

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  $n \geq n_0$ .

Thus,  $f(n) = O(n^4)$ .

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

$$\begin{aligned} T(n) &= 8T\left(\frac{n}{2}\right) + n^3 = 8\left(8T\left(\frac{n}{4}\right) + \left(\frac{n}{2}\right)^3\right) + n^3 = 8\left(8\left(8T\left(\frac{n}{8}\right) + \left(\frac{n}{4}\right)^3\right) + \left(\frac{n}{2}\right)^3\right) + n^3 \\ &= 8^k T\left(\frac{n}{2^k}\right) + \sum_{i=0}^{k-1} 8^i \left(\frac{n}{2^i}\right)^3 = 8^k T\left(\frac{n}{2^k}\right) + n^3 \sum_{i=0}^{k-1} \left(\frac{8}{2^3}\right)^i = 8^k T\left(\frac{n}{2^k}\right) + n^3 k \\ &= 8^{\log_2 \frac{n}{100}} T(100) + n^3 \log_2 \frac{n}{100} = \left(\frac{n}{100}\right)^3 + n^3 \log_2 \frac{n}{100} = O(n^3 \log_2 n) \end{aligned}$$

(c)

```
for (i = n; i > 0; i /= 2) {  
    for (j = 1; j < n; j++) {  
        for (k = 1; k < n; k += 2) {  
            sum += (i + j * k);  
        }  
    }  
}
```

Cost:

$$\begin{aligned} &1 + (\log_2 n + 2) + 2(\log_2 n + 1) \\ &(\log_2 n + 1)(1 + n + 2(n - 1)) \\ &n(\log_2 n + 1) \left(1 + \left\lceil \frac{n-1}{2} \right\rceil + 1\right) + 2 \left\lceil \frac{n-1}{2} \right\rceil \\ &n(\log_2 n + 1) \left(\left\lceil \frac{n-1}{2} \right\rceil\right) \cdot 4 \end{aligned}$$

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 1<sup>st</sup> swap: [16, 6, **9**, 21, 10, 21, 13, 7, **28**, **39**]

After 2<sup>nd</sup> swap: [16, 6, 9, **21**, 10, 21, 13, 7, **28**, 39]

After 3<sup>rd</sup> swap: [16, 6, 9, **7**, 10, **21**, 13, **21**, 28, 39]

After 4<sup>th</sup> swap: [**16**, 6, 9, 7, 10, **13**, **21**, 21, 28, 39]

After 5<sup>th</sup> swap: [**13**, 6, 9, 7, 10, **16**, 21, 21, 28, 39]

After 5<sup>th</sup> swap: [**10**, 6, 9, 7, **13**, 16, 21, 21, 28, 39]

After 6<sup>th</sup> swap: [**7**, 6, **9**, **10**, 13, 16, 21, 21, 28, 39]

After 7<sup>th</sup> swap: [**7**, 6, **9**, 10, 13, 16, 21, 21, 28, 39]

After 8<sup>th</sup> 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 1<sup>st</sup> pass: [6, 16, 39, 21, 10, 21, 13, 7, 28, 9]

After 2<sup>nd</sup> pass: [6, 16, 39, 21, 10, 21, 13, 7, 28, 9]

After 3<sup>rd</sup> pass: [6, 16, 21, 39, 10, 21, 13, 7, 28, 9]

After 4<sup>th</sup> pass: [6, 10, 16, 21, 39, 21, 13, 7, 28, 9]

After 5<sup>th</sup> pass: [6, 10, 16, 21, 21, 39, 13, 7, 28, 9]

After 6<sup>th</sup> pass: [6, 10, 13, 16, 21, 21, 39, 7, 28, 9]

After 7<sup>th</sup> pass: [6, 7, 10, 13, 16, 21, 21, 39, 28, 9]

After 8<sup>th</sup> pass: [6, 7, 10, 13, 16, 21, 21, 28, 39, 9]

After 9<sup>th</sup> pass: [6, 7, 9, 10, 13, 16, 21, 21, 28, 39]

