/**
* Title : Homework 1
* Author : Munib Emre Sevilgen
* ID: 21602416
* Section : 1
* Assignment : 1
* Description : Part a, b, c, and d of the Question 1, the screenshot for the Question 2, and Question 3
*/

Question 1:

(a)

$f(n) = 100n^3 + 8n^2 + 4n$ is $O(n^4)$ by the Big-O definition, if $f(n) \leq cn^4$ for some $n \geq n_0$. If $100n^3 + 8n^2 + 4n \leq cn^4$ then $\frac{100}{n} + \frac{8}{n^2} + \frac{4}{n^3} \leq c$. Therefore, this condition holds for $n \geq n_0 = 1$ and $c \geq 112 = (100 + 8 + 4)$.

(b)

$T(n) = 8T(n/2) + n^3$ where $T(n) = 1$ for all $n \leq 100$

$T(n) = n^3 + 8T(n/2)$

$$= n^3 + 8(8T\left(\frac{n}{2^2}\right) + (\frac{n}{2})^3)$$

$$= n^3 + n^3 + 8^2T\left(\frac{n}{2^2}\right)$$

$$= n^3 + n^3 + 8^2(8T\left(\frac{n}{2^3}\right) + (\frac{n}{2^2})^3)$$

$$= n^3 + n^3 + n^3 + 8^3T\left(\frac{n}{2^3}\right)$$

$$= n^3 + n^3 + n^3 + 8^3(8T\left(\frac{n}{2^4}\right) + (\frac{n}{2^3})^3)$$

$$= n^3 + n^3 + n^3 + n^3 + 8^4T\left(\frac{n}{2^4}\right)$$

$$= n^3 + n^3 + n^3 + n^3 + 8^4(8T\left(\frac{n}{2^5}\right) + (\frac{n}{2^4})^3)$$

$$= \cdots$$

$$= n^3 + n^3 + n^3 + n^3 + n^3 + \cdots + T(\frac{n}{2^x})$$ This is the last step that $\frac{n}{2^x} \leq 100$

and $x \leq log_2 \frac{n}{100}$

$$= n^3 + n^3 + n^3 + n^3 + n^3 + \cdots + 1 = xn^3 + 1 \leq (log_2 \frac{n}{100})(n^3) + 1$$

Therefore $T(n)$ is $O(n^3 logn)$

(c)

```
for (i = n; i > 0; i /= 2) { // This loop is $O(logn)$ because it turns $logn$ times.
        for (j = 1; j < n; j++) { // This loop is $O(n)$ because it turns $n$ times.
                for (k = 1; k < n; k += 2) { // This loop is $O(n)$ because it turns $n/2$ times.
                        sum += (i + j * k); } } }
```

Therefore, the time complexity of the code is $O(n) * O(n) * O(logn) = O(n^2 logn)$

(d) Sort the array [16, 6, 39, 21, 10, 21, 13, 7, 28, 19] in ascending with selection and insertion sort.

Selection sort:

Note: "|" separates the sorted and unsorted part.

Initial array: [16, 6, 39, 21, 10, 21, 13, 7, 28, 19]

After 1st swap: [16, 6, **19**, 21, 10, 21, 13, 7, 28 | **39**]

After 2nd swap: [16, 6, 19, **7**, 10, 21, 13 | **21**, 28, 39]

After 3rd swap: [16, 6, 19, 7, 10, **13** | **21**, 21, 28, 39]

After 4th swap: [16, 6, **13**, 7, 10 | **19**, 21, 21, 28, 39]

After 5th swap: [**10**, 6, 13, 7 | **16**, 19, 21, 21, 28, 39]

After 6th swap: [10, 6, **7** | **13**, 16, 19, 21, 21, 28, 39]

After 7th swap: [**7**, 6 | **10**, 13, 16, 19, 21, 21, 28, 39]

After 8th swap: [**6** | **7**, 10, 13, 16, 19, 21, 21, 28, 39]

Sorted array: [6, 7, 10, 13, 16, 19, 21, 21, 28, 39]

