CS 202, Spring 2022
Homework 1
-
Algorithm Efficiency

Alperen CAN

Question 1

a) Show that $f(n) = 8n^4 + 5n^3 + 7$ is $O(n^5)$ by specifying appropriate c and n_0 values in Big-O definition.

We need to show that $f(n) = O(n^5)$ if $\exists$ positive constants c, n_0 such that

(Eq 1) $0 \le f(n) \le cn^5, \forall n \ge n_0.$

If we choose c and n_0 such that c = 20 and n_0 = 1, then our equation becomes

(Eq 2) $0 \le f(n) = 8n^4 + 5n^3 + 7 \le 20n^5, \forall n \ge 1$

(Eq 3) $0 \le f(n) = 8n^4 \le 8n^5, \forall n \ge 1$

(Eq 4) $0 \le f(n) = 5n^3 \le 5n^5, \forall n \ge 1$

(Eq 5) $0 \le f(n) = 7 \le 7n^5, \forall n \ge 1$

If we add Eq (3-5), we get our equation Eq 2. Therefore, Eq 2 is correct and $f(n) = 8n^4 + 5n^3 + 7$ is $O(n^5)$.

b) Trace the following sorting algorithms to sort the array [22, 8, 49, 25, 18, 30, 20, 15, 35, 27] in ascending order. Use the array implementation of the algorithms as described in the textbook and show all major steps.

- Selection sort
- Bubble sort

1) Selection Sort

- Initial: All Unsorted

22	8	49	25	18	30	20	15	35	27

- Largest -> 49: (49 - 27 swap)

| Sorted

22	8	27	25	18	30	20	15	35	49

- Largest -> 35

| Sorted

22	8	27	25	18	30	20	15	35	49

- Largest -> 30: (30 - 15 swap)

| Sorted

22	8	27	25	18	15	20	30	35	49

- Largest -> 27: (27 - 20 swap)

| Sorted

22	8	20	25	18	15	27	30	35	49

- Largest -> 25: (25 - 15 swap)

| Sorted

22	8	20	15	18	25	27	30	35	49

- Largest -> 22: (22 - 18 swap)

| Sorted

18	8	20	15	22	25	27	30	35	49

- Largest -> 20: (20 - 15 swap)

| Sorted

18	8	15	20	22	25	27	30	35	49

- Largest -> 20: (18 - 15 swap)

| Sorted

15	8	18	20	22	25	27	30	35	49

- Largest -> 15: (15 - 8 swap)