## Question 2:

"C:\Users\shkha\OneDrive\Desktop\Courses\Spring 18-19\CS 202\HW1\Part2\sorting\bin\Debug\sorting.exe"

```
Performing bubbleSort: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
compCount: 110 moveCount: 174
Performing quickSort: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
compCount: 46 moveCount: 99
Performing mergeSort: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
compCount: 46 moveCount: 128
Performing radixSort: 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	0.1 ms
6000	0.301 ms
10000	0.488 ms
14000	0.703 ms
18000	0.888 ms
22000	1.071 ms
26000	1.273 ms
30000	1.469 ms

### Part c - Time analysis of Bubble Sort

Array size	Time Elapsed	compCount	moveCount
2000	16 ms	1998139	3021726
6000	111 ms	17972247	26852877
10000	326 ms	49940385	74900940
14000	671 ms	97991775	145574610
18000	1126 ms	161985329	244135926
22000	1779 ms	241975797	362384622
26000	2408 ms	337977409	506618007
30000	3234 ms	449966472	674337231

### Part c - Time analysis of Merge Sort

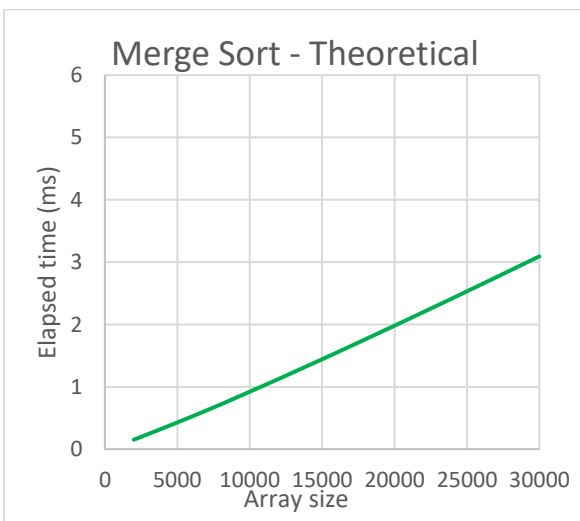
