CS202
HOMEWORK 1 REPORT
SORTING AND ALGORTIHM EFFICIENCY
2020-2021 Spring

Name: Funda Tan
ID: 21801861
Section: 03

QUESTION 1

(a) Big-O

We should find a pair of c and n-zero for this equations.

$$f(n) = 5n^3 + 4n^2 + 10, n \geq n_0$$

$$5n^3 + 4n^2 + 10 \leq cn^4$$

We can choose c as 5 and n-zero as 2. Doing that, we find a pair and say that this function is O(N^4).

$$c = 5, n_0 = 2$$

(b) Tracing

[24, 8, 51, 28, 20, 29, 21, 17, 38, 27] is our array.

-Insertion sort

INITIAL ARRAY
Key = 8
sorted|unsorted

24	8	51	28	20	29	21	17	38	27

PASS 1
Key = 51
sorted|unsorted

8	24	51	28	20	29	21	17	38	27

PASS 2
Key = 28
sorted|unsorted

8	24	51	28	20	29	21	17	38	27

PASS 3
Key = 20
sorted|unsorted

8	24	28	51	20	29	21	17	38	27

PASS 4
Key = 29

sorted|unsorted

8	20	24	28	51	29	21	17	38	27

PASS 5
Key = 21

sorted|unsorted

8	20	24	28	29	51	21	17	38	27

PASS 6
Key = 17

sorted|unsorted

8	20	21	24	28	29	51	17	38	27

PASS 6
Key =38

sorted|unsorted

8	17	20	21	24	28	29	51	38	27

PASS 7
Key = 27

sorted|sorted

8	17	20	21	24	28	29	38	51	27

SORTED:

