CS202 HW1

Berdan Akyürek
21600904
CS202-1
HW1

Q1-

(a)
$f(n) = 20n^4+10n^2+5$
$g(n) = n^5$
By the definition of big-oh,

If $f(n) <= c*g(n)$ for all $n >= n_0$, $f(n) \in O(g(n))$.
Then, If $20n^4+10n^2+5 <= c*n^5$ for all $n >= n_0$, $f(n) \in O(n^5)$.
If we take $n_0 = 2$ and $c = 32$,

$20n^4+10n^2+5 <= 32*n^5$ for all $n >= 2$ is a true statement.
So, by the definition, $f(n) \in O(n^5)$.

(b)
Selection Sort:

left of the |, unsorted part
right of the |, sorted part
[18 4 47 24 15 24 17 11 31 23 |]

<u>Insertion 1:</u>

Find the biggest number in unsorted part.
[18 4 **47** 24 15 24 17 11 31 23 |]
Insert it at the beginning of sorted part. Shift unsorted numbers left.
[18 4 24 15 24 17 11 31 23 | **47**]

<u>Insertion 2:</u>

Find the biggest number in unsorted part.
[18 4 24 15 24 17 11 **31** 23 | 47]
Insert it at the beginning of sorted part.
[18 4 24 15 24 17 11 23 | **31** 47]

<u>Insertion 3:</u>

Find the biggest number in unsorted part.
[18 4 **24** 15 24 17 11 23 | 31 47]
Insert it at the beginning of sorted part.
[18 4 15 24 17 11 23 | **24** 31 47]

<u>Insertion 4:</u>

Find the biggest number in unsorted part.
[18 4 15 **24** 17 11 23 | 24 31 47]
Insert it at the beginning of sorted part.
[18 4 15 17 11 23 | **24** 24 31 47]

<u>Insertion 5:</u>

Find the biggest number in unsorted part.

[18 4 15 17 11 **23** | 24 24 31 47]
Insert it at the beginning of sorted part.
[18 4 15 17 11 | **23** 24 24 31 47]

<u>Insertion 6:</u>
Find the biggest number in unsorted part.
[**18** 4 15 17 11 | 23 24 24 31 47]
Insert it at the beginning of sorted part.
[4 15 17 11 | **18** 23 24 24 31 47]

<u>Insertion 7:</u>
Find the biggest number in unsorted part.
[4 15 **17** 11 | 18 23 24 24 31 47]
Insert it at the beginning of sorted part.
[4 15 11 | **17** 18 23 24 24 31 47]

<u>Insertion 8:</u>
Find the biggest number in unsorted part.
[4 **15** 11 | 17 18 23 24 24 31 47]
Insert it at the beginning of sorted part.
[4 11 | **15** 17 18 23 24 24 31 47]

<u>Insertion 9:</u>
Find the biggest number in unsorted part.
[4 **11** | 15 17 18 23 24 24 31 47]
Insert it at the beginning of sorted part.
[4 | **11** 15 17 18 23 24 24 31 47]

<u>Insertion 10:</u>
Find the biggest number in unsorted part.
[**4** | 11 15 17 18 23 24 24 31 47]
Insert it at the beginning of sorted part.
[| **4** 11 15 17 18 23 24 24 31 47]

Array is sorted now.

Bubble Sort:

[18, 4, 47, 24, 15, 24, 17, 11, 31, 23]
<u>Turn 1:</u>
<u>Swap 1:</u>
Check first two elements. 18>4. Swap.
[**4**, **18**, 47, 24, 15, 24, 17, 11, 31, 23]

<u>Swap 2:</u>
Check next two elements. 18<47. No swap.
[4, **18**, **47**, 24, 15, 24, 17, 11, 31, 23]
……….
……….
<u>Swap 9:</u>
Check last two elements. 47>23. Swap.
[4, 18, 24, 15, 24, 17, 11, 31, **23**, **47**]

Now last element is on the right place. No need to check it.

