CS202 HW1

Berdan Akyürek
21600904

1.

a) f(n) is $O(n^4)$ if $f(n) \leq c \cdot n^4$ for some $n \geq n_0$.

$f(n) = 5n^3+4n^2+10 \leq c \cdot n^4$

$5/n+4/n^2+10/n^4 \leq c$

If $n_0 = 1$ and $c \geq 19$, $5/n+4/n^2+10/n^4 \leq 19$ is satisfied for all $n \geq 1$.

Therefore, f(n) is $O(n^4)$.

b)

Insertion sort:

| marks between sorted and unsorted parts.

[24 | 8, 51, 28, 20, 29, 21, 17, 38, 27]
[8, 24 | 51, 28, 20, 29, 21, 17, 38, 27] insert 8 to index 0
[8, 24, 51 | 28, 20, 29, 21, 17, 38, 27] insert 51 to index 2
[8, 24, 28, 51 | 20, 29, 21, 17, 38, 27] insert 28 to index 2
[8, 20, 24, 28, 51 | 29, 21, 17, 38, 27] insert 20 to index 1
[8, 20, 24, 28, 29, 51 | 21, 17, 38, 27] insert 29 to index 4
[8, 20, 21, 24, 28, 29, 51 | 17, 38, 27] insert 21 to index 2
[8, 17, 20, 21, 24, 28, 29, 51 | 38, 27] insert 17 to index 1
[8, 17, 20, 21, 24, 28, 29, 38, 51 | 27] insert 38 to index 7
[8, 17, 20, 21, 24, 27, 28, 29, 38, 51|] insert 27 to index 5
[8, 17, 20, 21, 24, 27, 28, 29, 38, 51] array is sorted now

Bubble sort:

[24, 8, 51, 28, 20, 29, 21, 17, 38, 27]: swap(24, 8)
[8, 24, 51, 28, 20, 29, 21, 17, 38, 27]: 24 < 51. No swap.
[8, 24, 28, 51, 20, 29, 21, 17, 38, 27]: swap(28, 51)
[8, 24, 28, 20, 51, 29, 21, 17, 38, 27]: swap(20, 51)
[8, 24, 28, 20, 29, 51, 21, 17, 38, 27]: swap(29, 51)
[8, 24, 28, 20, 29, 21, 51, 17, 38, 27]: swap(21, 51)
[8, 24, 28, 20, 29, 21, 17, 51, 38, 27]: swap(17, 51)
[8, 24, 28, 20, 29, 21, 17, 38, 51, 27]: swap(38, 51)
[8, 24, 28, 20, 29, 21, 17, 38, 27, 51]: swap(27, 51)
[8, 24, 28, 20, 29, 21, 17, 38, 27, 51]: 8 < 24, 24 < 28, no swap.
[8, 24, 20, 28, 29, 21, 17, 38, 27, 51]: swap(20, 28)
[8, 24, 20, 28, 29, 21, 17, 38, 27, 51]: 28 < 29. no swap.
[8, 24, 20, 28, 21, 29, 17, 38, 27, 51]: swap(21, 29)
[8, 24, 20, 28, 21, 17, 29, 38, 27, 51]: swap(17, 29)
[8, 24, 20, 28, 21, 17, 29, 38, 27, 51]: 29 < 38. no swap.

