Title: Sorting and Algorithm Efficiency

Author: Tolga Han Arslan

ID: 22003061

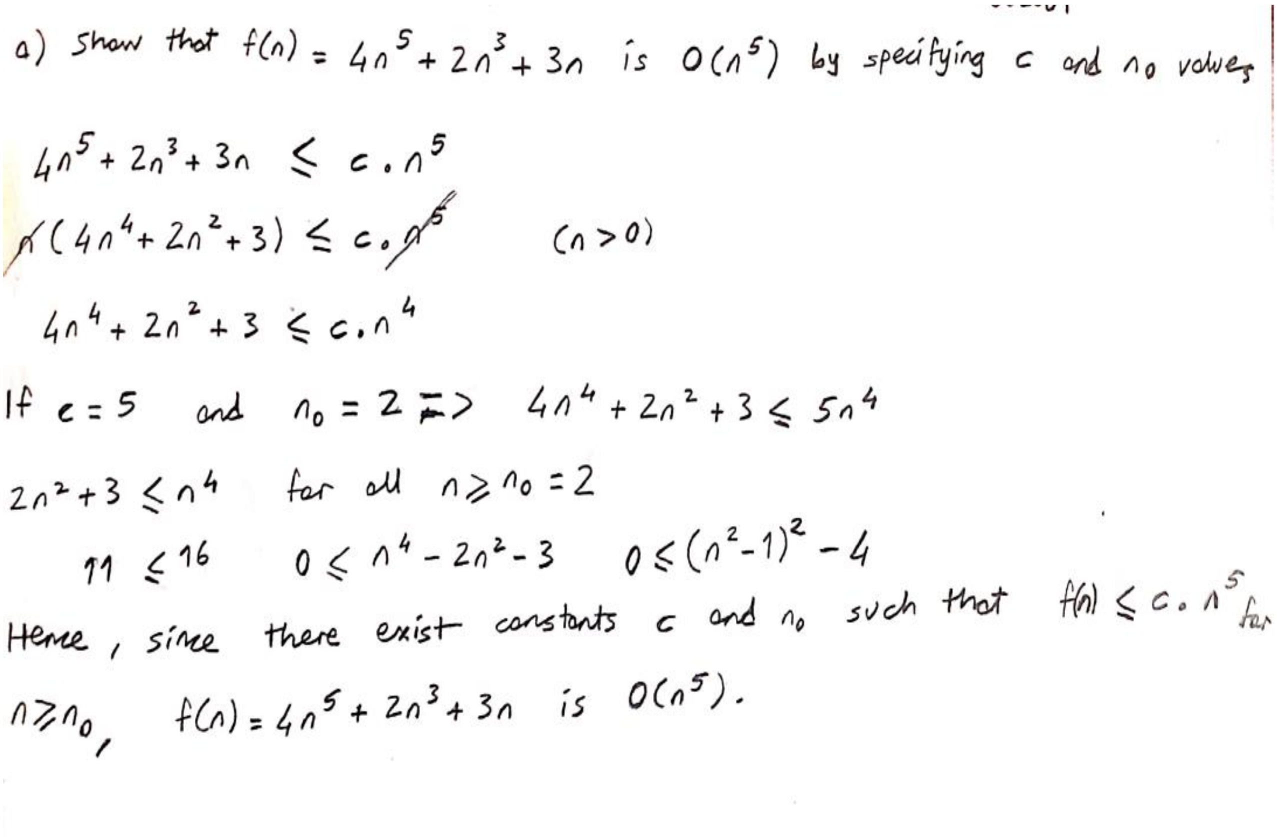
Section: 1

Assignment: 1

Description: HW1 Report

Question 1:

(a) [*5 points*] Show that $f(n) = 4n^5 + 2n^3 + 3n$ is $O(n^5)$ by specifying appropriate c and n_0 values in Big-O definition.



a) Show that $f(n) = 4n^5 + 2n^3 + 3n$ is $O(n^5)$ by specifying c and n_0 values

$4n^5 + 2n^3 + 3n \leq c \cdot n^5$

$\cancel{n}(4n^4 + 2n^2 + 3) \leq c \cdot \cancel{n^5}$ $(n > 0)$

$4n^4 + 2n^2 + 3 \leq c \cdot n^4$

If $c = 5$ and $n_0 = 2 \Rightarrow 4n^4 + 2n^2 + 3 \leq 5n^4$

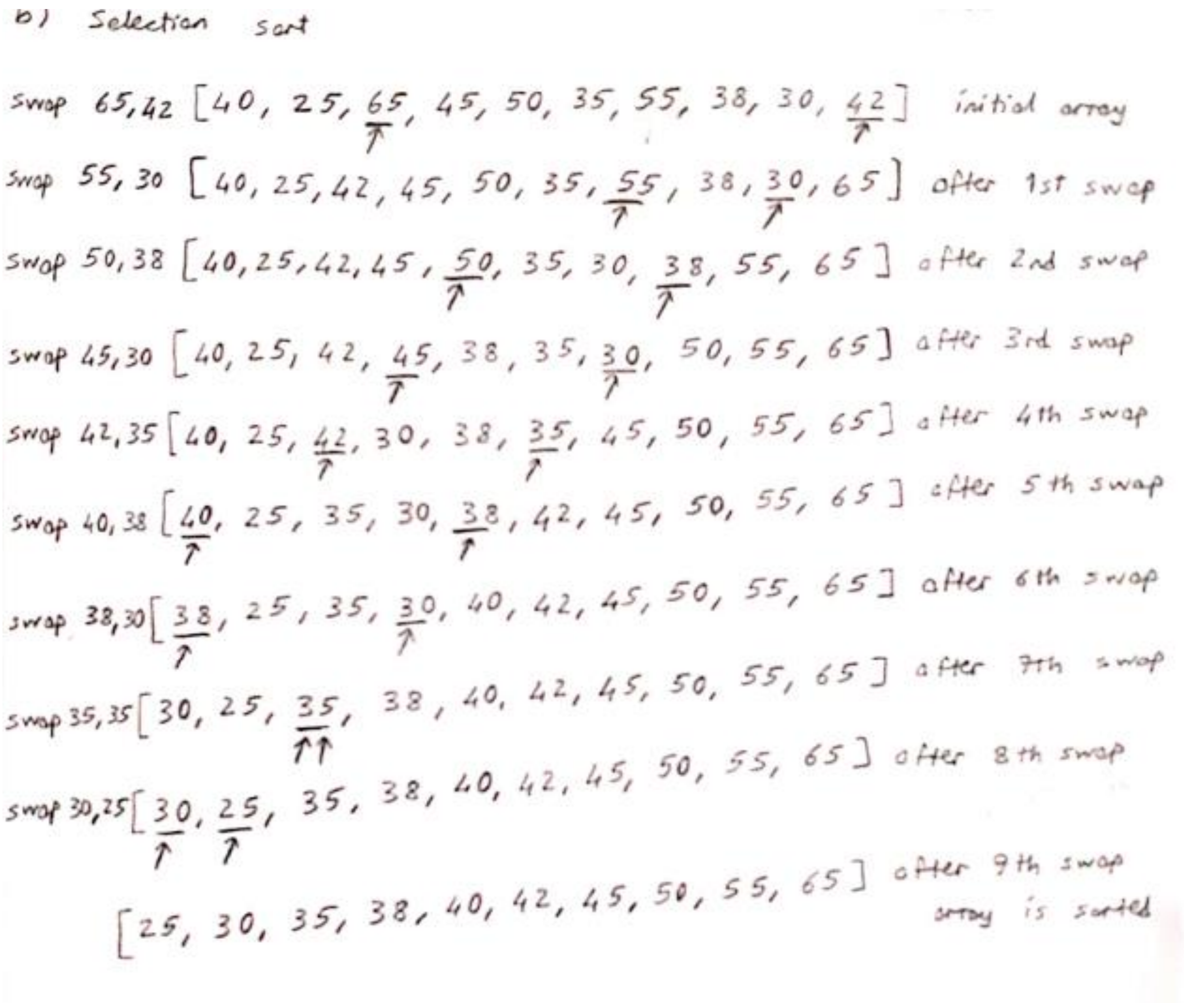
$2n^2 + 3 \leq n^4$ for all $n \geq n_0 = 2$

