Ata Seren

21901575

Section-1

Homework 1

Question 1)

a) We need to find two positive constants: c and n_0 such that:

$$0 \leq 5n^3 + 4n^2 + 10 \leq cn^4 \text{ for all } n \geq n_0$$

Choose c = 5 and $n_0 = 2$

$$5n^3 + 4n^2 + 10 \leq 5n^4 \text{ for all } n \geq 2$$

b) Tracing of sorting algorithms for array: [24, 8, 51, 28, 20, 29, 21, 17, 38, 27]

Tracing of insertion sort:

Sorted (left of the red bar) ← → Unsorted (right of the red bar)

: first element of unsorted part

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	20	24	28	51	29	21	17	38	27
8	20	24	28	29	51	21	17	38	27
8	20	21	24	28	29	51	17	38	27
8	17	20	21	24	28	29	51	38	27
8	17	20	21	24	28	29	38	51	27
8	17	20	21	24	27	28	29	38	51

Tracing of bubble sort:

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	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

Pass 8:

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 9:

8	17	20	21	24	27	28	29	38	51
8	17	20	21	24	27	28	29	38	51

Question 2)

Screenshot of djikstra console for part C)

```
login as: ata.seren
ata.seren@dijkstra.ug.bcc.bilkent.edu.tr's password:
Last login: Wed Feb 24 14:00:08 2021 from 10.201.182.153
[ata.seren@dijkstra ~]$ cd /home/cs/ata.seren/cs202hw1
[ata.seren@dijkstra cs202hw1]$ ls
hw1  main.cpp  main.o  Makefile  sorting.cpp  sorting.h  sorting.o
[ata.seren@dijkstra cs202hw1]$ ./hw1
Array which will be sorted is: 12 7 11 18 19 9 6 14 21 3 17 20 5 12 14 8
Sorting array with selection sort and its result: 3 5 6 7 8 9 11 12 12 14 14 17
18 19 20 21
Key comparison count: 120
Data moves count: 45

Sorting array with merge sort and its result: 3 5 6 7 8 9 11 12 12 14 14 17 18 1
9 20 21
Key comparison count: 46
Data moves count: 188

Sorting array with quick sort and its result: 3 5 6 7 8 9 11 12 12 14 14 17 18 1
9 20 21
Key comparison count: 64
Data moves count: 102

Sorting array with radix sort and its result: 3 5 6 7 8 9 11 12 12 14 14 17 18 1
9 20 21
```

Exact output of performanceAnalysis function for part D)

```
For random arrays:
Analysis of Selection Sort
Array Size    Elapsed time          compCount          moveCount
6000              90 ms           17997000              17997
10000            240 ms            49995000              29997
14000            480 ms            97993000              41997
18000            800 ms           161991000              53997
22000           1190 ms           241989000              65997
26000           1660 ms           337987000              77997
30000           2210 ms           449985000              89997
Analysis of Merge Sort
Array Size    Elapsed time          compCount          moveCount
6000              0 ms               67827             175612
10000            10 ms              120545              307228
14000             0 ms              175370              443228
18000            10 ms              232044              582460
22000             0 ms              290049              726460
26000            10 ms              349302              870460
30000            10 ms              408667             1014460
Analysis of Quick Sort
Array Size    Elapsed time          compCount          moveCount
6000              0 ms               95058             151863
10000            10 ms              167774              260436
14000             0 ms              233609              352888
18000            10 ms              335352              497456
22000             0 ms              394266              641792
26000            10 ms              492442              732519
30000            10 ms              570984              904696
Analysis of Radix Sort
Array Size    Elapsed time
6000              0 ms
```

10000	10 ms
14000	20 ms
18000	10 ms
22000	30 ms
26000	20 ms
30000	30 ms

For ascending arrays:

Analysis of Selection Sort

Array Size	Elapsed time	compCount	moveCount
6000	80 ms	17997000	17997
10000	220 ms	49995000	29997
14000	430 ms	97993000	41997
18000	720 ms	161991000	53997
22000	1060 ms	241989000	65997
26000	1490 ms	337987000	77997
30000	1980 ms	449985000	89997

Analysis of Merge Sort

Array Size	Elapsed time	compCount	moveCount
6000	10 ms	39152	175612
10000	0 ms	69008	307228
14000	0 ms	99360	443228
18000	0 ms	130592	582460
22000	10 ms	165024	726460
26000	0 ms	197072	870460
30000	10 ms	227728	1014460

Analysis of Quick Sort

Array Size	Elapsed time	compCount	moveCount
6000	90 ms	18008999	23996
10000	230 ms	50014999	39996
14000	460 ms	98020999	55996
18000	760 ms	162026999	71996
22000	1140 ms	242032999	87996

26000	1600 ms	338038999	103996
30000	2120 ms	450044999	119996

Analysis of Radix Sort

Array Size	Elapsed time
6000	0 ms
10000	0 ms
14000	10 ms
18000	0 ms
22000	10 ms
26000	10 ms
30000	20 ms

For descending arrays:

Analysis of Selection Sort

Array Size	Elapsed time	compCount	moveCount
6000	80 ms	17997000	17997
10000	230 ms	49995000	29997
14000	460 ms	97993000	41997
18000	750 ms	161991000	53997
22000	1130 ms	241989000	65997
26000	1570 ms	337987000	77997
30000	2100 ms	449985000	89997