| Sorted

8	15	18	20	22	25	27	30	35	49

2) Bubble Sort

- Initial: All Unsorted

22	8	49	25	18	30	20	15	35	27

- Pass 1: 22 > 8: swap

8	22	49	25	18	30	20	15	35	27

- Pass 1: 49 > 25: swap

8	22	25	49	18	30	20	15	35	27

- Pass 1: 49 > 18: swap

8	22	25	18	49	30	20	15	35	27

- Pass 1: 49 > 30: swap

8	22	25	18	30	49	20	15	35	27

- Pass 1: 49 > 20: swap

8	22	25	18	30	20	49	15	35	27

- Pass 1: 49 > 15: swap

8	22	25	18	30	20	15	49	35	27

- Pass 1: 49 > 35: swap

8	22	25	18	30	20	15	35	49	27

- Pass 1: 49 > 27: swap

| Sorted

8	22	25	18	30	20	15	35	27	49

- Pass 2: 25 > 18: swap

8	22	18	25	30	20	15	35	27	49

- Pass 2: 30 > 20: swap

8	22	18	25	20	30	15	35	27	49

- Pass 2: 30 > 15: swap

8	22	18	25	20	15	30	35	27	49

- Pass 2: 35 > 27: swap

| Sorted

8	22	18	25	20	15	30	27	35	49

- Pass 3: 22 > 18: swap

8	18	22	25	20	15	30	27	35	49

- Pass 3: 25 > 20: swap

8	18	22	20	25	15	30	27	35	49

- Pass 3: 25 > 15: swap

8	18	22	20	15	25	30	27	35	49

- Pass 3: 30 > 27: swap

| Sorted

8	18	22	20	15	25	27	30	35	49

- Pass 4: 22 > 20: swap

8	18	20	22	15	25	27	30	35	49

- Pass 4: 22 > 15: swap

| Sorted

8	18	20	15	22	25	27	30	35	49

- Pass 5: 20 > 15: swap

| Sorted

8	18	15	20	22	25	27	30	35	49

- Pass 6: 18 > 15: swap

| Sorted

8	15	18	20	22	25	27	30	35	49

- Pass 7: No swap: Finish

| Sorted

8	15	18	20	22	25	27	30	35	49

Question 2

Screenshots for the requested outputs of the program are listed below.

```
------------------------Insertion Sort------------------------------
Comparison Count: 69
Move Count: 88
[1,2,4,5,6,7,8,9,11,12,13,16,16,17,18,20]
------------------------Bubble Sort---------------------------------
Comparison Count: 110
Move Count: 174
[1,2,4,5,6,7,8,9,11,12,13,16,16,17,18,20]
------------------------Merge Sort----------------------------------
Comparison Count: 47
Move Count: 128
[1,2,4,5,6,7,8,9,11,12,13,16,16,17,18,20]
------------------------Quick Sort----------------------------------
Comparison Count: 50
Move Count: 116
[1,2,4,5,6,7,8,9,11,12,13,16,16,17,18,20]

--------------------------------------------------------------------
--------------------------------------------------------------------
```

Output 1. Array Sorting Results

```
--------------------------------------------------------------------
Analysis of Random Array Scenario
-----------------------------------------------------
Analysis of Insertion Sort
Array Size      Elapsed time            compCount               moveCount
5000            15263 ms                6198993         6203997
10000           60343 ms                24984764                24994770
15000           132682 ms               56059223                56074233
20000           236596 ms               99610742                99630752
25000           368020 ms               156813750               156838753
30000           528064 ms               224861836               224891844
35000           724337 ms               307993705               308028712
40000           941167 ms               399617471               399657477
-----------------------------------------------------
Analysis of Bubble Sort
Array Size      Elapsed time            compCount               moveCount
5000            44602 ms                12495154                18581997
10000           207984 ms               49984269                74924316
15000           529845 ms               112468629               168132705
20000           996093 ms               199982374               298772262
25000           1613127 ms              312472794               470366265
30000           2368750 ms              449946219               674495538
35000           3297113 ms              612465664               923876142
40000           4335733 ms              799933640               1198732437
-----------------------------------------------------
Analysis of Merge Sort
Array Size      Elapsed time            compCount               moveCount
5000            732 ms          55250           123616
10000           1476 ms         120511          267232
15000           2341 ms         189259          417232
20000           3141 ms         260865          574464
25000           3821 ms         334163          734464
30000           4747 ms         408755          894464
35000           5528 ms         484664          1058928
40000           6453 ms         561905          1228928
-----------------------------------------------------
Analysis of Quick Sort
Array Size      Elapsed time            compCount               moveCount
5000            449 ms          66435           113918
10000           971 ms          171886          277808
15000           1469 ms         239528          386254
20000           2036 ms         346591          568652
25000           2612 ms         441529          666306
30000           3101 ms         514365          858243
35000           3726 ms         623024          1031365
40000           4347 ms         727982          1239837

--------------------------------------------------------------------
--------------------------------------------------------------------
```