$11 \leq 16$ $0 \leq n^4 - 2n^2 - 3$ $0 \leq (n^2-1)^2 - 4$

Hence, since there exist constants c and n_0 such that $f(n) \leq c \cdot n^5$ for $n \geq n_0$, $f(n) = 4n^5 + 2n^3 + 3n$ is $O(n^5)$.

(b) [*10 points*] Trace the following sorting algorithms to sort the array [40, 25, 65, 45, 50, 35, 55, 38, 30, 42] in ascending order. Use the array implementation of the algorithms as described in the textbook and show all major steps.

Selection Sort:



b) Selection sort

swap 65,42 [40, 25, 65, 45, 50, 35, 55, 38, 30, 42] initial array

swap 55, 30 [40, 25, 42, 45, 50, 35, 55, 38, 30, 65] after 1st swap

swap 50,38 [40, 25, 42, 45, 50, 35, 30, 38, 55, 65] after 2nd swap

swap 45,30 [40, 25, 42, 45, 38, 35, 30, 50, 55, 65] after 3rd swap

swap 42,35 [40, 25, 42, 30, 38, 35, 45, 50, 55, 65] after 4th swap

swap 40, 38 [40, 25, 35, 30, 38, 42, 45, 50, 55, 65] after 5th swap

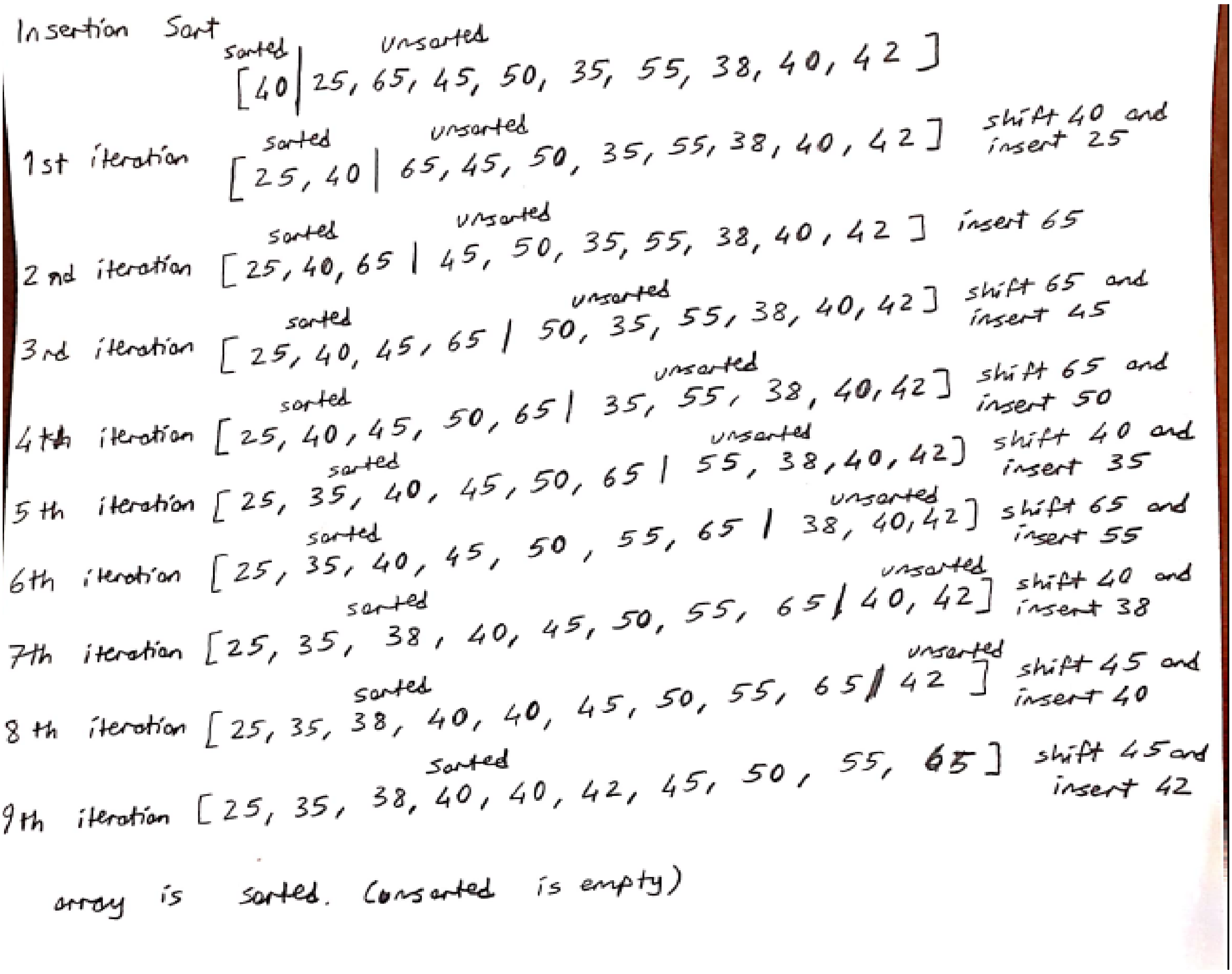
swap 38,30 [38, 25, 35, 30, 40, 42, 45, 50, 55, 65] after 6th swap