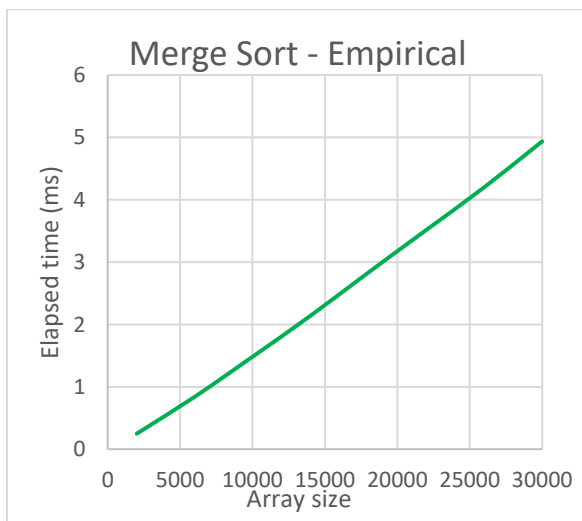
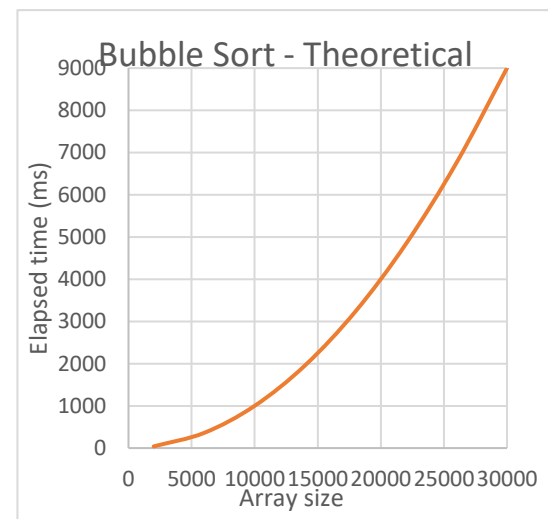
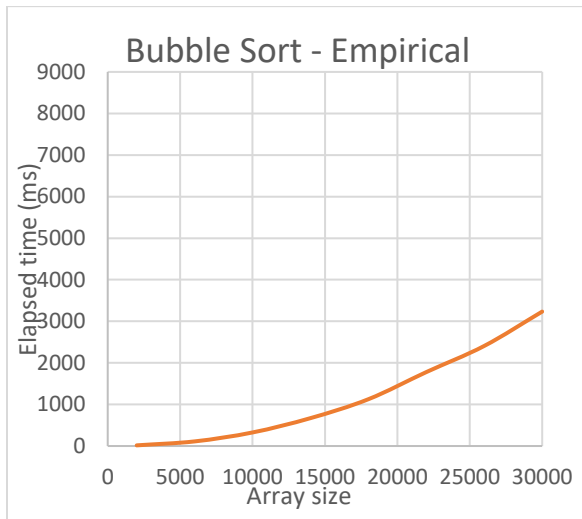
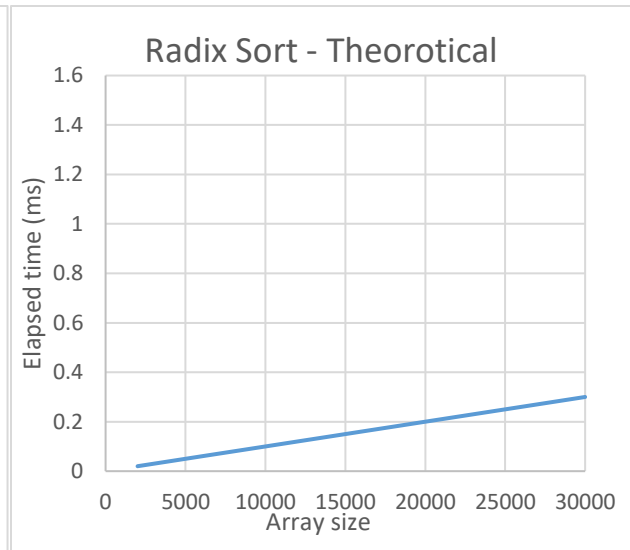
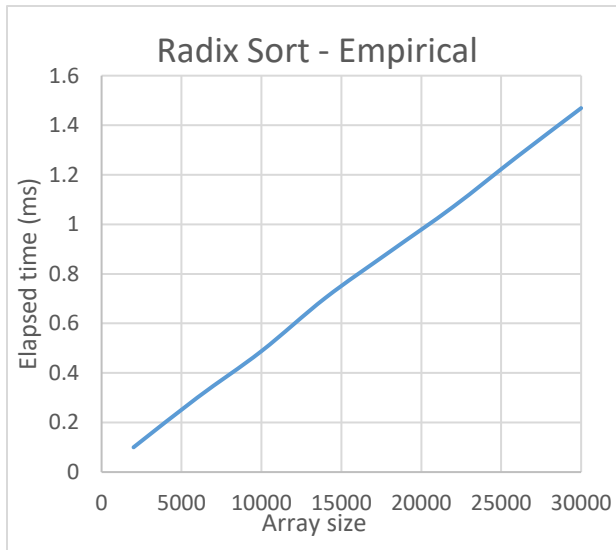
Array size	Time Elapsed	compCount	moveCount
2000	0.249 ms	19355	43904
6000	0.839 ms	67887	151616
10000	1.483 ms	120534	267232
14000	2.14 ms	175291	387232
18000	2.834 ms	232017	510464
22000	3.517 ms	290047	638464
26000	4.201 ms	348989	766464
30000	4.938 ms	408627	894464

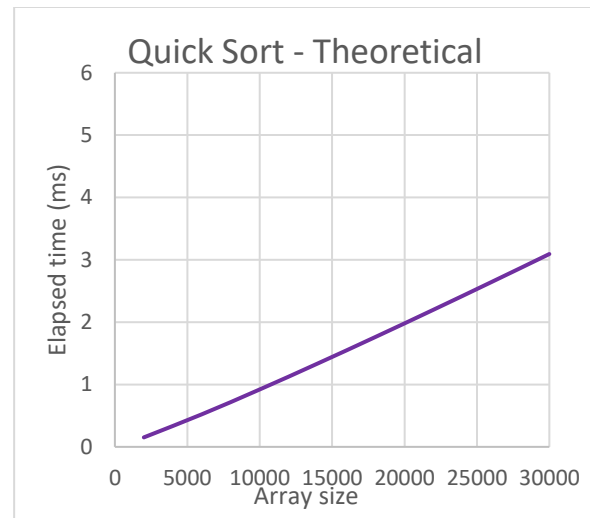
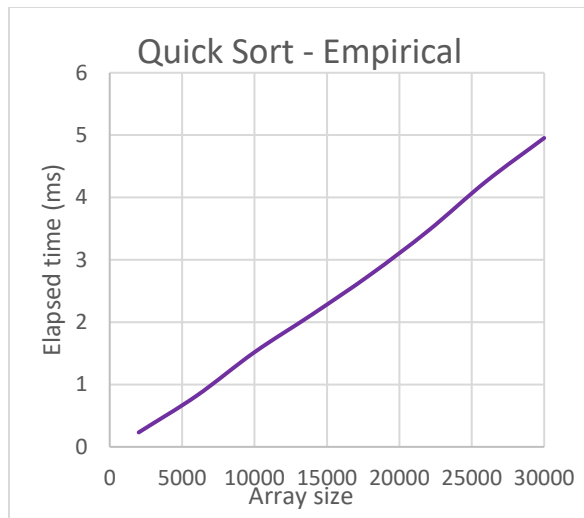
### Part c - Time analysis of Quick Sort

Array size	Time Elapsed	compCount	moveCount
2000	0.232 ms	25499	43038
6000	0.82 ms	85229	149229
10000	1.519 ms	170446	310083
14000	2.127 ms	237935	402834
18000	2.763 ms	300498	508464
22000	3.468 ms	389871	623046
26000	4.26 ms	503619	804870
30000	4.954 ms	582259	923451

Figure 1 Console output of main.cpp

### Question 3:





### 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.