Efe Beydoğan

21901548

CS202 Section: 1

HW1 Report

Question 1:

(a) To show that $5n^3+4n^2+10$ is $O(n^4)$, we need to find two positive constants c and n_0 such that $0 \leq 5n^3+4n^2+10 \leq cn^4$ for all $n \geq n_0$. If c = 5 and n = 2, it can be seen that $0 \leq 5n^3+4n^2+10 \leq 5n^4$ for all $n \geq 2$.

(b)

Tracing of insertion sort:

Initial array:

24	8	51	28	20	29	21	17	38	27

Copy 8, shift 24 to 2nd position and paste 8 into first position:

8	24	51	28	20	29	21	17	38	27

Copy 51, insert 51 on top of itself:

8	24	51	28	20	29	21	17	38	27

Copy 28, shift 51 into 4th position and insert 28 into 3rd position:

8	24	28	51	20	29	21	17	38	27

Copy 20, shift 24, 28 and 51, insert 20 into its place:

8	20	24	28	51	29	21	17	38	27

Copy 29, shift 51, insert 29 into its place:

8	20	24	28	29	51	21	17	38	27

Copy 21, shift 24, 28, 29, 51, insert 21 into its place:

8	20	21	24	28	29	51	17	38	27

Copy 17, shift 20, 21, 24, 28, 29, 51, insert 17 into its place:

8	17	20	21	24	28	29	51	38	27

Copy 38, shift 51, insert 38:

8	17	20	21	24	28	29	38	51	27

Copy 27, shift 28, 29, 38, 51, insert 27:

8	17	20	21	24	27	28	29	38	51

The array is sorted.

Tracing of bubble sort:

Initial array:

24	8	51	28	20	29	21	17	38	27

Pass 1:

24	8	51	28	20	29	21	17	38	27

8	24	51	28	20	29	21	17	38	27

8	24	51	28	20	29	21	17	38	27

8	24	28	51	20	29	21	17	38	27

8	24	28	20	51	29	21	17	38	27

8	24	28	20	29	51	21	17	38	27

8	24	28	20	29	21	51	17	38	27

8	24	28	20	29	21	17	51	38	27

8	24	28	20	29	21	17	38	51	27

8	24	28	20	29	21	17	38	27	51

Pass 2:

8	24	28	20	29	21	17	38	27	51

8	24	28	20	29	21	17	38	27	51

8	24	28	20	29	21	17	38	27	51

8	24	20	28	29	21	17	38	27	51

8	24	20	28	29	21	17	38	27	51

8	24	20	28	21	29	17	38	27	51

8	24	20	28	21	17	29	38	27	51

8	24	20	28	21	17	29	38	27	51

8	24	20	28	21	17	29	27	38	51

Pass 3:

8	24	20	28	21	17	29	27	38	51

8	24	20	28	21	17	29	27	38	51

8	20	24	28	21	17	29	27	38	51

8	20	24	28	21	17	29	27	38	51

8	20	24	21	28	17	29	27	38	51

8	20	24	21	17	28	29	27	38	51

8	20	24	21	17	28	29	27	38	51

8	20	24	21	17	28	27	29	38	51

Pass 4:

8	20	24	21	17	28	27	29	38	51

8	20	24	21	17	28	27	29	38	51

8	20	24	21	17	28	27	29	38	51

8	20	21	24	17	28	27	29	38	51

8	20	21	17	24	28	27	29	38	51

8	20	21	17	24	28	27	29	38	51

8	20	21	17	24	27	28	29	38	51

Pass 5:

8	20	21	17	24	27	28	29	38	51

8	20	21	17	24	27	28	29	38	51

8	20	21	17	24	27	28	29	38	51

8	20	17	21	24	27	28	29	38	51

8	20	17	21	24	27	28	29	38	51

8	20	17	21	24	27	28	29	38	51

Pass 6:

8	20	17	21	24	27	28	29	38	51

8	20	17	21	24	27	28	29	38	51

8	20	17	21	24	27	28	29	38	51

8	17	20	21	24	27	28	29	38	51

8	17	20	21	24	27	28	29	38	51

8	17	20	21	24	27	28	29	38	51

Pass 7:

8	17	20	21	24	27	28	29	38	51

8	17	20	21	24	27	28	29	38	51

8	17	20	21	24	27	28	29	38	51

8	17	20	21	24	27	28	29	38	51

7th pass will be the last pass since the array is sorted.