swap 35,35 [30, 25, 35, 38, 40, 42, 45, 50, 55, 65] after 7th swap

swap 30,25 [30, 25, 35, 38, 40, 42, 45, 50, 55, 65] after 8th swap

[25, 30, 35, 38, 40, 42, 45, 50, 55, 65] after 9th swap array is sorted

Insertion Sort:



Insertion Sort

sorted | unsorted

[40| 25, 65, 45, 50, 35, 55, 38, 40, 42]

sorted unsorted

1st iteration [25, 40 | 65, 45, 50, 35, 55, 38, 40, 42] shift 40 and insert 25

sorted unsorted

2nd iteration [25, 40, 65 | 45, 50, 35, 55, 38, 40, 42] insert 65

sorted unsorted

3rd iteration [25, 40, 45, 65 | 50, 35, 55, 38, 40, 42] shift 65 and insert 45

sorted unsorted

4th iteration [25, 40, 45, 50, 65 | 35, 55, 38, 40, 42] shift 65 and insert 50

sorted unsorted

5th iteration [25, 35, 40, 45, 50, 65 | 55, 38, 40, 42] shift 40 and insert 35

sorted unsorted

6th iteration [25, 35, 40, 45, 50, 55, 65 | 38, 40, 42] shift 65 and insert 55

sorted unsorted

7th iteration [25, 35, 38, 40, 45, 50, 55, 65 | 40, 42] shift 40 and insert 38

sorted unsorted

8th iteration [25, 35, 38, 40, 40, 45, 50, 55, 65 | 42] shift 45 and insert 40

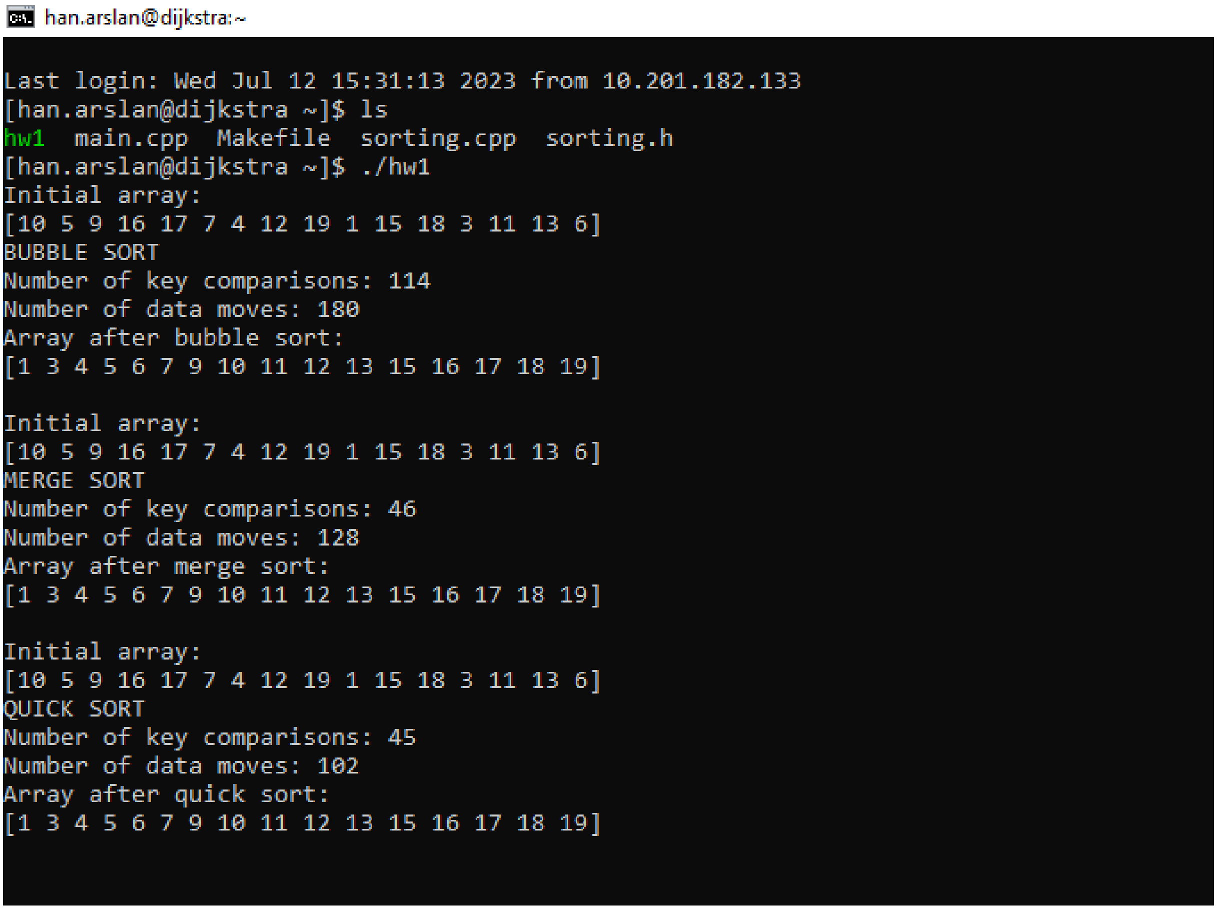
sorted

9th iteration [25, 35, 38, 40, 40, 42, 45, 50, 55, 65] shift 45 and insert 42

array is sorted. (unsorted is empty)

Question 2:

Screenshot of the output in part c:



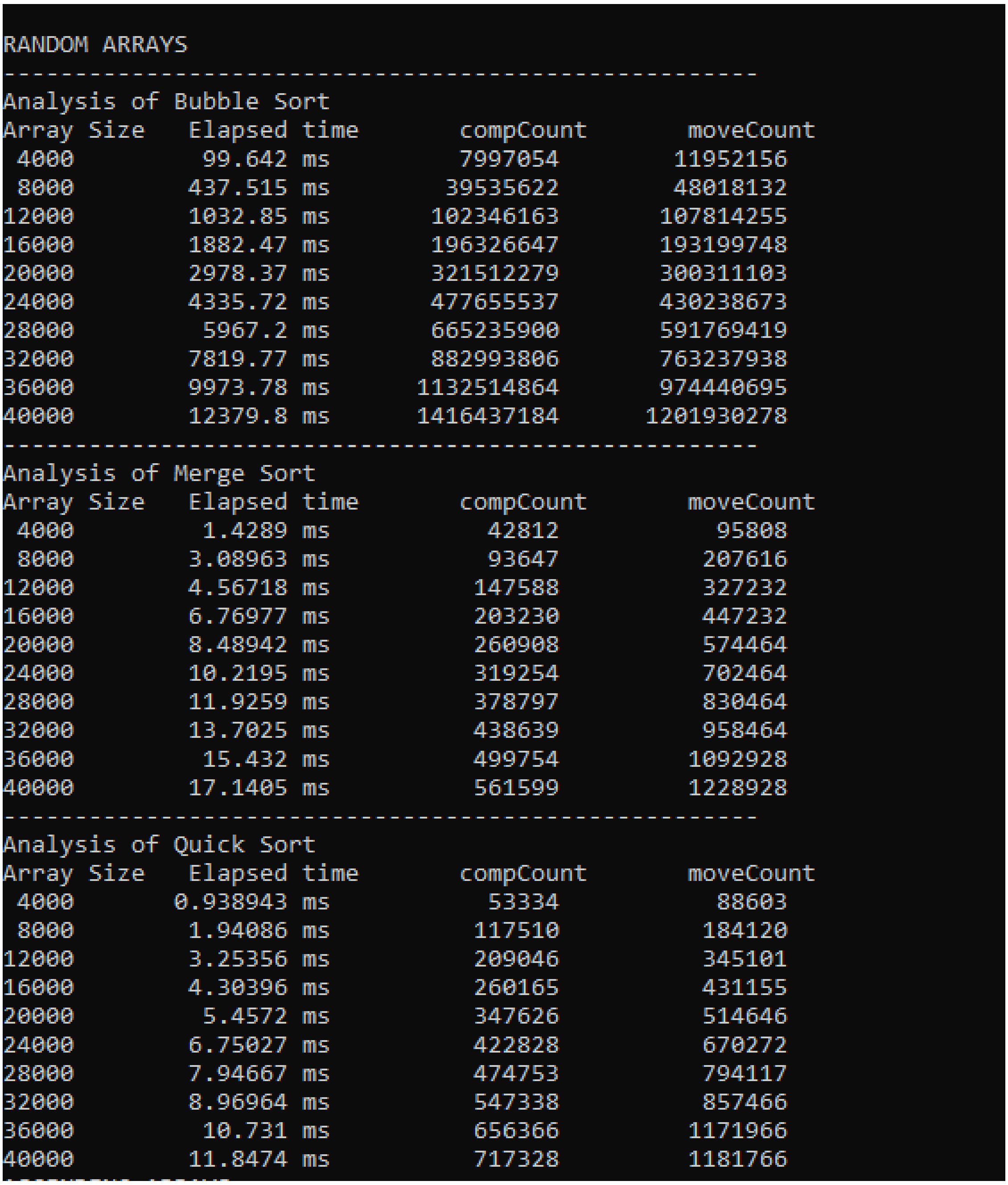
```
Last login: Wed Jul 12 15:31:13 2023 from 10.201.182.133
[han.arslan@dijkstra ~]$ ls
hw1  main.cpp  Makefile  sorting.cpp  sorting.h
[han.arslan@dijkstra ~]$ ./hw1
Initial array:
[10 5 9 16 17 7 4 12 19 1 15 18 3 11 13 6]
BUBBLE SORT
Number of key comparisons: 114
Number of data moves: 180
Array after bubble sort:
[1 3 4 5 6 7 9 10 11 12 13 15 16 17 18 19]

Initial array:
[10 5 9 16 17 7 4 12 19 1 15 18 3 11 13 6]
MERGE SORT
Number of key comparisons: 46
Number of data moves: 128
Array after merge sort:
[1 3 4 5 6 7 9 10 11 12 13 15 16 17 18 19]

Initial array:
[10 5 9 16 17 7 4 12 19 1 15 18 3 11 13 6]
QUICK SORT
Number of key comparisons: 45
Number of data moves: 102
Array after quick sort:
[1 3 4 5 6 7 9 10 11 12 13 15 16 17 18 19]
```

Screenshots of the output in part d:

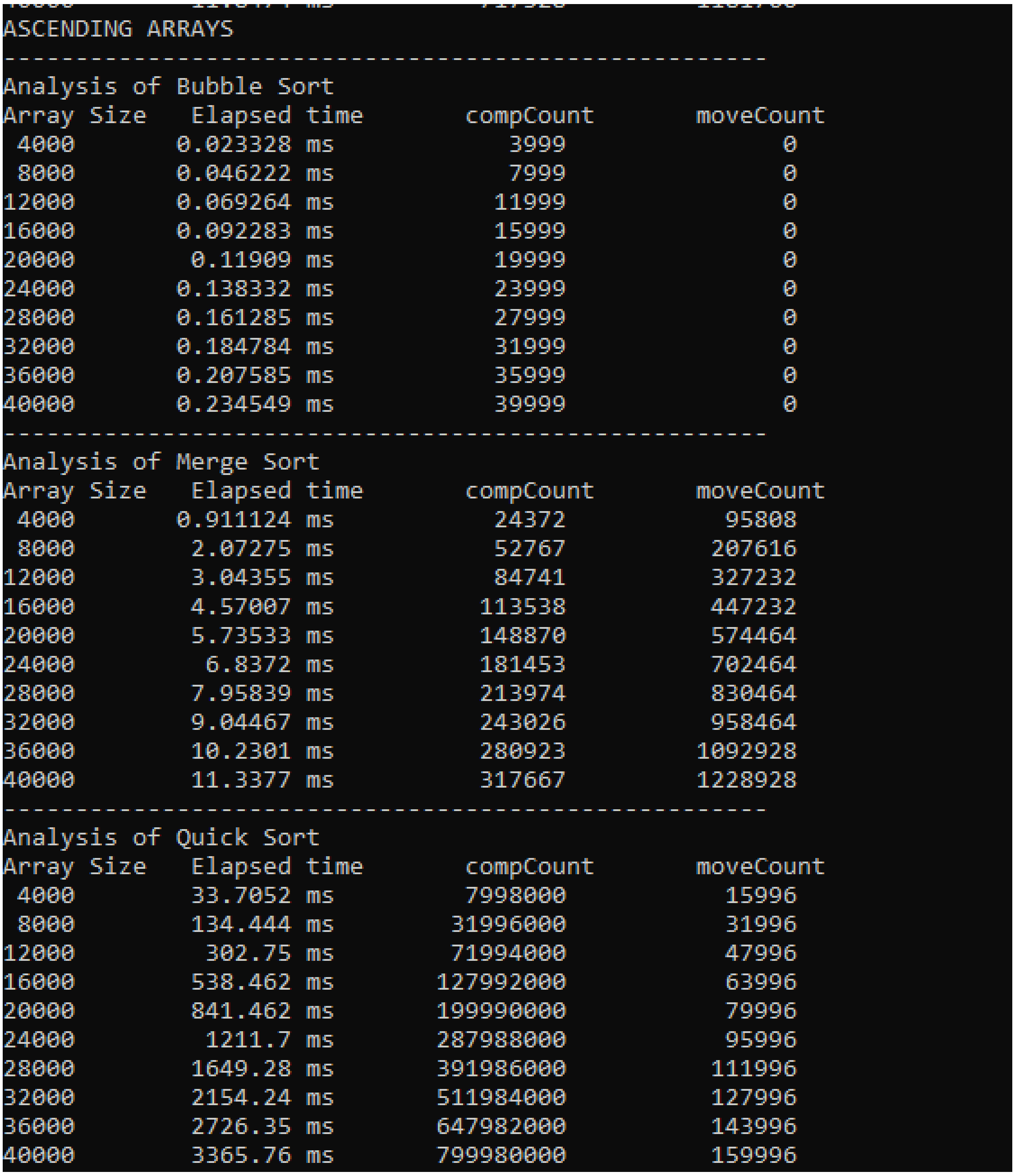
For Random Arrays:



```
RANDOM ARRAYS
-----------------------------------------------------
Analysis of Bubble Sort
Array Size   Elapsed time        compCount        moveCount
 4000          99.642 ms          7997054         11952156
 8000         437.515 ms         39535622         48018132
12000         1032.85 ms        102346163        107814255
16000         1882.47 ms        196326647        193199748
20000         2978.37 ms        321512279        300311103
24000         4335.72 ms        477655537        430238673
28000          5967.2 ms        665235900        591769419
32000         7819.77 ms        882993806        763237938
36000         9973.78 ms       1132514864        974440695
40000         12379.8 ms       1416437184       1201930278
-----------------------------------------------------
Analysis of Merge Sort
Array Size   Elapsed time        compCount        moveCount
 4000          1.4289 ms            42812            95808
 8000         3.08963 ms            93647           207616
12000         4.56718 ms           147588           327232
16000         6.76977 ms           203230           447232
20000         8.48942 ms           260908           574464
24000         10.2195 ms           319254           702464
28000         11.9259 ms           378797           830464
32000         13.7025 ms           438639           958464
36000          15.432 ms           499754          1092928
40000         17.1405 ms           561599          1228928
-----------------------------------------------------
Analysis of Quick Sort
Array Size   Elapsed time        compCount        moveCount
 4000        0.938943 ms            53334            88603
 8000         1.94086 ms           117510           184120
12000         3.25356 ms           209046           345101
16000         4.30396 ms           260165           431155
20000          5.4572 ms           347626           514646
24000         6.75027 ms           422828           670272
28000         7.94667 ms           474753           794117
32000         8.96964 ms           547338           857466
36000          10.731 ms           656366          1171966
40000         11.8474 ms           717328          1181766
```