Output 2. Performance Analysis for Random Array Scenario

```
----------------------------------------------------------------------
Analysis of Almost Sorted Array Scenario
------------------------------------------------------
Analysis of Insertion Sort
Array Size      Elapsed time            compCount               moveCount
5000            1964 ms         805951          810950
10000           7593 ms         3142005         3152004
15000           16214 ms                6882767         6897766
20000           28590 ms                12077141                12097140
25000           44673 ms                18932441                18957440
30000           67480 ms                28582649                28612648
35000           91439 ms                38713069                38748068
40000           115543 ms               49134849                49174848
------------------------------------------------------
Analysis of Bubble Sort
Array Size      Elapsed time            compCount               moveCount
5000            33199 ms                12475345                2402856
10000           130333 ms               49928570                9396018
15000           290717 ms               112335980               20603304
20000           522758 ms               199730440               36171426
25000           824692 ms               312294369               56722326
30000           1192574 ms              449743835               85657950
35000           1627187 ms              611942220               116034210
40000           2118706 ms              798912009               147284550
------------------------------------------------------
Analysis of Merge Sort
Array Size      Elapsed time            compCount               moveCount
5000            494 ms          50728           123616
10000           1030 ms         111862          267232
15000           1554 ms         174810          417232
20000           2142 ms         244943          574464
25000           2757 ms         310343          734464
30000           3260 ms         379756          894464
35000           3904 ms         451400          1058928
40000           4462 ms         526799          1228928
------------------------------------------------------
Analysis of Quick Sort
Array Size      Elapsed time            compCount               moveCount
5000            720 ms          219085          244765
10000           1345 ms         428898          409271
15000           2229 ms         681569          805633
20000           3481 ms         1122294         1068541
25000           3436 ms         1027109         1294360
30000           7155 ms         2515687         1846758
35000           7673 ms         2716333         1747525
40000           8943 ms         3161653         2149333

----------------------------------------------------------------------
----------------------------------------------------------------------
```

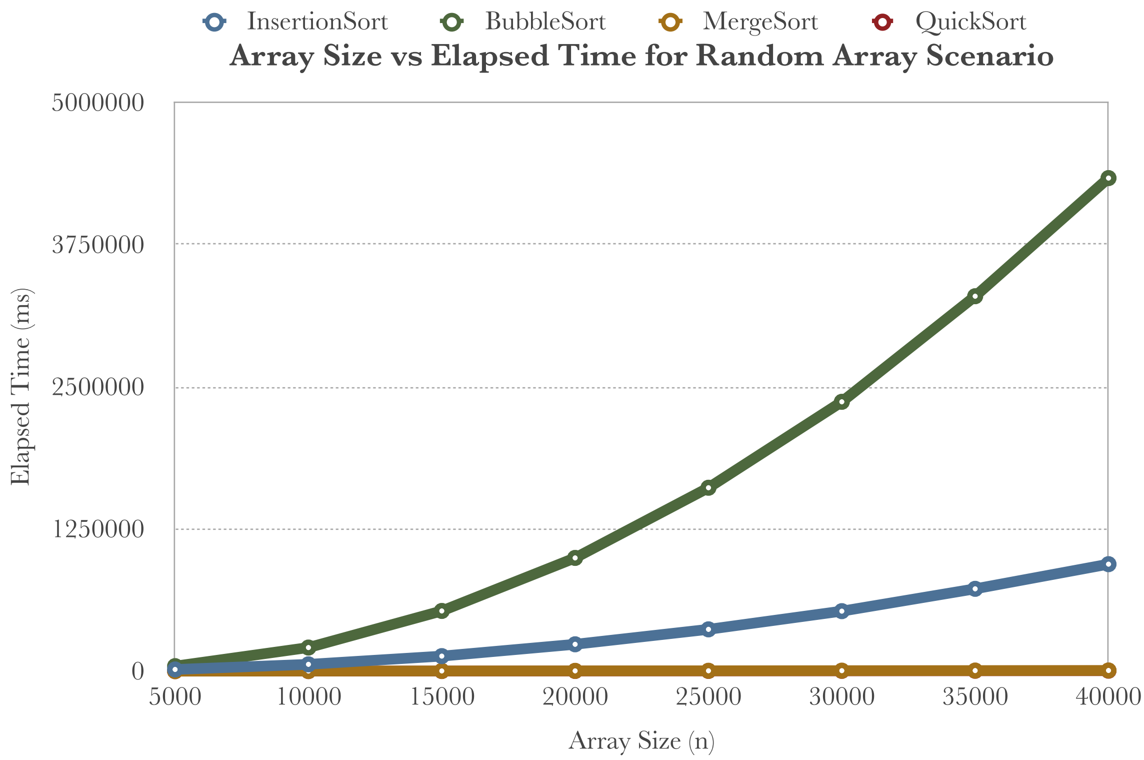
Output 3. Performance Analysis for Almost Sorted Array Scenario

```
-------------------------------------------------------------------
Analysis of Almost Unsorted Array Scenario
-----------------------------------------------------
Analysis of Insertion Sort
Array Size      Elapsed time            compCount               moveCount
5000            28305 ms                11749486                11754526
10000           111034 ms               46965684                46975720
15000           251008 ms               105823860               105838912
20000           447707 ms               187548589               187568608
25000           684111 ms               292448938               292473952
30000           991281 ms               421061959               421091966
35000           1368525 ms              574801767               574836776
40000           1791252 ms              750541408               750581434
-----------------------------------------------------
Analysis of Bubble Sort
Array Size      Elapsed time            compCount               moveCount
5000            57362 ms                12497500                35233584
10000           229604 ms               49995000                140867166
15000           516003 ms               112492500               317426742
20000           930673 ms               199990000               562585830
25000           1452823 ms              312487500               877271862
30000           2077711 ms              449985000               1263095904
35000           2842616 ms              612482500               1724300334
40000           3748185 ms              799979999               -2043462988
-----------------------------------------------------
Analysis of Merge Sort
Array Size      Elapsed time            compCount               moveCount
5000            490 ms          49075           123616
10000           1071 ms         107353          267232
15000           1569 ms         172032          417232
20000           2179 ms         236660          574464
25000           2694 ms         303203          734464
30000           3273 ms         372575          894464
35000           4110 ms         444875          1058928
40000           4563 ms         510406          1228928
-----------------------------------------------------
Analysis of Quick Sort
Array Size      Elapsed time            compCount               moveCount
5000            963 ms          285230          447106
10000           1823 ms         547183          846278
15000           3672 ms         1129081         1732508
20000           2325 ms         640509          1024558
25000           2807 ms         776297          1243358
30000           3299 ms         918893          1411658
35000           3772 ms         1036778         1639937
40000           4946 ms         1390056         2179591
```