[8, 24, 20, 28, 21, 17, 29, 27, 38, 51]: swap(27, 38)
[8, 24, 20, 28, 21, 17, 29, 27, 38, 51]: 8 < 24. no swap.
[8, 20, 24, 28, 21, 17, 29, 27, 38, 51]: swap(20, 24)
[8, 20, 24, 28, 21, 17, 29, 27, 38, 51]: 24 < 28. no swap.
[8, 20, 24, 21, 28, 17, 29, 27, 38, 51]: swap(21, 28)
[8, 20, 24, 21, 17, 28, 29, 27, 38, 51]: swap(17, 28)
[8, 20, 24, 21, 17, 28, 29, 27, 38, 51]: 28 < 29, no swap.
[8, 20, 24, 21, 17, 28, 27, 29, 38, 51]: swap(27, 29)
[8, 20, 24, 21, 17, 28, 27, 29, 38, 51]: 8 < 20, 20 < 24. no swap.
[8, 20, 21, 24, 17, 28, 27, 29, 38, 51]: swap(21, 24)
[8, 20, 21, 17, 24, 28, 27, 29, 38, 51]: swap(17, 24)
[8, 20, 21, 17, 24, 28, 27, 29, 38, 51]: 24 < 28. no swap.
[8, 20, 21, 17, 24, 27, 28, 29, 38, 51]: swap(27, 28)
[8, 20, 21, 17, 24, 27, 28, 29, 38, 51]: 8 < 20, 20 < 21. no swap.
[8, 20, 17, 21, 24, 27, 28, 29, 38, 51]: swap(17, 21)
[8, 20, 17, 21, 24, 27, 28, 29, 38, 51]: 21 < 24, 24 < 27, 8 < 20. no swap.
[8, 17, 20, 21, 24, 27, 28, 29, 38, 51]: swap(17, 20)
[8, 17, 20, 21, 24, 27, 28, 29, 38, 51]: array is sorted now.

2.

Execution results of question 2-c:

```
Before sorting
12 7 11 18 19 9 6 14 21 3 17 20 5 12 14 8
-------------------------
After selection sort
3 5 6 7 8 9 11 12 12 14 14 17 18 19 20 21
compCount: 120
moveCount: 45
-------------------------
After merge sort
3 5 6 7 8 9 11 12 12 14 14 17 18 19 20 21
compCount: 46
moveCount: 128
-------------------------
After quick sort
3 5 6 7 8 9 11 12 12 14 14 17 18 19 20 21
compCount: 45
moveCount: 102
-------------------------
After radix sort
3 5 6 7 8 9 11 12 12 14 14 17 18 19 20 21
-------------------------
```

Execution results of performanceAnalysis function:

Random arrays:

```
Random
-----------------------------------------------------
Analysis of Selection Sort
Array Size      Elapsed time    compCount       moveCount
6000            76 ms           17997000        17997
10000           122 ms          49995000        29997
14000           234 ms          97993000        41997
18000           383 ms          161991000       53997
22000           572 ms          241989000       65997
26000           798 ms          337987000       77997
30000           1094 ms         449985000       89997
-----------------------------------------------------
Analysis of Merge Sort
Array Size      Elapsed time    compCount       moveCount
6000            0 ms            67695           151616
10000           1 ms            120128          267232
14000           3 ms            174953          387232
18000           3 ms            231356          510464
22000           4 ms            289384          638464
26000           5 ms            347944          766464
30000           6 ms            407530          894464
-----------------------------------------------------
Analysis of Quick Sort
Array Size      Elapsed time    compCount       moveCount
6000            1 ms            230915          92861
10000           2 ms            589205          156726
14000           4 ms            1097555         248293
18000           6 ms            1763388         285059
22000           9 ms            2596335         371829
26000           12 ms           3615774         439528
30000           16 ms           4739428         500957
-----------------------------------------------------
Analysis of Radix Sort
Array Size      Elapsed time
6000            0 ms
10000           0 ms
14000           1 ms
18000           1 ms
22000           1 ms
26000           2 ms
30000           2 ms
```