Question 2:

Screenshot of the console for part (c):

```
Selection sort:
Number of key comparisons: 120
Number of data moves: 45
Array after sorting: 3, 5, 6, 7, 8, 9, 11, 12, 12, 14, 14, 17, 18, 19, 20, 21

mergesort:
Number of key comparisons: 46
Number of data moves: 128
Array after sorting: 3, 5, 6, 7, 8, 9, 11, 12, 12, 14, 14, 17, 18, 19, 20, 21

quicksort:
Number of key comparisons: 45
Number of data moves: 102
Array after sorting: 3, 5, 6, 7, 8, 9, 11, 12, 12, 14, 14, 17, 18, 19, 20, 21

radixsort:
Array after sorting: 3, 5, 6, 7, 8, 9, 11, 12, 12, 14, 14, 17, 18, 19, 20, 21


Process returned 0 (0x0)   execution time : 0.232 s
Press any key to continue.
```

Screenshots of the console for part (d) performance analysis:

```
Analysis of Selection Sort with Random Arrays
Array Size          Elapsed time (ms)   compCount           moveCount
6000                38                  17997000            17997
10000               102                 49995000            29997
14000               195                 97993000            41997
18000               337                 161991000           53997
22000               476                 241989000           65997
26000               670                 337987000           77997
30000               917                 449985000           89997

Analysis of Selection Sort with Ascending Arrays
Array Size          Elapsed time (ms)   compCount           moveCount
6000                36                  17997000            17997
10000               117                 49995000            29997
14000               264                 97993000            41997
18000               323                 161991000           53997
22000               481                 241989000           65997
26000               750                 337987000           77997
30000               943                 449985000           89997
```

Analysis of Selection Sort with Descending Arrays

Array Size	Elapsed time (ms)	compCount	moveCount
6000	38	17997000	17997
10000	106	49995000	29997
14000	195	97993000	41997
18000	355	161991000	53997
22000	503	241989000	65997
26000	673	337987000	77997
30000	911	449985000	89997

Analysis of Merge Sort with Random Arrays

Array Size	Elapsed time (ms)	compCount	moveCount
6000	9	67846	151616
10000	5	120467	267232
14000	5	175302	387232
18000	6	232082	510464
22000	7	290009	638464
26000	8	349093	766464
30000	10	408587	894464

Analysis of Merge Sort with Ascending Arrays

Array Size	Elapsed time (ms)	compCount	moveCount
6000	2	39152	151616
10000	3	69008	267232
14000	4	99360	387232
18000	4	130592	510464
22000	6	165024	638464
26000	7	197072	766464
30000	8	227728	894464

Analysis of Merge Sort with Descending Arrays

Array Size	Elapsed time (ms)	compCount	moveCount
6000	2	36656	151616
10000	3	64608	267232
14000	4	94256	387232
18000	5	124640	510464
22000	6	154208	638464
26000	7	186160	766464
30000	8	219504	894464

Analysis of Quick Sort with Random Arrays

Array Size	Elapsed time (ms)	compCount	moveCount
6000	0	86544	142049
10000	1	158140	228780
14000	1	228797	375944
18000	1	305144	523456
22000	3	392433	648441
26000	3	442417	740623
30000	3	539100	973080

Analysis of Quick Sort with Ascending Arrays

Array Size	Elapsed Time(ms)	compCount	moveCount
6000	80	17997000	23996
10000	240	49995000	39996
14000	460	97993000	55996
18000	760	161991000	71996
22000	1140	241989000	87996
26000	1600	337987000	103996
30000	2120	449985000	119996

Analysis of Quick Sort with Descending Arrays

Array Size	Elapsed Time(ms)	compCount	moveCount
6000	120	17997000	27023996
10000	340	49995000	75039996
14000	660	97993000	147055996
18000	1100	161991000	243071996
22000	1640	241989000	363087996
26000	2290	337987000	507103996
30000	3050	449985000	675119996