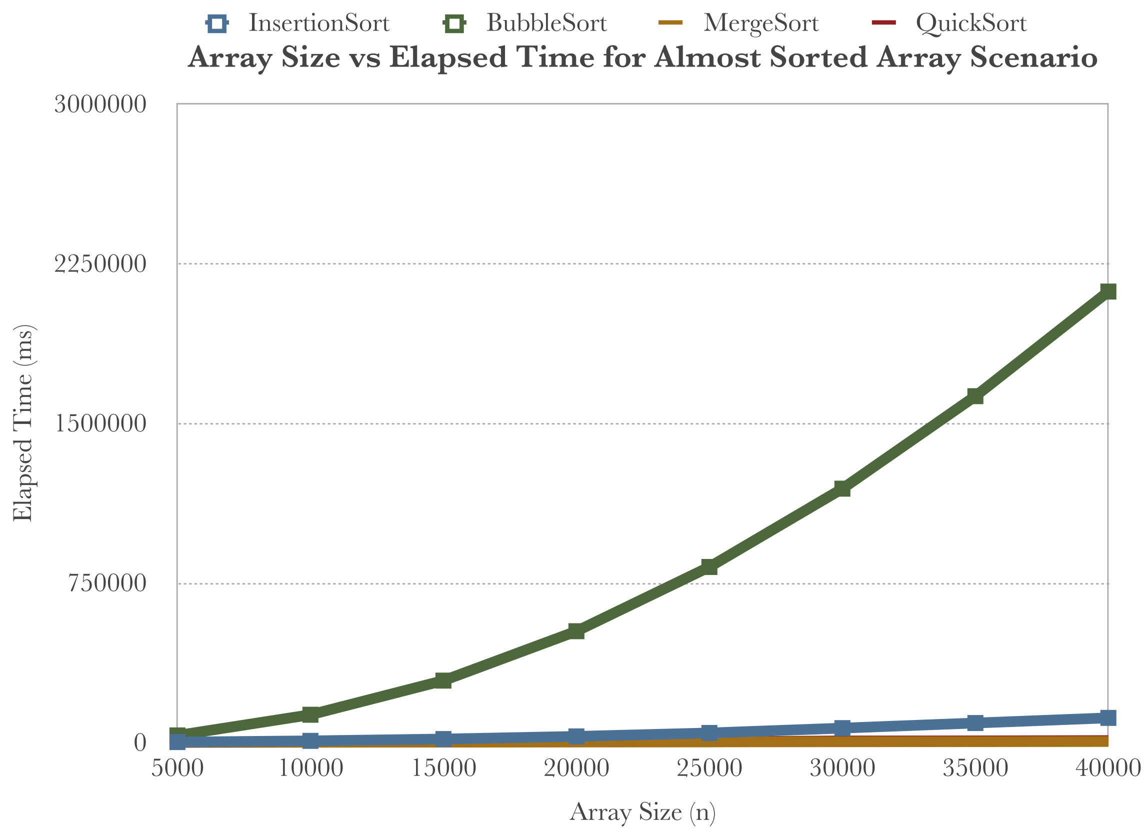
Output 4. Performance Analysis for Almost Unsorted Array Scenario