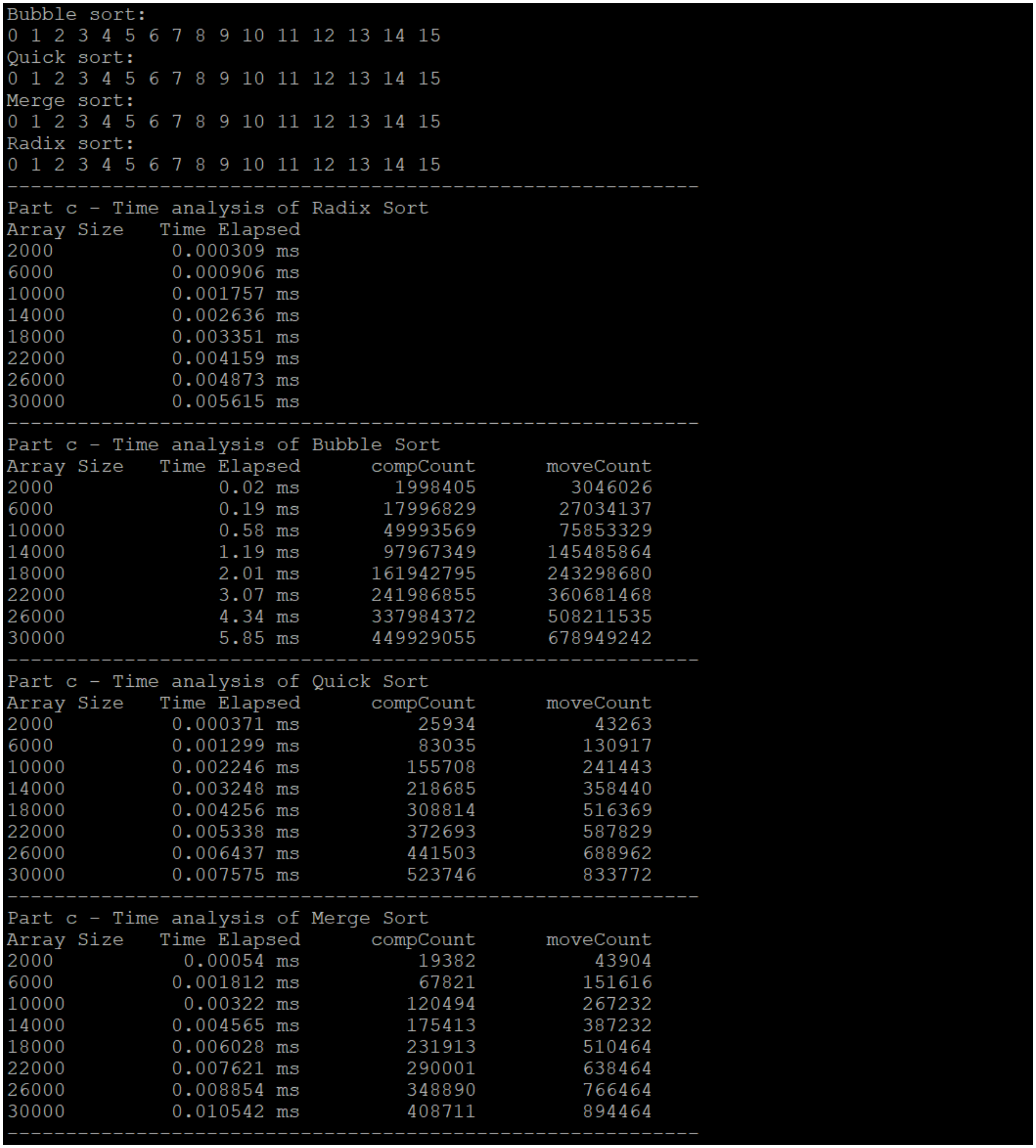
Insertion sort:

Note: "|" separates the sorted and unsorted part.

Initial array:	[16 | **6**, 39, 21, 10, 21, 13, 7, 28, 19]	Copy 6
	[16, **16** | 39, 21, 10, 21, 13, 7, 28, 19]	Shift 16
	[**6**, 16 | **39** 21, 10, 21, 13, 7, 28, 19]	Insert 6; copy 39, insert 39 on top of itself
	[6, 16, 39 | **21**, 10, 21, 13, 7, 28, 19]	Copy 21
	[6, 16, 39, **39** | 10, 21, 13, 7, 28, 19]	Shift 39
	[6, 16, **21**, 39 | **10**, 21, 13, 7, 28, 19]	Insert 21; copy 10
	[6, 16, **16**, **21**, **39** | 21, 13, 7, 28, 19]	Shift 39, 21, 16

	[6, **10**, 16, 21, 39 | **21**, 13, 7, 28, 19]	Insert 10; copy 21
	[6, 10, 16, 21, 39, **39** | 13, 7, 28, 19]	Shift 39
	[6, 10, 16, 21, **21**, 39 | **13**, 7, 28, 19]	Insert 21; copy 13
	[6, 16, **16**, **21**, **21**, **39** | 13, 7, 28, 19]	Shift 39, 21, 21, 16
	[6, **13**, 16, 16, 21, 21, 39 | **7**, 28, 19]	Insert 13; copy 7
	[6, 13, **13**, **16**, **16**, **21**, **21**, **39** | 28, 19]	Shift 39, 21, 21, 16, 16, 13
	[6, **7**, 13, 16, 16, 21, 21, 39 | **28**, 19]	Insert 7; copy 28
	[6, 7, 13, 16, 16, 21, 21, 39, **39** | 19]	Shift 39
	[6, 7, 13, 16, 16, 21, 21, **28**, 39 | **19**]	Insert 28; copy 19
	[6, 7, 13, 16, 16, 21, **21**, **21**, **28**, **39** |]	Shift 39, 28, 21, 21
	[6, 7, 13, 16, 16, **19**, 21, 21, 28, 39 |]	Insert 19
Sorted array:	[6, 7, 13, 16, 16, 19, 21, 21, 28, 39]	