Analysis of Radix Sort with Random Arrays

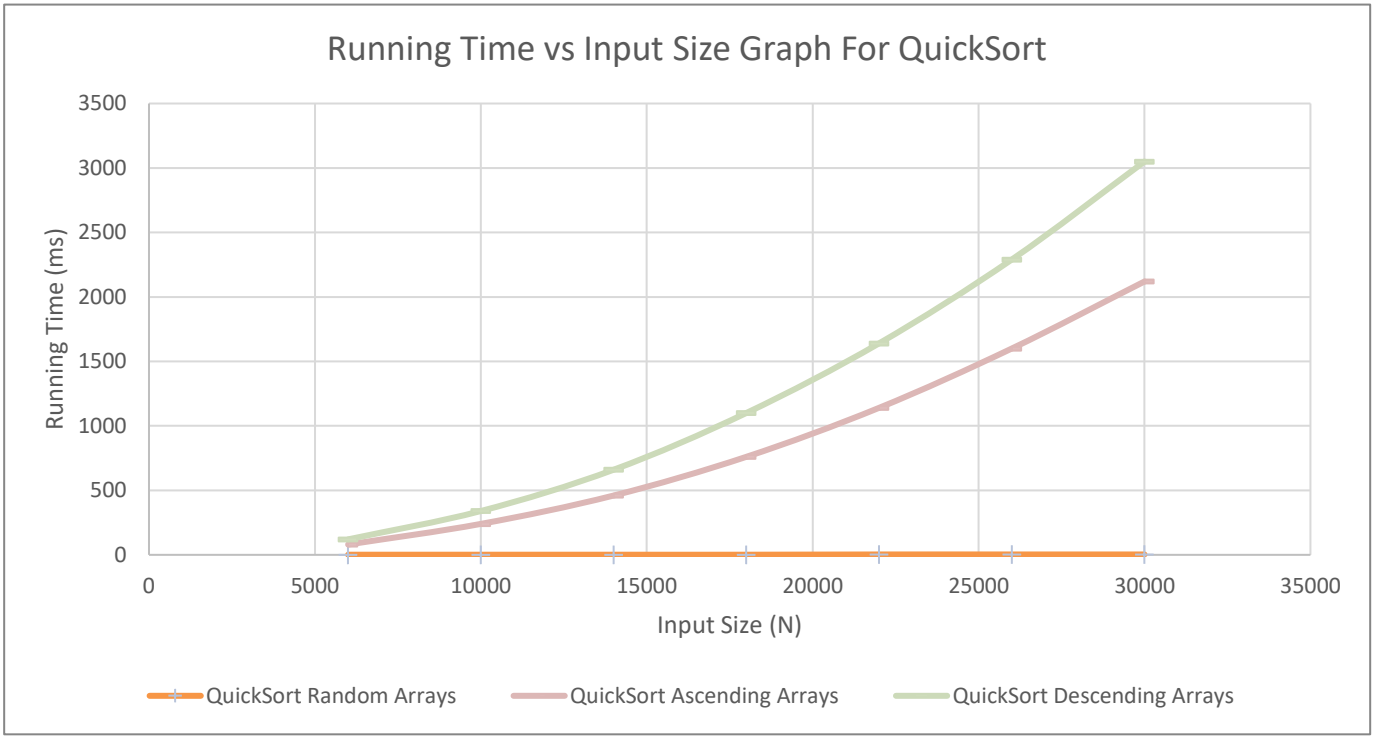
Array Size	Elapsed time (ms)
6000	1
10000	2
14000	2
18000	3
22000	4
26000	5
30000	6