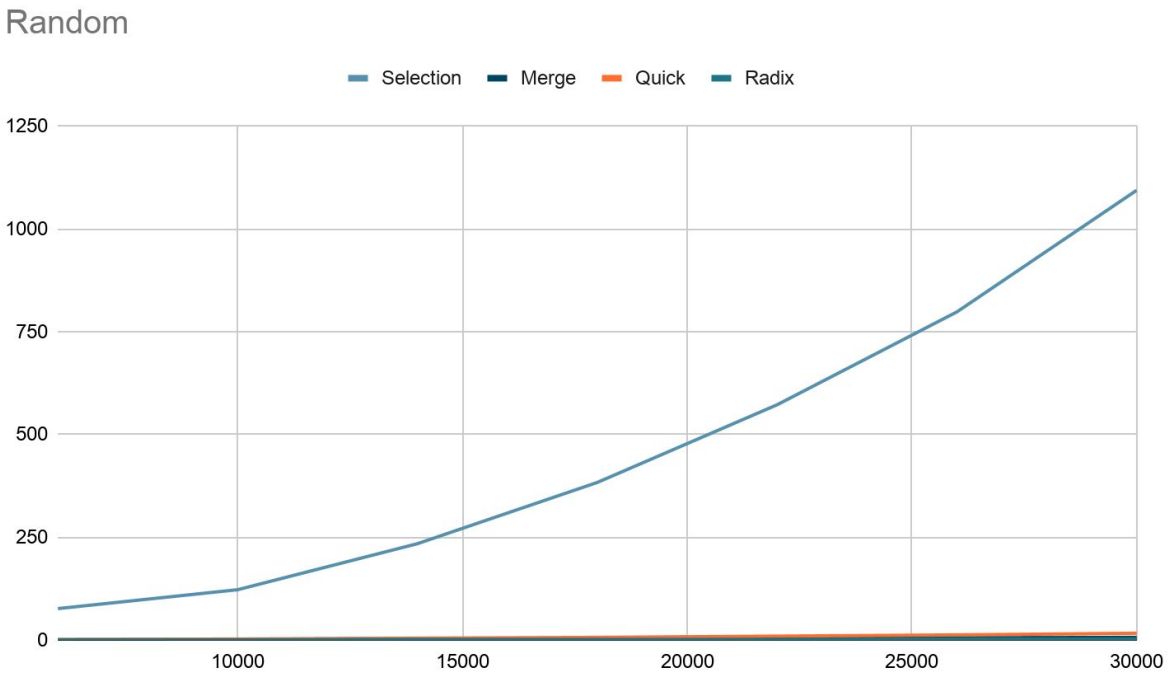
Ascending arrays:

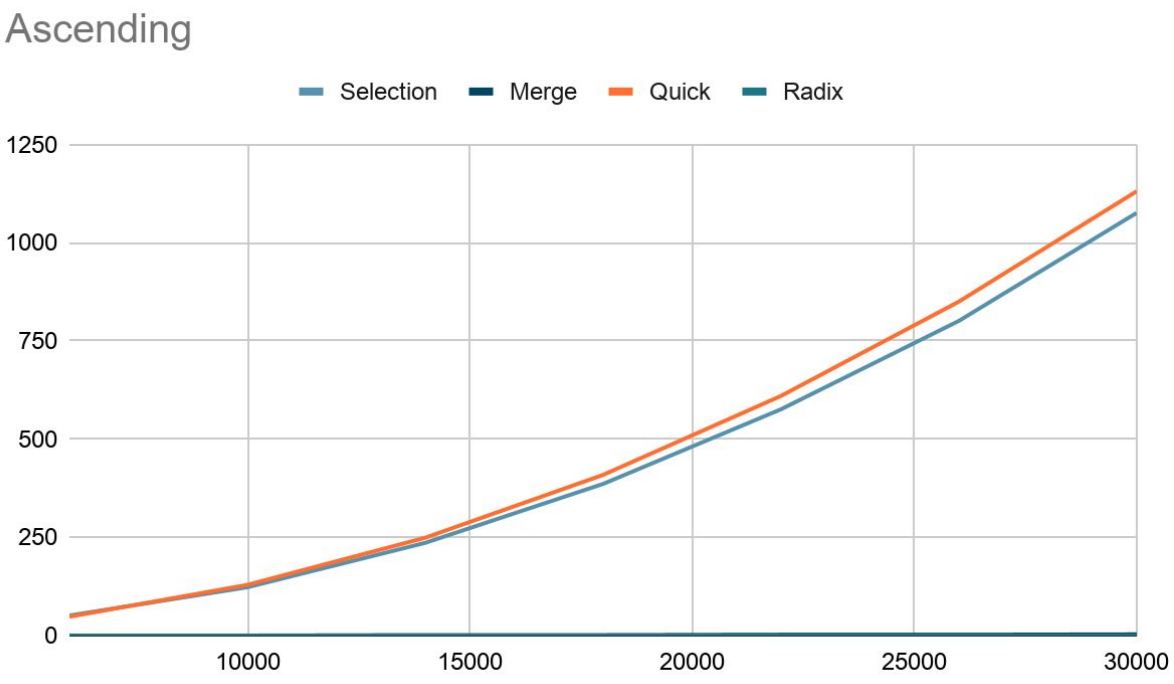
```
Ascending
-----------------------------------------------------
Analysis of Selection Sort
Array Size      Elapsed time    compCount       moveCount
6000            51 ms           17997000        17997
10000           123 ms          49995000        29997
14000           236 ms          97993000        41997
18000           386 ms          161991000       53997
22000           576 ms          241989000       65997
26000           801 ms          337987000       77997
30000           1076 ms         449985000       89997
-----------------------------------------------------
Analysis of Merge Sort
Array Size      Elapsed time    compCount       moveCount
6000            0 ms            42267           151616
10000           0 ms            74977           267232
14000           1 ms            107898          387232
18000           1 ms            141781          510464
22000           2 ms            175277          638464
26000           2 ms            210231          766464
30000           3 ms            249333          894464
-----------------------------------------------------
Analysis of Quick Sort
Array Size      Elapsed time    compCount       moveCount
6000            48 ms           17997000        23996
10000           129 ms          49995000        39996
14000           249 ms          97993000        55996
18000           409 ms          161991000       71996
22000           610 ms          241989000       87996
26000           850 ms          337987000       103996
30000           1131 ms         449985000       119996
-----------------------------------------------------
Analysis of Radix Sort
Array Size      Elapsed time
6000            0 ms
10000           0 ms
14000           0 ms
18000           0 ms
22000           0 ms
26000           0 ms
30000           0 ms
```