Question 2:



```
Bubble sort:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Quick sort:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Merge sort:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Radix sort:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
-----------------------------------------------------------
Part c - Time analysis of Radix Sort
Array Size   Time Elapsed
2000          0.000309 ms
6000          0.000906 ms
10000         0.001757 ms
14000         0.002636 ms
18000         0.003351 ms
22000         0.004159 ms
26000         0.004873 ms
30000         0.005615 ms
-----------------------------------------------------------
Part c - Time analysis of Bubble Sort
Array Size   Time Elapsed      compCount      moveCount
2000              0.02 ms        1998405        3046026
6000              0.19 ms       17996829       27034137
10000             0.58 ms       49993569       75853329
14000             1.19 ms       97967349      145485864
18000             2.01 ms      161942795      243298680
22000             3.07 ms      241986855      360681468
26000             4.34 ms      337984372      508211535
30000             5.85 ms      449929055      678949242
-----------------------------------------------------------
Part c - Time analysis of Quick Sort
Array Size   Time Elapsed      compCount      moveCount
2000          0.000371 ms          25934          43263
6000          0.001299 ms          83035         130917
10000         0.002246 ms         155708         241443
14000         0.003248 ms         218685         358440
18000         0.004256 ms         308814         516369
22000         0.005338 ms         372693         587829
26000         0.006437 ms         441503         688962
30000         0.007575 ms         523746         833772
-----------------------------------------------------------
Part c - Time analysis of Merge Sort
Array Size   Time Elapsed      compCount      moveCount
2000           0.00054 ms          19382          43904
6000          0.001812 ms          67821         151616
10000          0.00322 ms         120494         267232
14000         0.004565 ms         175413         387232
18000         0.006028 ms         231913         510464
22000         0.007621 ms         290001         638464
26000         0.008854 ms         348890         766464
30000         0.010542 ms         408711         894464
-----------------------------------------------------------
```

Question 3:

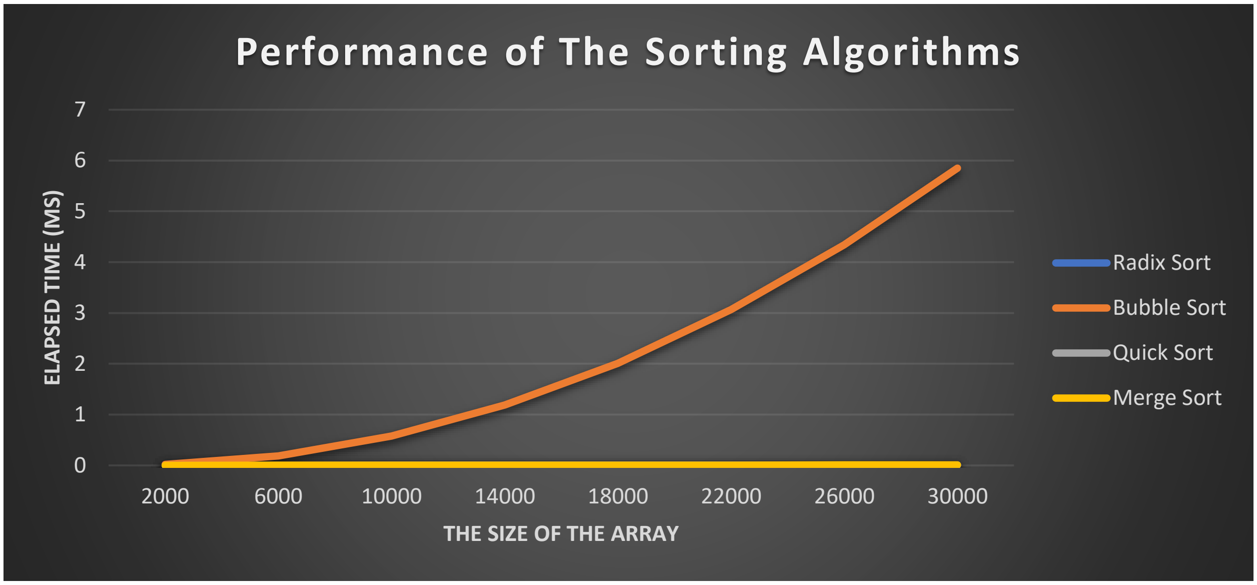


Figure 1: Performance of the sorting algorithms with the Bubble Sort

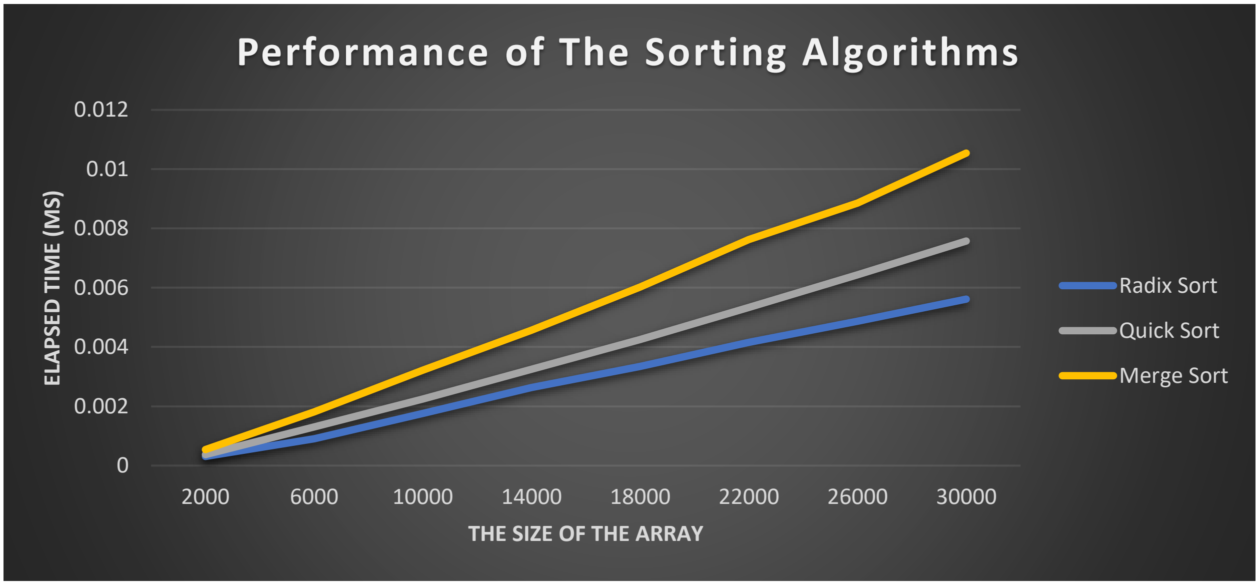


Figure 2: Performance of the sorting algorithms without the Bubble Sort

Algorithm	**Time Complexity**		
	Best	**Average**	**Worst**
Radix Sort	$\Omega(nk)$	$\theta(nk)$	$O(nk)$
Bubble Sort	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$
Quick Sort	$\Omega(nlog(n))$	$\theta(nlog(n))$	$O(n^2)$
Merge Sort	$\Omega(nlog(n))$	$\theta(nlog(n))$	$O(nlog(n))$

Table 1: Time Complexities of the Sorting Algorithms

By considering the time complexity of the Bubble Sort at the *Table 1,* we can say that the empirical result and the theoretical result are same. The elapsed times of the Bubble Sort is very large than the other sort algorithms in *Figure 1*. When we look at the other algorithms, the times of the Merge and the Quick Sort are larger than the Radix Sort's in *Figure 2* as expected, because the numbers in the array we used in these algorithms has not much digits. Therefore, the Radix Sort has an advantage because its time complexity is $O(nk)$ where the k is the digit number. When we consider the other two algorithms, their average case time complexities are $O(nlog(n))$. However, their worst-case time complexities are different. Quick Sort has $O(n^2)$ worst-case time complexity. By contrast with the complexities, the Quick Sort algorithm's elapsed times are smaller than the Merge Sort's. By looking the compare and move counts at the screenshot of Question 2, we can see that the move count of the Quick Sort is smaller than the move count of the Merge Sort. Although the Quick Sort has more compare count than the Merge Sort, the cost of the move can be more than the compare cost. Therefore, the Quick Sort is quicker than the Merge Sort. If the array has descending order, then this is the worst-case of the Bubble Sort. Its elapsed times would be larger. The Quick Sort also effected in the same way because we choose the pivot as the first element of the partition every time. Then, the algorithm calls the partition function for n-1 elements every time. However, the Merge Sort wouldn't be affected from this case. Because it does the merge process all the time independent from the data. The Radix Sort would also be affected in a bad way. It must move every element to its position.