Turn 2:

Start from the beginning again.

Swap 1:

Check first two elements. 4<18. No Swap.

[**4, 18**, 24, 15, 24, 17, 11, 31, 23, 47]

……….

……….

Swap 8:

Check next two elements. 31>23. No Swap.

[4, 18, 15, 24, 17, 11, 24, **23**, **31**, 47]

The last element is already on its right place. No need to check.

Turn 3:

Start from the beginning.

……….

[4, 15, 18, 17, 11, 24, 23, **24, 31, 47**]

Turn 4:

Start from the beginning.

……….

[4, 15, 17, 11, 18, 23, **24, 24, 31, 47**]

Turn 5:

[4, 15, 11,17, 18, **23**, **24, 24, 31, 47**]

Turn 6:

[4, 11, 15, 17, **18**, **23**, **24, 24, 31, 47**]

Turn 7:

[4, 11, 15, **17**, **18**, **23**, **24, 24, 31, 47**]

Turn 8:

[4, 11, **15**, **17**, **18**, **23**, **24, 24, 31, 47**]

Turn 9:

[4, **11**, **15**, **17**, **18**, **23**, **24, 24, 31, 47**]

Array is sorted now.

Q2-

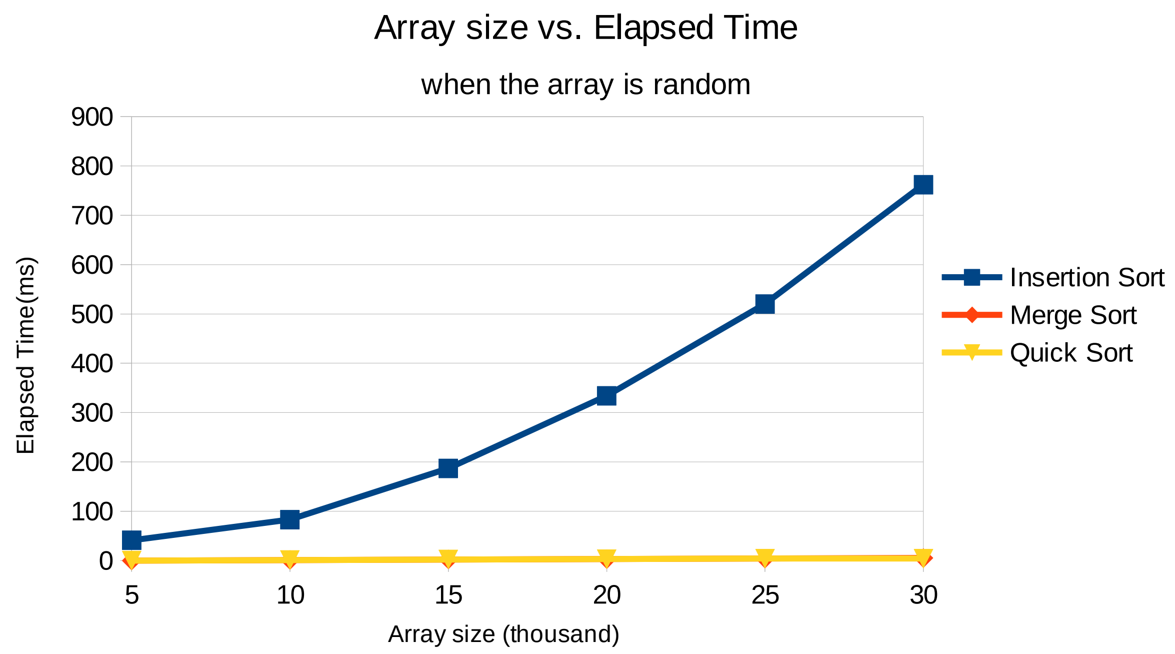
Output of the hw1 executable created with Makefile:

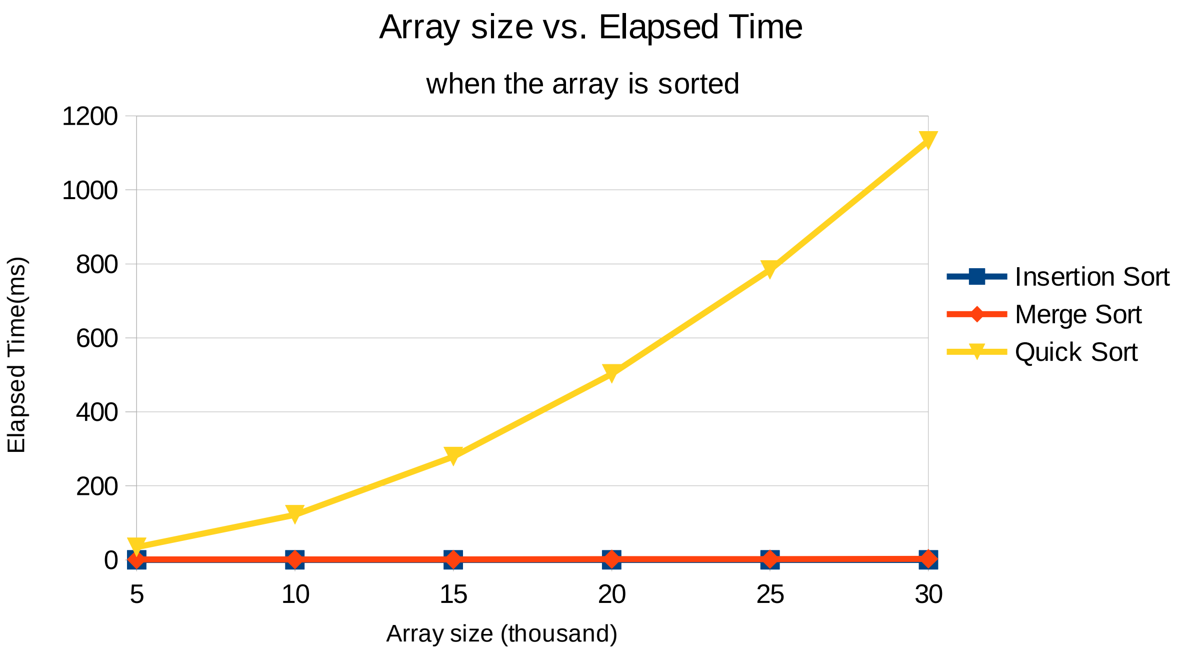
```
berdan@berdan-Inspiron-15-3567:~/Documents/Dersler/CS202/HW1$ make
g++ -c main.cpp
g++ -c sorting.cpp
g++ -c auxArrayFunctions.cpp
g++ main.o sorting.o auxArrayFunctions.o -o hw1
berdan@berdan-Inspiron-15-3567:~/Documents/Dersler/CS202/HW1$ ls
auxArrayFunctions.cpp  auxArrayFunctions.h  auxArrayFunctions.o  hw1  main.cpp  main.o  Makefile  sorting.cpp  sorting.h  sorting.o
berdan@berdan-Inspiron-15-3567:~/Documents/Dersler/CS202/HW1$ ./hw1
Results For Insertion Sort:
Comparison Count: 74
Move Count: 89
0       2       3       5       6       7       8       9       9       11      11      14      15      16      17      18
Results For Merge Sort:
Comparison Count: 46
Move Count: 128
0       2       3       5       6       7       8       9       9       11      11      14      15      16      17      18
Results For Quick Sort:
Comparison Count: 47
Move Count: 114
0       2       3       5       6       7       8       9       9       11      11      14      15      16      17      18
berdan@berdan-Inspiron-15-3567:~/Documents/Dersler/CS202/HW1$
```