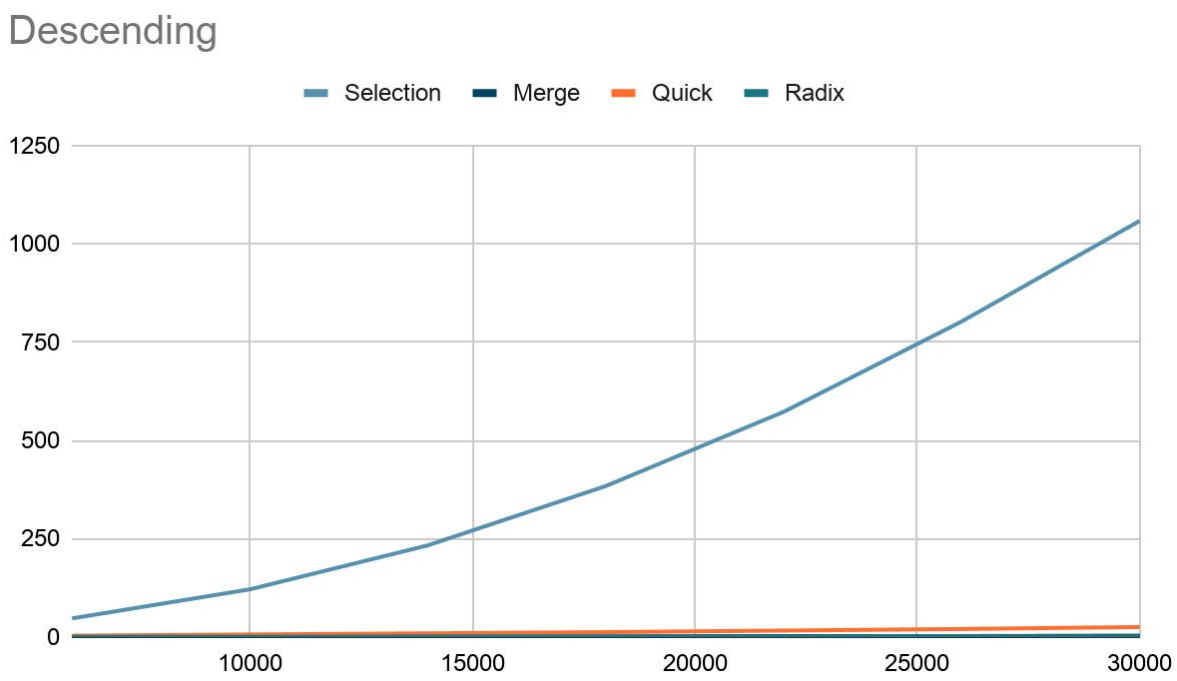
Descending arrays:

```
Descending
-----------------------------------------------------
Analysis of Selection Sort
Array Size      Elapsed time    compCount       moveCount
6000            47 ms           17997000        17997
10000           121 ms          49995000        29997
14000           233 ms          97993000        41997
18000           384 ms          161991000       53997
22000           573 ms          241989000       65997
26000           803 ms          337987000       77997
30000           1059 ms         449985000       89997
-----------------------------------------------------
Analysis of Merge Sort
Array Size      Elapsed time    compCount       moveCount
6000            0 ms            36656           151616
10000           1 ms            64608           267232
14000           1 ms            94256           387232
18000           1 ms            124640          510464
22000           2 ms            154208          638464
26000           2 ms            186160          766464
30000           3 ms            219504          894464
-----------------------------------------------------
Analysis of Quick Sort
Array Size      Elapsed time    compCount       moveCount
6000            3 ms            761357          900117
10000           6 ms            1476911         1512520
14000           9 ms            2355790         2134339
18000           12 ms           3380877         2734811
22000           16 ms           4570049         3340695
26000           20 ms           5918021         3945445
30000           25 ms           7446080         4568468
-----------------------------------------------------
Analysis of Radix Sort
Array Size      Elapsed time
6000            0 ms
10000           0 ms
14000           1 ms
18000           1 ms
22000           1 ms
26000           2 ms
30000           1 ms
```

3.


Random
Selection
Merge
Quick
Radix
1250
1000
750
500
250
0
10000
15000
20000
25000
30000


Ascending
Selection
Merge
Quick
Radix
1250
1000
750
500
250
0
10000
15000
20000
25000
30000



According to the results of the experiment, experimental results were close to theoretical results. It is possible to see a quadratic growth in selection sort for any case in graphs. This is as expected since all best, average and worst cases of selection sort is n^2.
In merge sort, the growth rate was similar for all cases. This is as expected because the best, average and worst cases of merge sort is all n log n.
In quick sort, it is possible to see an increase similar to merge sort for best and average cases. But in the worst case, quick sort is closer to selection sort. This is normal as well because the best and average cases of quick sort is n log n, which is the same as merge sort, but in the worst case it is n^2 which is the same as selection sort.
Lastly, in radix sort, it is possible to see a growth closer to linear. This is also as expected because all best, average and worst case of radix sort is nk which is linear.
Although the results were close to theoretical results, they were not perfectly the same. This is also expected because the operation speed of the CPU is variable over time according to various reasons.
The only significant change in time complexity between random, ascending and descending arrays was on merge sort. In an ascending array, quick sort growth is n^2 while n log n in others. This is normal because the pivot of quick sort is always chosen as the first element. If the array is already sorted, the pivot is always the minimum element of the array. This increases the running time.