Output 1: Algorithm comparison with randomly generated arrays

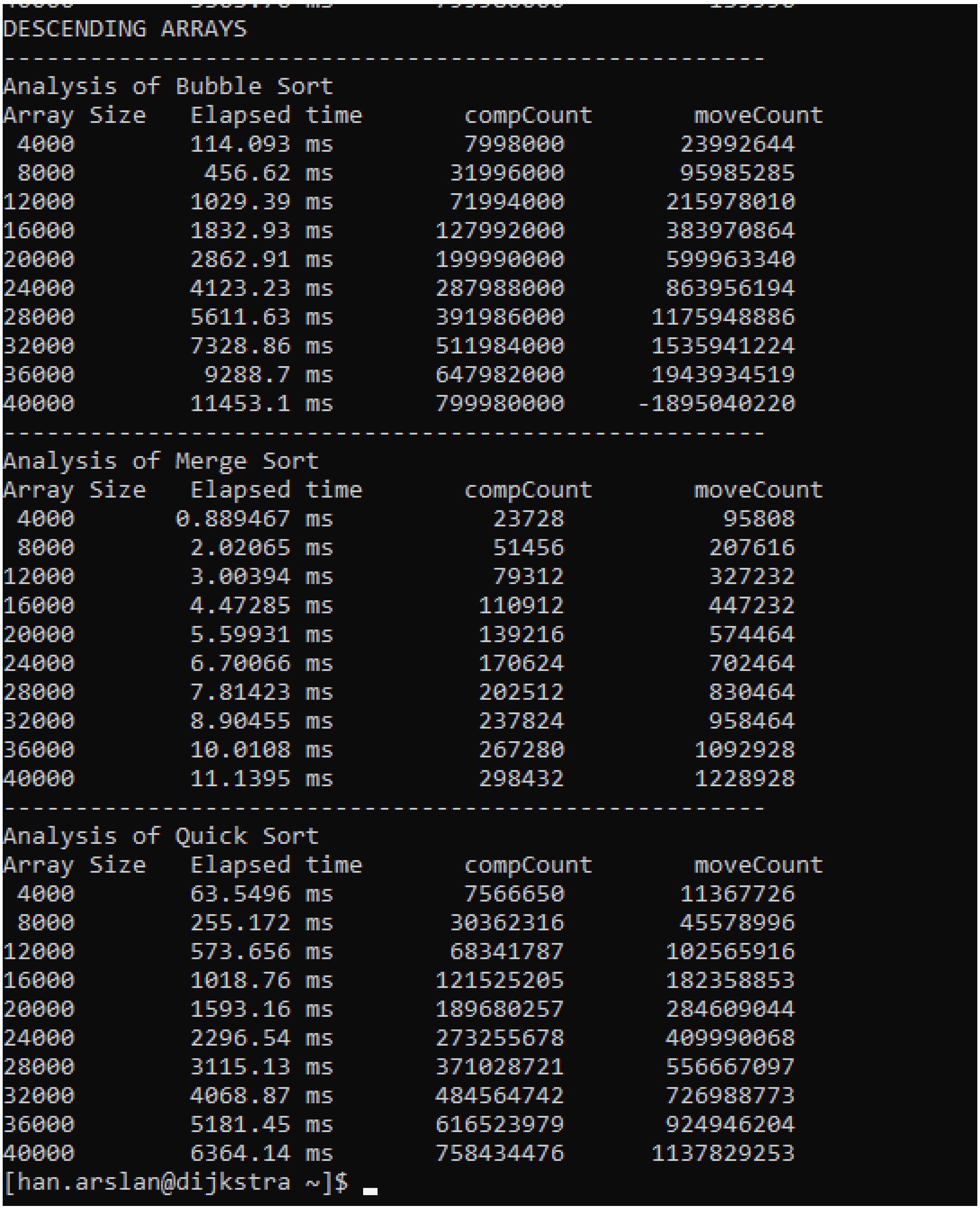
For Ascending Arrays:



```
ASCENDING ARRAYS
-----------------------------------------------------
Analysis of Bubble Sort
Array Size    Elapsed time        compCount       moveCount
 4000        0.023328 ms           3999              0
 8000        0.046222 ms           7999              0
12000        0.069264 ms          11999              0
16000        0.092283 ms          15999              0
20000         0.11909 ms          19999              0
24000        0.138332 ms          23999              0
28000        0.161285 ms          27999              0
32000        0.184784 ms          31999              0
36000        0.207585 ms          35999              0
40000        0.234549 ms          39999              0
-----------------------------------------------------
Analysis of Merge Sort
Array Size    Elapsed time        compCount       moveCount
 4000        0.911124 ms          24372           95808
 8000         2.07275 ms          52767          207616
12000         3.04355 ms          84741          327232
16000         4.57007 ms         113538          447232
20000         5.73533 ms         148870          574464
24000          6.8372 ms         181453          702464
28000         7.95839 ms         213974          830464
32000         9.04467 ms         243026          958464
36000         10.2301 ms         280923         1092928
40000         11.3377 ms         317667         1228928
-----------------------------------------------------
Analysis of Quick Sort
Array Size    Elapsed time        compCount       moveCount
 4000         33.7052 ms        7998000           15996
 8000         134.444 ms       31996000           31996
12000          302.75 ms       71994000           47996
16000         538.462 ms      127992000           63996
20000         841.462 ms      199990000           79996
24000          1211.7 ms      287988000           95996
28000         1649.28 ms      391986000          111996
32000         2154.24 ms      511984000          127996
36000         2726.35 ms      647982000          143996
40000         3365.76 ms      799980000          159996
```

Output 2: Algorithm comparison with ascending arrays

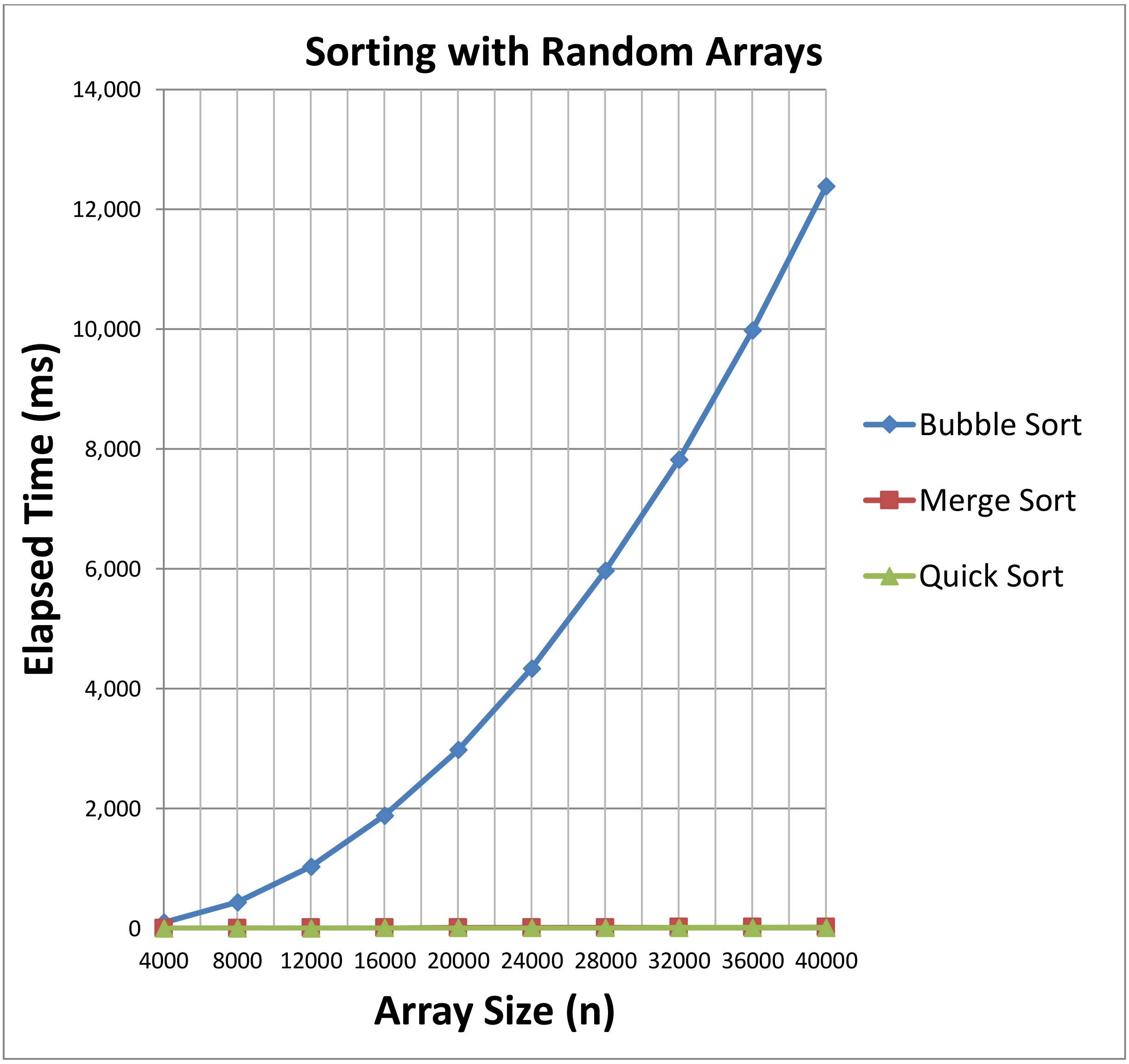
For Descending Arrays:



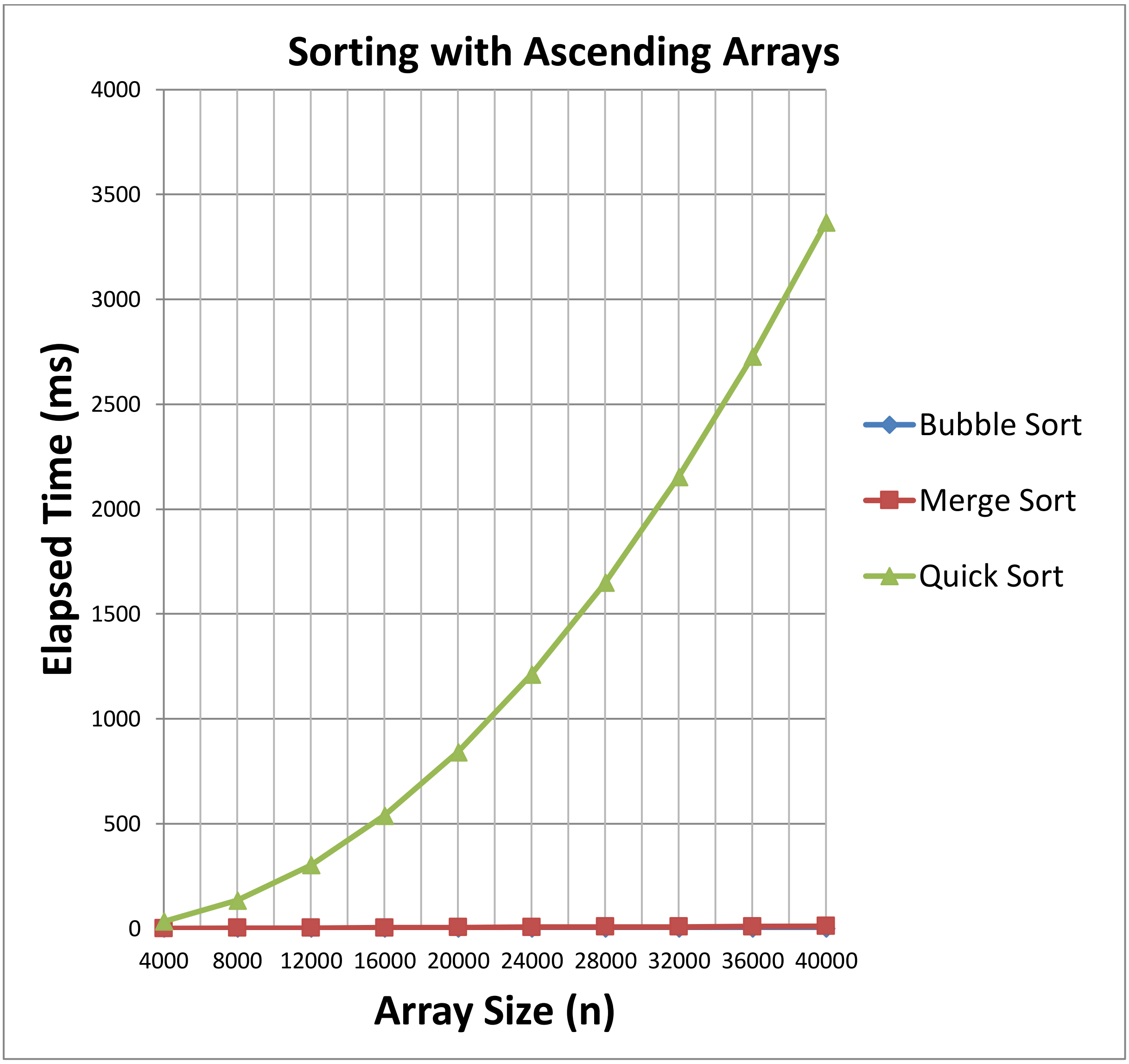
```
DESCENDING ARRAYS
-----------------------------------------------------
Analysis of Bubble Sort
Array Size    Elapsed time        compCount        moveCount
 4000         114.093 ms          7998000          23992644
 8000          456.62 ms         31996000          95985285
12000         1029.39 ms         71994000         215978010
16000         1832.93 ms        127992000         383970864
20000         2862.91 ms        199990000         599963340
24000         4123.23 ms        287988000         863956194
28000         5611.63 ms        391986000        1175948886
32000         7328.86 ms        511984000        1535941224
36000          9288.7 ms        647982000        1943934519
40000         11453.1 ms        799980000       -1895040220
-----------------------------------------------------
Analysis of Merge Sort
Array Size    Elapsed time        compCount        moveCount
 4000        0.889467 ms            23728            95808
 8000         2.02065 ms            51456           207616
12000         3.00394 ms            79312           327232
16000         4.47285 ms           110912           447232
20000         5.59931 ms           139216           574464
24000         6.70066 ms           170624           702464
28000         7.81423 ms           202512           830464
32000         8.90455 ms           237824           958464
36000         10.0108 ms           267280          1092928
40000         11.1395 ms           298432          1228928
-----------------------------------------------------
Analysis of Quick Sort
Array Size    Elapsed time        compCount        moveCount
 4000         63.5496 ms          7566650          11367726
 8000         255.172 ms         30362316          45578996
12000         573.656 ms         68341787         102565916
16000         1018.76 ms        121525205         182358853
20000         1593.16 ms        189680257         284609044
24000         2296.54 ms        273255678         409990068
28000         3115.13 ms        371028721         556667097
32000         4068.87 ms        484564742         726988773
36000         5181.45 ms        616523979         924946204
40000         6364.14 ms        758434476        1137829253
[han.arslan@dijkstra ~]$ 
```