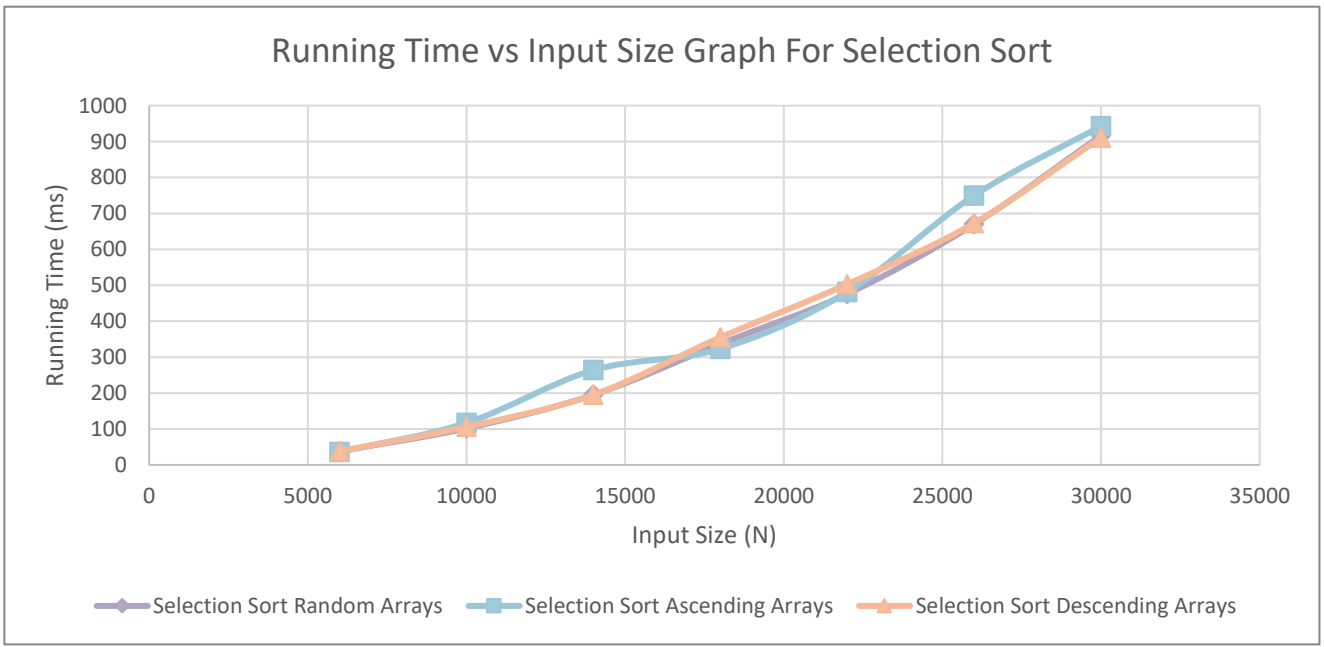
Analysis of Radix Sort with Ascending Arrays

