Author: Khasmamad Shabanovi

ID: 21701333

Section: 2

Assignment: 1

Question 1

(a) $f(n) = 100n^3 + 8n^2 + 4n$
If we take $c = 1000$ and $n_0 = 1$, then $f(n) \leq cn^4$ for all $n \geq n_0$.2
Thus, $f(n) = O(n^4)$.

(b) $T(n) = 8T\left(\frac{n}{2}\right) + n^3$ and $T(n) = 1$, when $n \leq 100$

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

$$= 8^k T\left(\frac{n}{2^k}\right) + \sum_{i=0}^{k-1} 8^i \left(\frac{n}{2^i}\right)^3 = 8^k T\left(\frac{n}{2^k}\right) + n^3 \sum_{i=0}^{k-1} \left(\frac{8}{2^3}\right)^i = 8^k T\left(\frac{n}{2^k}\right) + n^3 k$$

$$= 8^{\log_2 \frac{n}{100}} T(100) + n^3 \log_2 \frac{n}{100} = \left(\frac{n}{100}\right)^3 + n^3 \log_2 \frac{n}{100} = O(n^3 \log_2 n)$$

(c)

	Cost:
for (i = n; i > 0; i /=2) {	$1 + (\log_2 n + 2) + 2(\log_2 n + 1)$
for (j = 1; j < n; j++) {	$(\log_2 n + 1)(1 + n + 2(n-1))$
for (k = 1; k < n; k += 2) {	$n(\log_2 n + 1)\left(1 + \left(\left\lceil\frac{n-1}{2}\right\rceil + 1\right) + 2\left\lceil\frac{n-1}{2}\right\rceil\right)$
sum += (i + j * k) ; }}}	$n(\log_2 n + 1)\left(\left\lceil\frac{n-1}{2}\right\rceil\right) \cdot 4$

Total cost: $O(n^2 \log_2 n)$

(d) Selection sort:
Unsorted | Sorted
Bold numbers indicate the (first occurring) largest among the unsorted
Red numbers indicate swapped element(s)

Initial array:	[16, 6, **39**, 21, 10, 21, 13, 7, 28, 9|]
After 1st swap:	[16, 6, 9, 21, 10, 21, 13, 7, **28,**| 39]
After 2nd swap:	[16, 6, 9, **21**, 10, 21, 13, 7,| 28, 39]
After 3rd swap:	[16, 6, 9, 7, 10, **21**, 13,| 21, 28, 39]
After 4th swap:	[**16**, 6, 9, 7, 10, 13,| 21, 21, 28, 39]
After 5th swap:	[**13**, 6, 9, 7, 10,| 16, 21, 21, 28, 39]
After 5th swap:	[**10**, 6, 9, 7,| 13, 16, 21, 21, 28, 39]
After 6th swap:	[7, 6, **9**,| 10, 13, 16, 21, 21, 28, 39]
After 7th swap:	[**7**, 6,| 9, 10, 13, 16, 21, 21, 28, 39]

After 8th swap: [6,| 7, 9, 10, 13, 16, 21, 21, 28, 39]

Insertion Sort:

Sorted | Unsorted

Bold numbers indicated the key value (value to be inserted in the sorted list)

Blue numbers are less than or equal to, red numbers are greater than key (so that, key will inserted in-between red and blue numbers)

Initial array: [16,| **6**, 39, 21, 10, 21, 13, 7, 28, 9]
After 1st pass: [6, 16,| **39**, 21, 10, 21, 13, 7, 28, 9]
After 2nd pass: [6, 16, 39,| **21**, 10, 21, 13, 7, 28, 9]
After 3rd pass: [6, 16, 21, 39,| **10**, 21, 13, 7, 28, 9]
After 4th pass: [6, 10, 16, 21, 39,| **21**, 13, 7, 28, 9]
After 5th pass: [6, 10, 16, 21, 21, 39,| **13**, 7, 28, 9]
After 6th pass: [6, 10, 13, 16, 21, 21, 39,| **7**, 28, 9]
After 7th pass: [6, 7, 10, 13, 16, 21, 21, 39,| **28**, 9]
After 8th pass: [6, 7, 10, 13, 16, 21, 21, 28, 39,| **9**]
After 9th pass: [6, 7, 9, 10, 13, 16, 21, 21, 28, 39|]

Question 2:

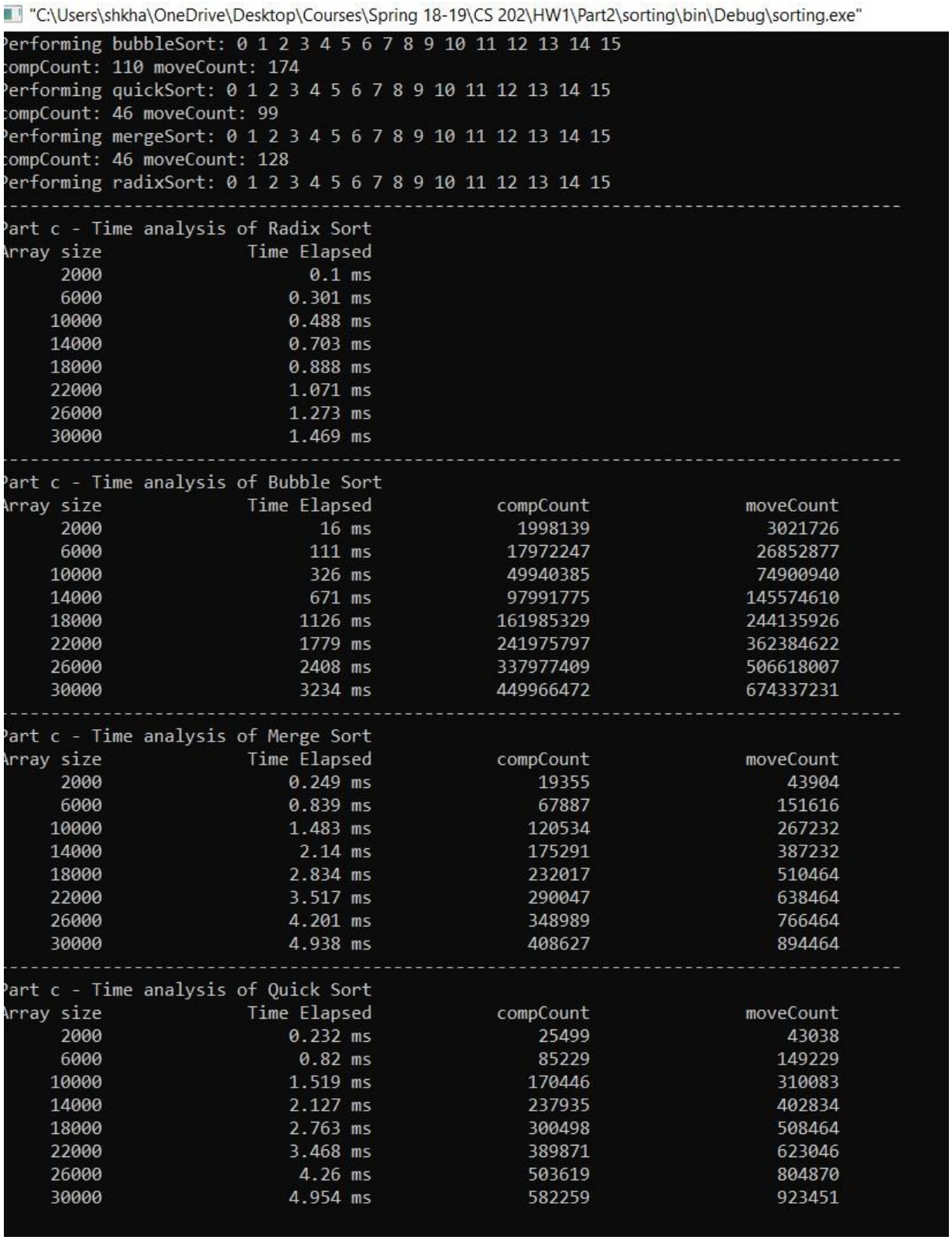
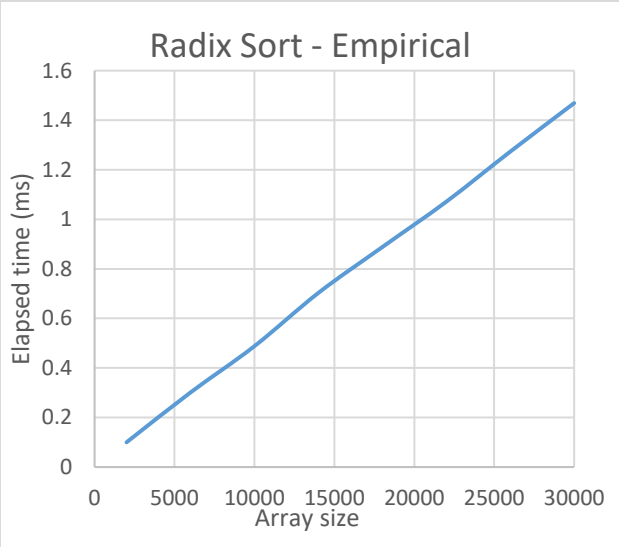
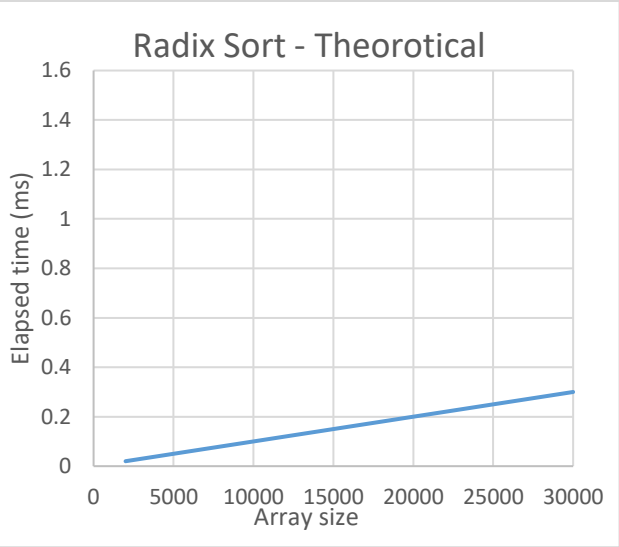
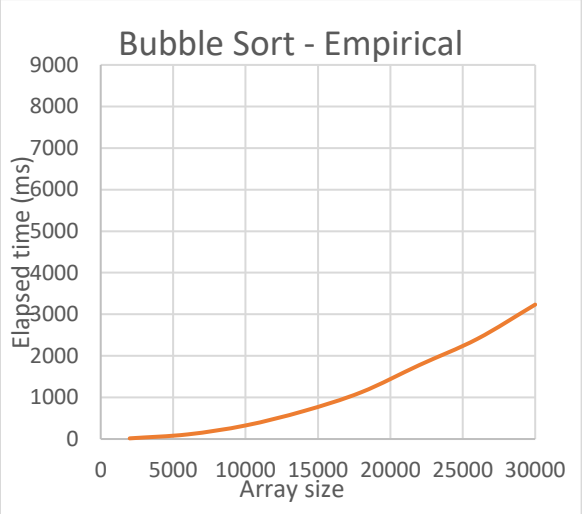