Output 3: Algorithm comparison with descending arrays

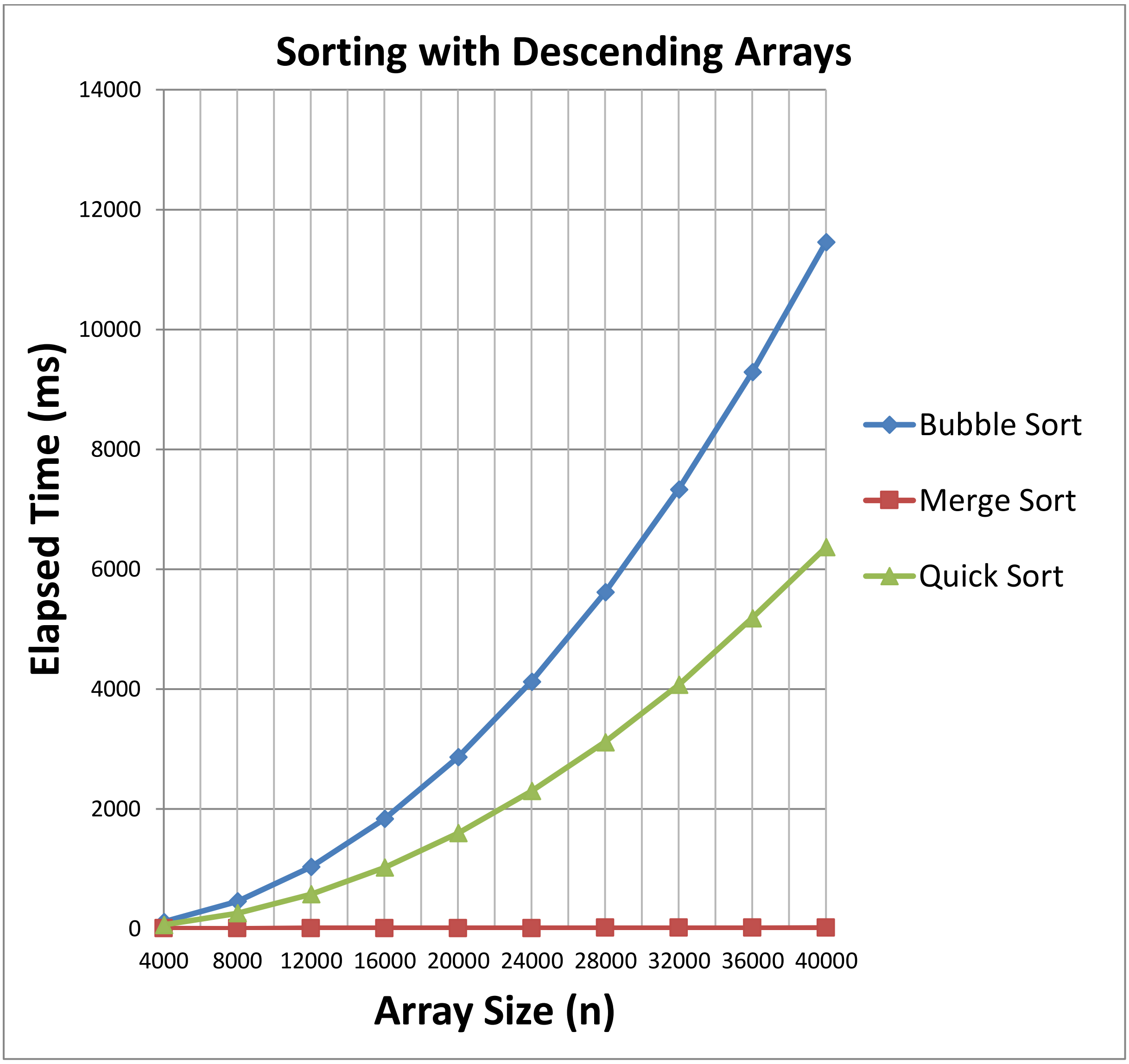
Question 3:



Graph 1: Sorting algorithm comparison with randomly generated arrays



Graph 2: Sorting algorithm comparison with ascending arrays



Graph 3: Sorting algorithm comparison with descending arrays

Comments:

- Bubble sort worst case behaviour is O(n^2) and it happens when the array is in reverse order. By looking at the graph 3, experimental results with descending arrays are consistent with theoretical results.
- Bubble sort average case behaviour is O(n^2), by looking at the graph 1 experimental results are consistent with theoretical results since the arrays are randomly generated.

- Bubble sort makes O(n^2) key comparisons and moves in worst case and average case, as seen in output 3 and output 1.
- Bubble sort best case behaviour is O(n) and it happens when the array is already sorted. This case happens in graph 2, when the array is in ascending order. The moveCount variable is 0 in all array samples, but bubble sort algorithm stil do (n-1) key comparisons as seen in output 2.
- There is integer overflow in descending array analysis with bubble sort of array size 40000, hence moveCount variable is negative.
- Merge sort is O(n*logn) in all cases, independent of the array configuration. Graph 1, 2 and 3 are consistent with this theoretical behaviour.
- Merge sort makes O(n*logn) key comparisons in average case and worst case, hence all three outputs are consistent with this theoretical behaviour.
- Quick sort is O(n*logn) algorithm in best case and average case, results in graph 1 is consistent with this behaviour.
- In worst case, when the array is already sorted or in reverse order, quick sort is O(n^2). Graph 2 and 3 shows this behaviour of quick sort algorithm's worst case.
- Although merge sort is stable and O(n*logn) in every case, quick sort is faster in terms of elapsed time in average case, as seen in output 1. This behaviour is most likely caused by merge sort's requirement of an extra array in its execution.
- Bubble sort's best case behavior O(n) is actually faster than merge sort and quick sort in all cases, as seen in the outputs, however this case rarely occurs in real-life implementations.