Performance Analysis results:

```
berdan@berdan-Inspiron-15-3567:~/Documents/Dersler/CS202/HW1$ make
g++ -c sorting.cpp
g++ main.o sorting.o auxArrayFunctions.o -o hw1
berdan@berdan-Inspiron-15-3567:~/Documents/Dersler/CS202/HW1$ ./hw1
Random array
-----------------------------------------------------
Part c - Time analysis of Insertion Sort
Array Size          Time Elapsed        compCount           moveCount
5000                41                  6291499             6296498
10000               83                  25082324            25092323
15000               187                 56197604            56212603
20000               334                 100794392           100814391
25000               520                 156059341           156084340
30000               762                 226424979           226454978
-----------------------------------------------------
Part c - Time analysis of Merge Sort
Array Size          Time Elapsed        compCount           moveCount
5000                0                   55245               123616
10000               1                   120469              267232
15000               2                   189378              417232
20000               3                   261028              574464
25000               4                   334062              734464
30000               5                   408642              894464
-----------------------------------------------------
Part c - Time analysis of Quick Sort
Array Size          Time Elapsed        compCount           moveCount
5000                0                   67954               115199
10000               1                   153700              246457
15000               2                   243269              356565
20000               3                   328301              526737
25000               4                   417577              677744
30000               4                   528019              838182
berdan@berdan-Inspiron-15-3567:~/Documents/Dersler/CS202/HW1$
```

```
berdan@berdan-Inspiron-15-3567:~/Documents/Dersler/CS202/HW1$ make
g++ -c sorting.cpp
g++ main.o sorting.o auxArrayFunctions.o -o hw1
berdan@berdan-Inspiron-15-3567:~/Documents/Dersler/CS202/HW1$ ./hw1
Already sorted array
-----------------------------------------------------
Part c - Time analysis of Insertion Sort
Array Size          Time Elapsed        compCount           moveCount
5000                0                   4999                9998
10000               0                   9999                19998
15000               0                   14999               29998
20000               0                   19999               39998
25000               0                   24999               49998
30000               0                   29999               59998
-----------------------------------------------------
Part c - Time analysis of Merge Sort
Array Size          Time Elapsed        compCount           moveCount
5000                1                   32004               123616
10000               1                   69008               267232
15000               1                   106364              417232
20000               2                   148016              574464
25000               2                   188476              734464
30000               3                   227728              894464
-----------------------------------------------------
Part c - Time analysis of Quick Sort
Array Size          Time Elapsed        compCount           moveCount
5000                34                  12497500            19996
10000               122                 49995000            39996
15000               279                 112492500           59996
20000               503                 199990000           79996
25000               784                 312487500           99996
30000               1133                449985000           119996
berdan@berdan-Inspiron-15-3567:~/Documents/Dersler/CS202/HW1$
```





According to the results, when the array is random, the least successful sorting algorithm is Insertion sort. Because insertion sort makes an array search for each step to find the biggest number and this takes time. In the other hand, when array is already sorted, the least successful sorting algorithm is Quick sort because the partition operation takes a big amount of time when array is sorted.