Question 3

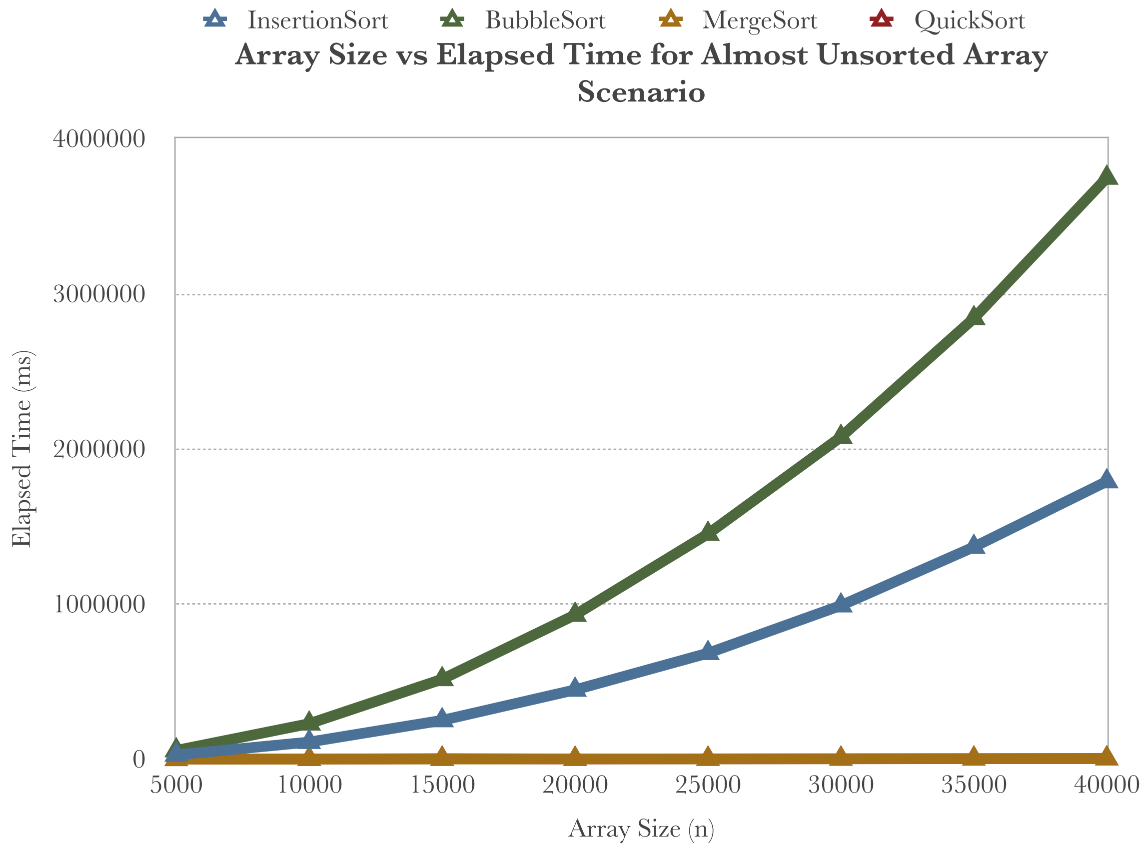
Graphs requested for this part of the homework for random, almost sorted and almost unsorted array scenarios are given below.



