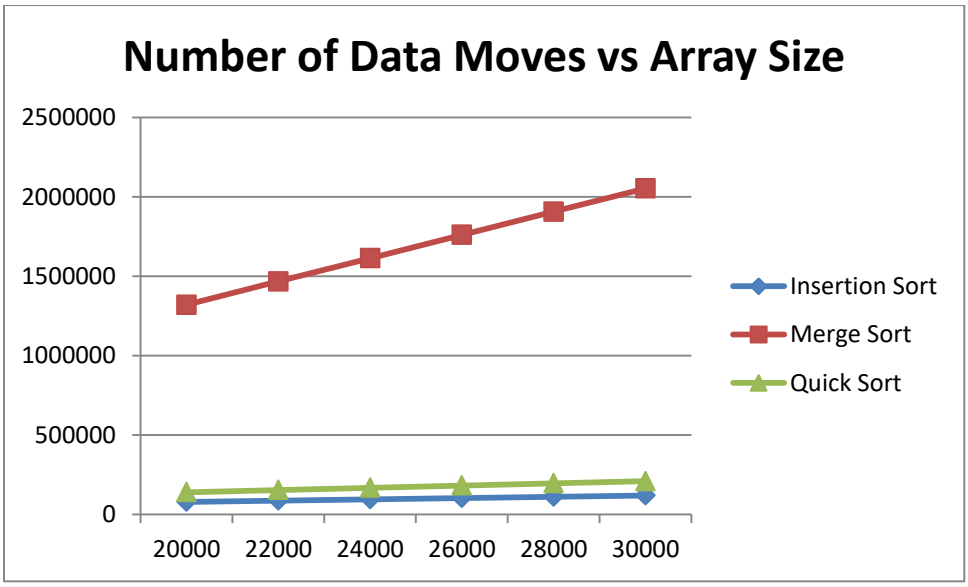
Array Size	Insertion Sort	Merge Sort	Quick Sort
20000	59998	1017140	400039998
22000	65998	1129026	484043998
24000	71998	1238578	576047998
26000	77998	1350802	676051998
28000	83998	1461626	784055998
30000	89998	1571478	900059998

Kübra Okumuş
21600980



Number of Data Moves vs Array Size

Array Size	Insertion Sort	Merge Sort	Quick Sort
20000	79996	1320268	139993
22000	87996	1467211	153993
24000	95996	1612987	167993
26000	103996	1760099	181993
28000	111996	1906511	195993
30000	119996	2052437	209993



Kübra Okumuş
21600980

Question 4

I used six different input sizes which are 20000, 22000, 24000, 26000, 28000, and 30000. I got error greater than 32000 array sizes.

Experiment with random integers, resulted that insertion sort algorithm takes longer time to sort the array compared to the merge sort and quick sort. Moreover, quick sort and the merge sort are more efficient ones with respect to the time elapsed, number of comparisons and number of moves. All of the three graphs look similar in this part and the lines are coherent with the theoretical results which should be $\mathbf{n^2}$ at insertion sort; **nlogn** at both merge sort and quick sort.

In the second part, the arrays with integers that are sorted in descending order are considered. The graphs show that insertion sort and quick sort take a long time to sort the reversed sorted data, while merge sort is faster. Moreover, the graphs of the number of key comparisons and the number of data moves are very similar to the elapsed time graph. The little differences are caused by choosing inputs with a small range. All these results are compatible with the theoretical ones. Because, the worst case occurs for the insertion sort and the quick sort which is proportional to $\mathbf{n^2}$**,** and merge sort lines are again grows with respect to **nlogn.**

In the third experiment, already sorted arrays are used. The graphical results of the elapsed time shows that the quick sort takes a long time, while the insertion sort is the fastest among them. Because, this is the best case for insertion sort and its time complexity is proportional to **n,** and the worst case occurs for the quick sort which is proportional to $\mathbf{n^2}$. In addition, merge sort shows the time complexity of **nlogn**. The graph of number of key comparisons is also very similar to the elapsed time graph. However, the graph of the number of data moves differs from the other two. It can be seen that insertion sort and quick sort have lesser data moves than merge sort in this case. The reason is that the number of data moves in insertion sort and quick sort are smaller compared to the other cases, while the merge sort has similar number of data moves in three cases. Because merge sort algorithm always divide the array into two until the one input size arrays are created, then it merge them. So the places of the numbers are not significantly affecting the number of data moves in merge sort. To sum up, all of these results are compatible with the theoretical results.