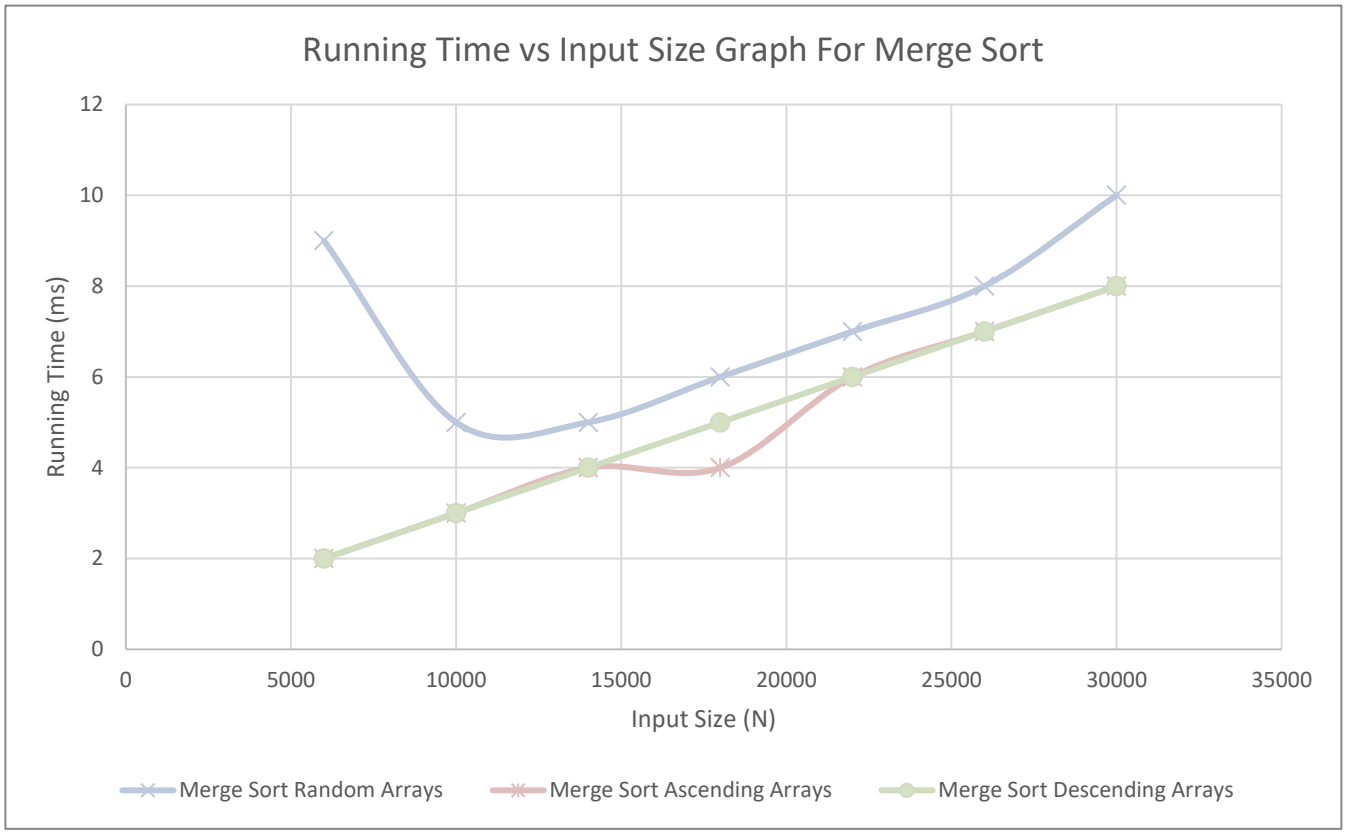
Array Size	Elapsed time (ms)
6000	2
10000	3
14000	5
18000	5
22000	7
26000	12
30000	11