Graph 1



Graph 2



Graph 3

While comparing these graphs and sorting algorithms, we need to keep in mind the following time complexities for these algorithms:

- Insertion Sort:
    - Best Case: O(n)
    - Worst Case: O(n^2)
    - Average Case: O(n^2)
- Bubble Sort:
    - Best Case: O(n)
    - Worst Case: O(n^2)
    - Average Case: O(n^2)
- Merge Sort:
    - Best Case: O(nlogn)
    - Worst Case: O(nlogn)
    - Average Case: O(nlogn)

- Quick Sort:
    - Best Case: O(nlogn)
    - Worst Case: $O(n^2)$
    - Average Case: O(nlogn)

Comments:

✓ For the results of insertion sort algorithm, we expect approximately O(n) time complexity for almost sorted array, $O(n^2)$ time complexity for other scenarios (worst and average cases). If we observe the graphs, we can see the dramatic increase with array size for almost unsorted and random array scenarios. It supports theoretical expectations for worst and average cases. If we look at the graph for random array, we can see that the increase with array size in the elapsed time is more passive compared to the other graphs. However, its elapsed time is still larger than the merge sort and quick sort algorithms. This can be due to the fact that even though the array is almost sorted, it is not completely sorted. Therefore, this condition is not sufficient for the best case of insertion sort algorithm, and when we look at the comparison and move counts for this scenario, we can see that its counts are larger than merge sort and quick sort algorithm's.

✓ For the results of the bubble sort algorithm, we expect O(n) time complexity for the best case, $O(n^2)$ time complexity for the worst and average case. When we look at the results bubble sort algorithm is the worst algorithm in terms of runtime. Its comparison and move counts are very large compared to other sorting algorithms, and its time increases dramatically by array size. Even almost sorted array scenario has the same results. Therefore, it can be observed that even if array is slightly unsorted, bubble sort algorithm is badly affected by this and its time complexity increases. Therefore, we cannot observe O(n) time complexity.

✓ For the results of merge sort algorithm, we expect similar results, because theoretical expectation is O(nlogn) for the best, worst, and average cases. When we look at the results, move counts of merge sort algorithm remains same for all algorithms. It is normal because merge sort basically moves all array elements to other small arrays and moves them back into original array in order. Comparison counts change slightly, but it does not cause a dramatic change. Therefore, merge sort has similar appearance on all three graphs. Also, its cost is smaller than insertion and bubble sort, which are $O(n^2)$ sorting algorithms and similar to the quick sort (O(nlogn)) algorithm.

✓ For the results of quick sort algorithm, actually, we expect worst case $O(n^2)$ for almost sorted and almost unsorted array scenarios. Because in these scenarios pivot divides the array into two arrays sized 0 and n-1. However, its behavior is not what we expected. It is still very performant compared to bubble sort and insertion sort algorithms and its performance does not dramatically change with array size. This may show that even if the array is slightly sorted or unsorted, quick sort algorithm still has a good performance. If we compare quick sort algorithm results with merge sort algorithm, we can see that its performance is worse than, even if they have close results, merge sort when array is almost sorted or unsorted. However, its performance is better than merge sort when a random array is used. This behavior is what we expected, because quick sort runs with O(nlogn) in average case like merge sort algorithm and it does not require as much moves or memory allocation as merge sort algorithm.