Analysis of Merge Sort

Array Size	Elapsed time	compCount	moveCount
6000	0 ms	36656	175612
10000	0 ms	64608	307228
14000	10 ms	94256	443228
18000	0 ms	124640	582460
22000	0 ms	154208	726460
26000	10 ms	186160	870460
30000	10 ms	219504	1014460

Analysis of Quick Sort

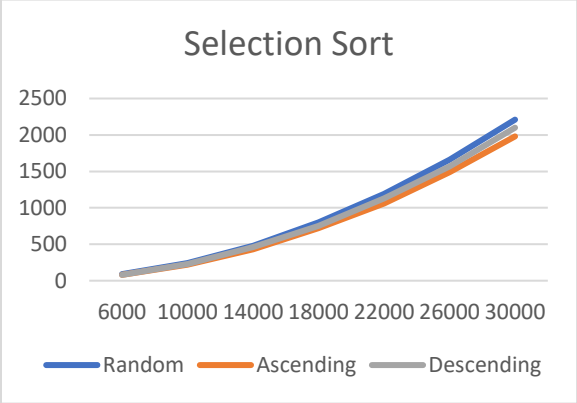
Array Size	Elapsed time	compCount	moveCount

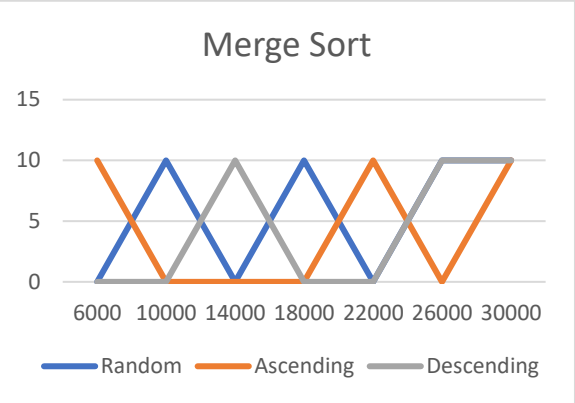
6000	150 ms	18008999	27023996
10000	420 ms	50014999	75039996
14000	820 ms	98020999	147055996
18000	1360 ms	162026999	243071996
22000	2030 ms	242032999	363087996
26000	2840 ms	338038999	507103996
30000	3780 ms	450044999	675119996

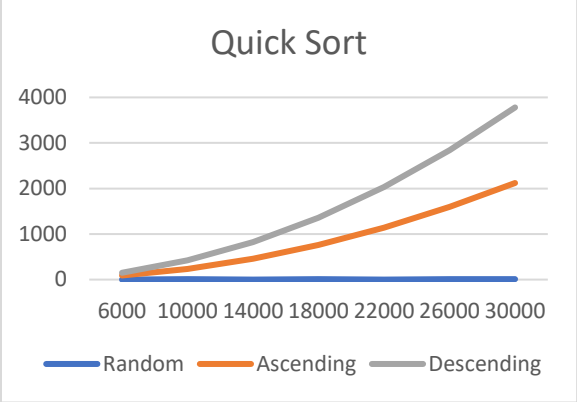
Analysis of Radix Sort

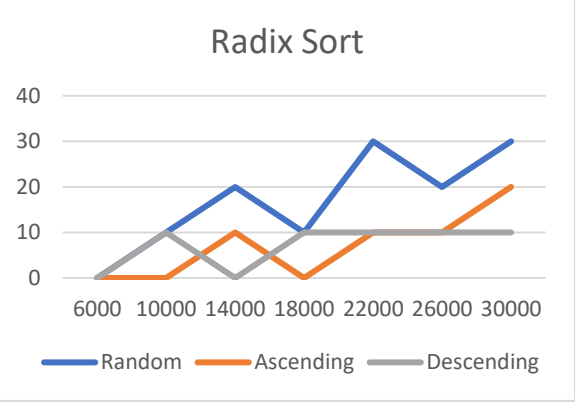
Array Size	Elapsed time
6000	0 ms
10000	10 ms
14000	0 ms
18000	10 ms
22000	10 ms
26000	10 ms
30000	10 ms

Question 3)









Empirical results for selection sort are very similar to theoretical results. It goes suitable with time complexity of $O(n^2)$ and it is similar in all types of arrays because it always runs for all array.

Merge sort is the fastest one which has the complexity of $O(n*\log_2 n)$. It is expected that it is more efficient than others in all types of arrays and we can see that from graphics.

Radix sort which has time complexity of O(n) is similar to merge sort too. It is also fast like merge sort. In random arrays, we see that it is little bit slower than other arrays.

Important thing about results of merge and radix sort is that, Djikstra machine gives results like 0 ms, 10 ms, 20 ms, etc. However, it is like 1 ms, 2 ms, 3 ms, etc. in my computer. This is probably caused that Djikstra can't measure time with a good precision and it rounds them. That's why their graphics are inaccurate. This rounding can be also seen in other functions' results. Also, we can't see accurate results according to their time complexities because of the small array sizes. However, if we use larger array sizes, we can see graphs different than these ones.

Quick sort is the most interesting one because it is very fast in random arrays, like merge and radix sort. However, in ascending and descending arrays, it is slower than with selection sort. Because, already sorted arrays are the worst cases of quick sorts. In random arrays, time complexity is $O(n*\log_2 n)$. However, it becomes $O(n^2)$ in sorted arrays. Because when we choose first array as pivot and try to sort a sorted array, we make algorithm to increase its running time on array.