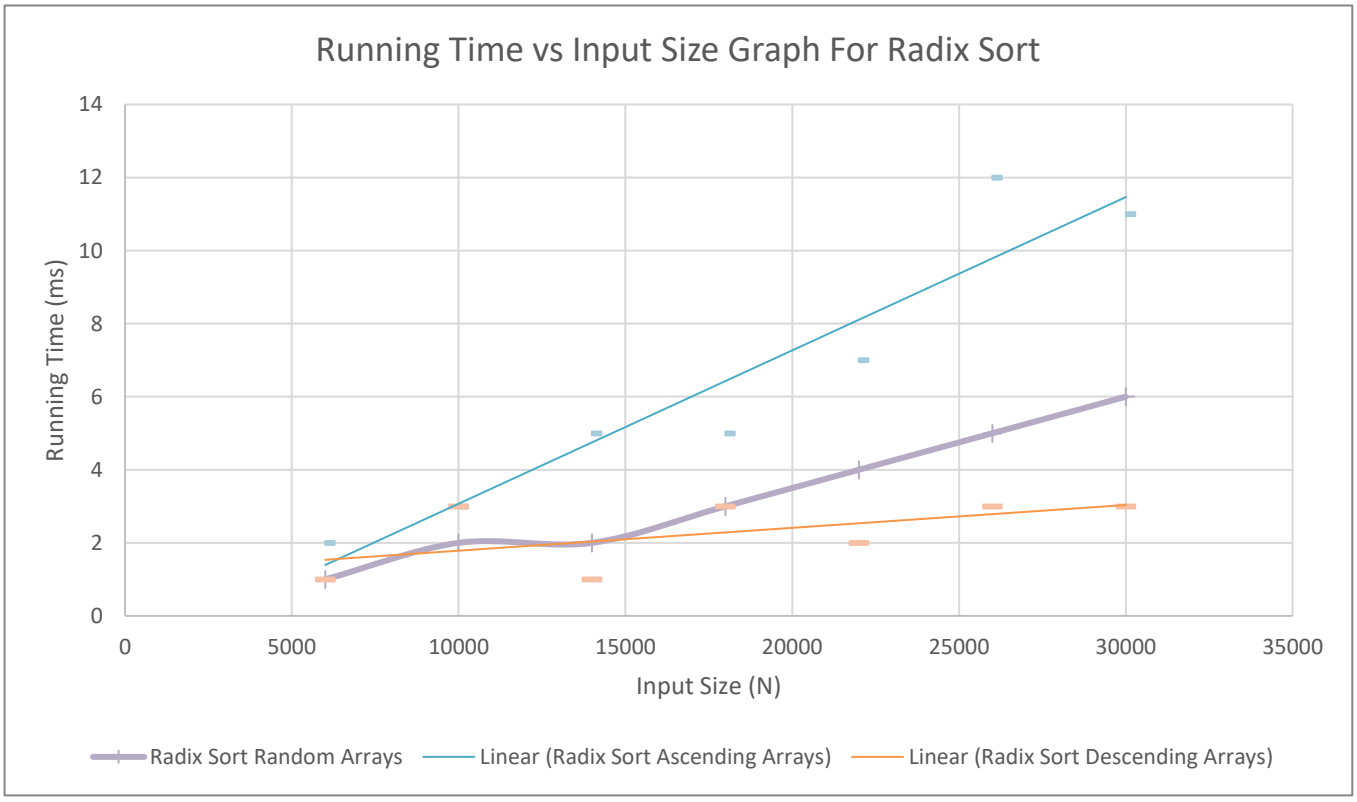
```
Analysis of Radix Sort with Descending Arrays
Array Size              Elapsed time (ms)
6000                    1
10000                   3
14000                   1
18000                   3
22000                   2
26000                   3
30000                   3
```

Question 3:






Running Time vs Input Size Graph For Merge Sort
Running Time (ms)
0
2
4
6
8
10
12
0
5000
10000
15000
20000
25000
30000
35000
Input Size (N)
Merge Sort Random Arrays
Merge Sort Ascending Arrays
Merge Sort Descending Arrays


Running Time vs Input Size Graph For Radix Sort
Running Time (ms)
0
2
4
6
8
10
12
14
0
5000
10000
15000
20000
25000
30000
35000
Input Size (N)
Radix Sort Random Arrays
Linear (Radix Sort Ascending Arrays)
Linear (Radix Sort Descending Arrays)

Selection sort is an algorithm which has O(n^2) time complexity and the results I've found support this. This can be seen by comparing the first and the last situations in the random array case for selection sort. For array size with 6000, the algorithm takes 38 ms to run and for size 30000, it takes 917 ms, so when the array size is multiplied by 5, the algorithm takes roughly 25 times longer. Merge sort is supposed to have O(nlogn) time complexity and the results that can be seen in my screenshots above show that merge sort running time varies in a way that shows it has O(nlogn) time, so my empirical results and the theoratical expectations match for this case. My results are consistent for quicksort, which is O(nlogn) and radix sort, which is O(n), as well. Any small changes or discrepancies between my empirical results and the theoretical ones could be caused by the hardware or the IDE I am using to test these algorithms.

When applied to randomly created arrays, the algorithms behaved in an expected way and yielded the most accurate results, supporting their theoretical run times. When used on ascending or descending arrays, selection sort maintained its O(n^2) complexity and since merge sort also works with O(nlogn) complexity under any circumstance, there weren't any visible changes in their running times. However, when quicksort was used on already sorted ascending or descending arrays, the running times were significantly higher due to the choice of pivot. Since the very first element of the array was chosen as pivot in any scenario, worst case behavior of quicksort was achieved, which made the quicksort algorithm complexity O(n^2) for those cases instead of O(nlogn), thus resulting in much higher running times than if the pivot was chosen in a better way. Radix sort also maintained its complexity for all three array types.