Q3-

Results of Performance analysis of Nearly sorted arrays for different K values:

```
g++ main.o sorting.o auxArrayFunctions.o -o hw1
berdan@berdan-Inspiron-15-3567:~/Documents/Dersler/CS202/HW1$ ls
auxArrayFunctions.cpp  auxArrayFunctions.h  auxArrayFunctions.o  hw1  hw1.odt  main.cpp  main.o  Makefile  sorting.cpp  sorting.h  sorting.o
berdan@berdan-Inspiron-15-3567:~/Documents/Dersler/CS202/HW1$ ./hw1
5.09
5.1244
5.11253
5.1106
5.1204
5.07513
Nearly sorted array K = 10
-----------------------------------------------------
Part c - Time analysis of Insertion Sort
Array Size          Time Elapsed        compCount           moveCount
5000                0                   21035               26034
10000               0                   42271               52270
15000               0                   63322               78321
20000               0                   84716               104715
25000               0                   105921              130920
30000               0                   126194              156193
-----------------------------------------------------
Part c - Time analysis of Merge Sort
Array Size          Time Elapsed        compCount           moveCount
5000                0                   36441               123616
10000               1                   77881               267232
15000               1                   121146              417232
20000               3                   165998              574464
25000               3                   211370              734464
30000               5                   257140              894464
-----------------------------------------------------
Part c - Time analysis of Quick Sort
Array Size          Time Elapsed        compCount           moveCount
5000                7                   2720877             36979
10000               30                  10449199            75027
15000               66                  23426785            112322
20000               112                 42218456            150366
25000               167                 64883862            187125
30000               244                 94111318            224850
berdan@berdan-Inspiron-15-3567:~/Documents/Dersler/CS202/HW1$
```

```
49.8817
50.4405
49.5894
49.9183
50.0128
49.3941
50.1609
49.4433
49.9509
50.0431
Nearly sorted array K = 100
-----------------------------------------------------
Part c - Time analysis of Insertion Sort
Array Size          Time Elapsed        compCount           moveCount
5000                0                   170977              175976
10000               1                   343242              353241
15000               2                   515134              530133
20000               2                   683115              703114
25000               3                   855246              880245
30000               4                   1026036             1056035
-----------------------------------------------------
Part c - Time analysis of Merge Sort
Array Size          Time Elapsed        compCount           moveCount
5000                0                   44220               123616
10000               1                   93602               267232
15000               2                   144553              417232
20000               3                   197383              574464
25000               4                   250720              734464
30000               4                   304678              894464
-----------------------------------------------------
Part c - Time analysis of Quick Sort
Array Size          Time Elapsed        compCount           moveCount
5000                1                   346266              82396
10000               4                   1294420             165571
15000               9                   2750888             256027
20000               16                  5202508             332793
25000               24                  7816218             424770
30000               32                  11107132            497921
berdan@berdan-Inspiron-15-3567:~/Documents/Dersler/CS202/HW1$
```

```
495.566
493.385
496.532
494.237
494.051
Nearly sorted array K = 1000
-----------------------------------------------------
Part c - Time analysis of Insertion Sort
Array Size          Time Elapsed        compCount           moveCount
5000                17                  1599753             1604752
10000               13                  3295574             3305573
15000               20                  4939255             4954254
20000               28                  6595513             6615512
25000               31                  8286025             8311024
30000               39                  9931120             9961119
-----------------------------------------------------
Part c - Time analysis of Merge Sort
Array Size          Time Elapsed        compCount           moveCount
5000                1                   52269               123616
10000               1                   110088              267232
15000               3                   169782              417232
20000               3                   231173              574464
25000               4                   292837              734464
30000               5                   355108              894464
-----------------------------------------------------
Part c - Time analysis of Quick Sort
Array Size          Time Elapsed        compCount           moveCount
5000                1                   96650               155262
10000               1                   244500              364330
15000               3                   514383              550339
20000               4                   923878              684044
25000               5                   1169494             893090
30000               7                   1538122             1070749
berdan@berdan-Inspiron-15-3567:~/Documents/Dersler/CS202/HW1$
```

In this experiment, when K is greater, the array is more random and when K is smaller, the array is more sorted.
The results show that when K is getting bigger, the efficiency of Insertion sort is decreasing and the efficiency of Quick Sort is increasing. Also when K s getting smaller, the efficiency of Insertion sort is increasing and the efficiency of Quick Sort is decreasing.
This means, for random arrays using quick sort is a better choice but for sorted and almost sorted arrays, using insertion sort is better.
But if we look Merge Sort, it works with almost the same efficiency for more random and more ordered arrays. The efficiency of Merge Sort is not strongly dependent to the order of the array.
Since we will need to sort random, sorted and nearly sorted arrays, it is better to use Merge sort as a solution to this problem. Because it works better comparing to other algorithms under different conditions.