sorted|

8	17	20	21	24	27	28	29	38	51

-Bubble sort

INITIAL ARRAY

unsorted|

24	8	51	28	20	29	21	17	38	27

PASS 1

unsorted|

24	8	51	28	20	29	21	17	38	27

unsorted|

8	24	51	28	20	29	21	17	38	27

unsorted|

8	24	51	28	20	29	21	17	38	27

unsorted|

8	24	28	51	20	29	21	17	38	27

unsorted|

8	24	28	20	51	29	21	17	38	27

unsorted|

8	24	28	20	29	51	21	17	38	27

unsorted|

8	24	28	20	29	21	51	17	38	27

unsorted|

8	24	28	20	29	21	17	51	38	27

unsorted|

8	24	28	20	29	21	17	38	51	27

unsorted|sorted

8	24	28	20	29	21	17	38	27	51

PASS 2

unsorted|sorted

8	24	28	20	29	21	17	38	27	51

unsorted|sorted

8	24	28	20	29	21	17	38	27	51

unsorted|sorted

8	24	28	20	29	21	17	38	27	51

unsorted|sorted

8	24	20	28	29	21	17	38	27	51

unsorted|sorted

8	24	20	28	29	21	17	38	27	51

unsorted|sorted

8	24	20	28	21	29	17	38	27	51

unsorted|sorted

8	24	20	28	21	17	29	38	27	51

unsorted|sorted

8	24	20	28	21	17	29	38	27	51

unsorted|sorted

8	24	20	28	21	17	29	27	38	51

PASS 3

unsorted|sorted

8	24	20	28	21	17	29	27	38	51

unsorted|sorted

8	24	20	28	21	17	29	27	38	51

unsorted|sorted

8	20	24	28	21	17	29	27	38	51

unsorted|sorted

8	20	24	28	21	17	29	27	38	51

unsorted|sorted

8	20	24	21	28	17	29	27	38	51

unsorted|sorted

8	20	24	21	17	28	29	27	38	51

unsorted|sorted

8	20	24	21	17	28	29	27	38	51

unsorted|sorted

8	20	24	21	17	28	27	29	38	51

unsorted|sorted

8	20	24	21	17	28	27	29	38	51

PASS 4

unsorted|sorted

8	20	24	21	17	28	27	29	38	51

unsorted|sorted

8	20	24	21	17	28	27	29	38	51

unsorted|sorted

8	20	24	21	17	28	27	29	38	51

unsorted|sorted

8	20	21	24	17	28	27	29	38	51

unsorted|sorted

8	20	21	17	24	28	27	29	38	51

unsorted|sorted

8	20	21	17	24	28	27	29	38	51

unsorted|sorted

8	20	21	17	24	27	28	29	38	51

unsorted|sorted

8	20	21	17	24	27	28	29	38	51

PASS 5

unsorted|sorted

8	20	21	17	24	27	28	29	38	51

unsorted|sorted

8	20	21	17	24	27	28	29	38	51

unsorted|sorted

8	20	21	17	24	27	28	29	38	51

unsorted|sorted

8	20	17	21	24	27	28	29	38	51

unsorted|sorted

8	20	17	21	24	27	28	29	38	51

unsorted|sorted

8	20	17	21	24	27	28	29	38	51

unsorted|sorted

8	20	17	21	24	27	28	29	38	51

PASS 6

unsorted|sorted

8	20	17	21	24	27	28	29	38	51

unsorted|sorted

8	20	17	21	24	27	28	29	38	51

unsorted|sorted

8	17	20	21	24	27	28	29	38	51

unsorted|sorted

8	17	20	21	24	27	28	29	38	51

unsorted|sorted

8	17	20	21	24	27	28	29	38	51

unsorted|sorted

8	17	20	21	24	27	28	29	38	51

PASS 7

unsorted|sorted

8	17	20	21	24	27	28	29	38	51

unsorted|sorted

8	17	20	21	24	27	28	29	38	51

unsorted|sorted

8	17	20	21	24	27	28	29	38	51

unsorted|sorted

8	17	20	21	24	27	28	29	38	51

unsorted|sorted

8	17	20	21	24	27	28	29	38	51

PASS 8

unsorted|sorted

8	17	20	21	24	27	28	29	38	51

unsorted|sorted

8	17	20	21	24	27	28	29	38	51

unsorted|sorted

8	17	20	21	24	27	28	29	38	51

unsorted|sorted

8	17	20	21	24	27	28	29	38	51

PASS 9

unsorted|sorted

8	17	20	21	24	27	28	29	38	51

unsorted|sorted

8	17	20	21	24	27	28	29	38	51

sorted|sorted

8	17	20	21	24	27	28	29	38	51

SORTED:

8	17	20	21	24	27	28	29	38	51

QUESTION 2

Running the executable in Dijkstra server:

```
[funda.tan@dijkstra ~]$ make
g++ sorting.h sorting.cpp main.cpp -o sorting
[funda.tan@dijkstra ~]$ ./sorting
Selection Sort
After selection sorting: 3 5 6 7 8 9 11 12 12 14 14 17 18 19 20 21
Number of key comparisons: 120
Number of data moves: 45

Merge Sort
After merge sorting: 3 5 6 7 8 9 11 12 12 14 14 17 18 19 20 21
Number of key comparisons: 46
Number of data moves: 128

Quick Sort
After quick sorting: 3 5 6 7 8 9 11 12 12 14 14 17 18 19 20 21
Number of key comparisons: 45
Number of data moves: 102

Radix Sort
After radix sorting: 3 5 6 7 8 9 11 12 12 14 14 17 18 19 20 21
```

Running performance analysis method with random ordered arrays in Dijkstra server:

```
EXPERIMENT IN RANDOM ORDERED ARRAY
Wait 10 seconds...
-------------------------------------------------
Analysis of Selection Sort
Array Size      Elapsed time    compCount       moveCount
6000            80              17997000        17997
10000           230             49995000        29997
14000           460             97993000        41997
18000           760             161991000       53997
22000           1130            241989000       65997
26000           1590            337987000       77997
30000           2100            449985000       89997

-------------------------------------------------
Analysis of Merge Sort
Array Size      Elapsed time    compCount       moveCount
6000            1.92308         67827           151616
10000           3.57143         120545          267232
14000           5               175370          387232
18000           5.55556         232044          510464
22000           8.33333         290049          638464
26000           8.33333         349302          766464
30000           10              408667          894464

-------------------------------------------------
Analysis of Quick Sort
Array Size      Elapsed time    compCount       moveCount
6000            1.5625          87053           151863
10000           2.63158         154417          260436
14000           3.125           214928          352888
18000           5.55556         311293          497456
22000           6.25            364933          641792
26000           7.14286         457887          732519
30000           8.33333         530871          904696

-------------------------------------------------
Analysis of Radix Sort
Array Size      Elapsed time
6000            8.33333
10000           12.5
14000           20
18000           25
22000           30
26000           40
30000           45
```