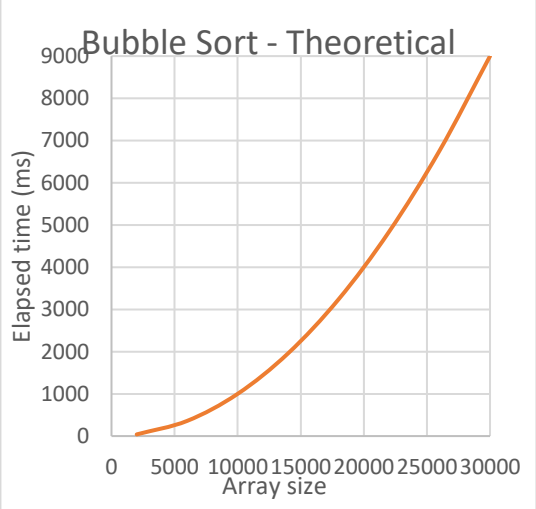
Figure 1 Console output of main.cpp

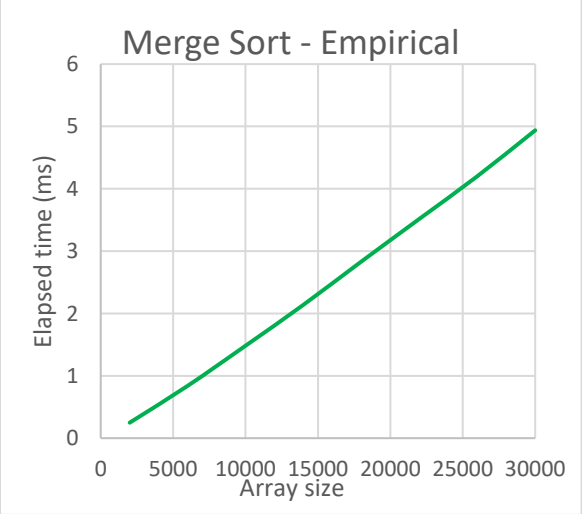
Question 3:

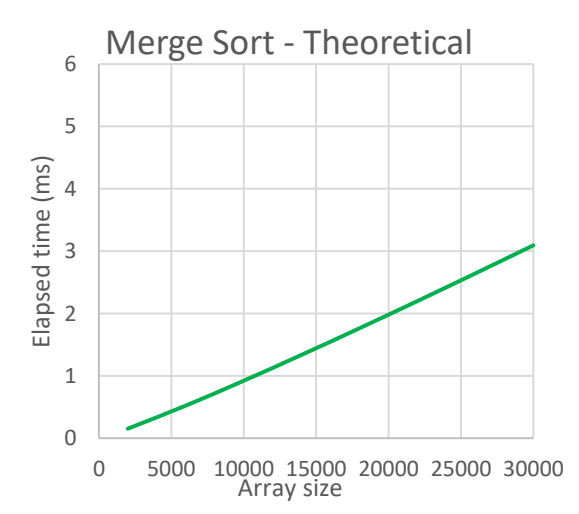

Radix Sort - Empirical
Elapsed time (ms)
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0 5000 10000 15000 20000 25000 30000
Array size

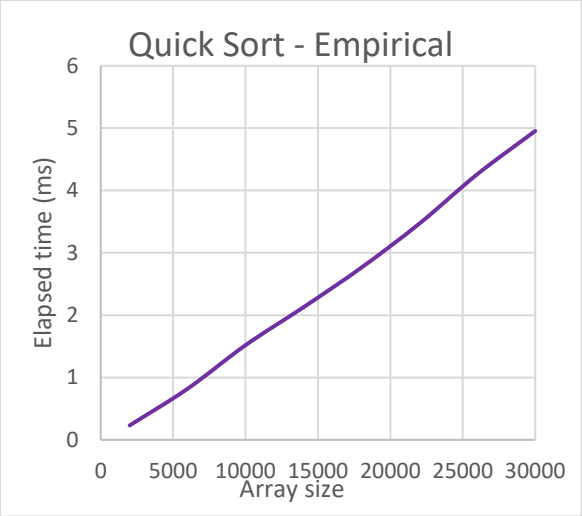

Radix Sort - Theorotical
Elapsed time (ms)
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0 5000 10000 15000 20000 25000 30000
Array size

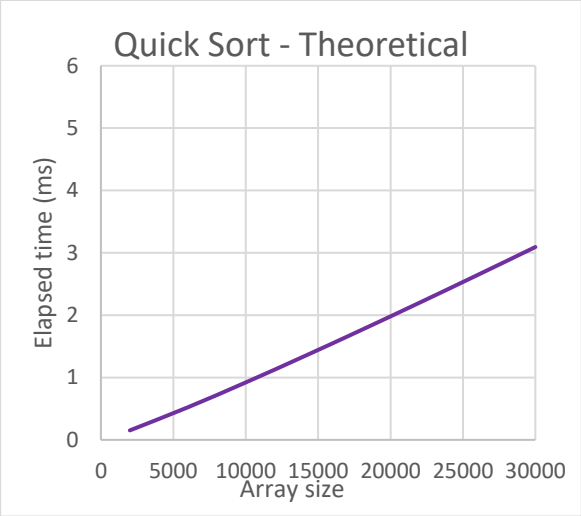

Bubble Sort - Empirical
Elapsed time (ms)
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0 5000 10000 15000 20000 25000 30000
Array size


Bubble Sort - Theoretical
Elapsed time (ms)
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0 5000 10000 15000 20000 25000 30000
Array size


Merge Sort - Empirical
Elapsed time (ms)
6
5
4
3
2
1
0
0 5000 10000 15000 20000 25000 30000
Array size


Merge Sort - Theoretical
Elapsed time (ms)
6
5
4
3
2
1
0
0 5000 10000 15000 20000 25000 30000
Array size





Observations:

Above graphs represent the running time complexities of Radix Sort, Bubble Sort, Merge Sort, and Quick Sort algorithms for differently sized arrays. Because of the technical difficulties (merge sort and quick sort overlapping and bubble sort increasing way faster than the rest) of plotting the results in the same graph, they are graphed separately. For theoretical results, it's assumed that the computer performs 10^8 operations per second. Moreover, to get more accurate results, the experiment is performed M = 1000 times and average is calculated (this might cause the program to run much slower than expected).

Although the empirical and theoretical results are similar in terms of growth rates, there are slight differences. Theoretically, radix sort performs in O(n) if the length of the largest number is 10. However, in the empirical case the number of digits in the largest number can be different, which contributes to the difference between theoretical and empirical results. Another reason for the very differences can be the fact that we eliminate constants and unit differences in time-complexity, ignore the time spent for memory allocation\deallocations and ignore the differences in speeds of different machines when performing theoretical analysis. In the case of bubble sort, however, the reason why it's empirically faster might be because we use the flag "sorted."

If the array was sorted in descending order, the time complexity for radix sort, merge sort, and bubble sort would not change. Nevertheless, number of swaps for merge sort and bubble sort would increase significantly compared to the average case. In the case of quick sort, time complexity would become $O(n^2)$, since the array would be partitioned into sizes of 1 and n-1 every time.