Running performance analysis method with ascending ordered arrays in Dijkstra server:

```
EXPERIMENT IN ASCENDING ARRAY
Wait 10 seconds...
-------------------------------------------------
Analysis of Selection Sort
Array Size      Elapsed time    compCount       moveCount
6000            170             17997000        17997
10000           480             49995000        29997
14000           950             97993000        41997
18000           1580            161991000       53997
22000           2350            241989000       65997
26000           3300            337987000       77997
30000           4360            449985000       89997

-------------------------------------------------
Analysis of Merge Sort
Array Size      Elapsed time    compCount       moveCount
6000            1.12876         39152           151616
10000           1.98413         69008           267232
14000           3.23529         99360           387232
18000           3.96825         130592          510464
22000           5.30303         165024          638464
26000           5.83333         197072          766464
30000           8.33333         227728          894464

-------------------------------------------------
Analysis of Quick Sort
Array Size      Elapsed time    compCount       moveCount
6000            81.5625         17997000        23996
10000           232.632         49995000        39996
14000           463.125         97993000        55996
18000           765.556         161991000       71996
22000           1146.25         241989000       87996
26000           1597.14         337987000       103996
30000           2128.33         449985000       119996

-------------------------------------------------
Analysis of Radix Sort
Array Size      Elapsed time
6000            3.24074
10000           8.75
14000           12.5
18000           18.3333
22000           25
26000           60
30000           55
```

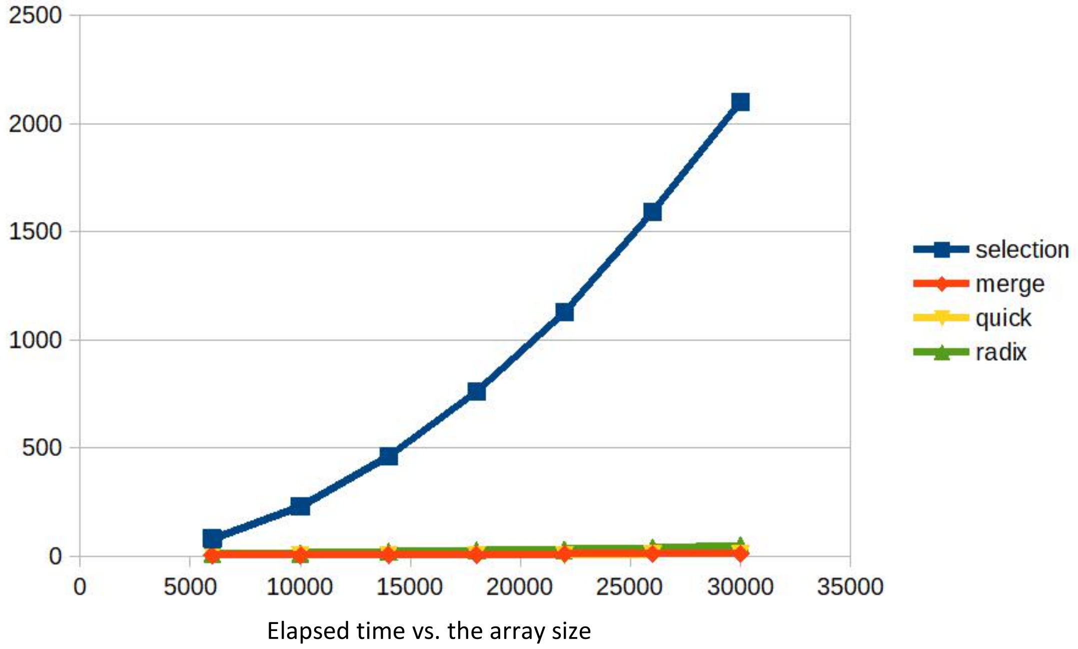
Running performance analysis method with descending ordered arrays in Dijkstra server:

```
EXPERIMENT IN DESCENDING ARRAY
Wait 10 seconds...
-------------------------------------------------
Analysis of Selection Sort
Array Size      Elapsed time    compCount       moveCount
6000            250             17997000        17997
10000           720             49995000        29997
14000           1430            97993000        41997
18000           2370            161991000       53997
22000           3530            241989000       65997
26000           4940            337987000       77997
30000           6540            449985000       89997

-------------------------------------------------
Analysis of Merge Sort
Array Size      Elapsed time    compCount       moveCount
6000            1.21735         36656           151616
10000           1.99939         64608           267232
14000           2.66176         94256           387232
18000           3.59788         124640          510464
22000           4.60859         154208          638464
26000           5.58333         186160          766464
30000           7.29167         219504          894464

-------------------------------------------------
Analysis of Quick Sort
Array Size      Elapsed time    compCount       moveCount
6000            241.562         17997000        27023996
10000           672.632         49995000        75039996
14000           1333.12         97993000        147055996
18000           2205.56         161991000       243071996
22000           3296.25         241989000       363087996
26000           4597.14         337987000       507103996
30000           6128.33         449985000       675119996

-------------------------------------------------
Analysis of Radix Sort
Array Size      Elapsed time
6000            2.66204
10000           5.875
14000           10.5
18000           14.5833
22000           27.5
26000           80
30000           75
```

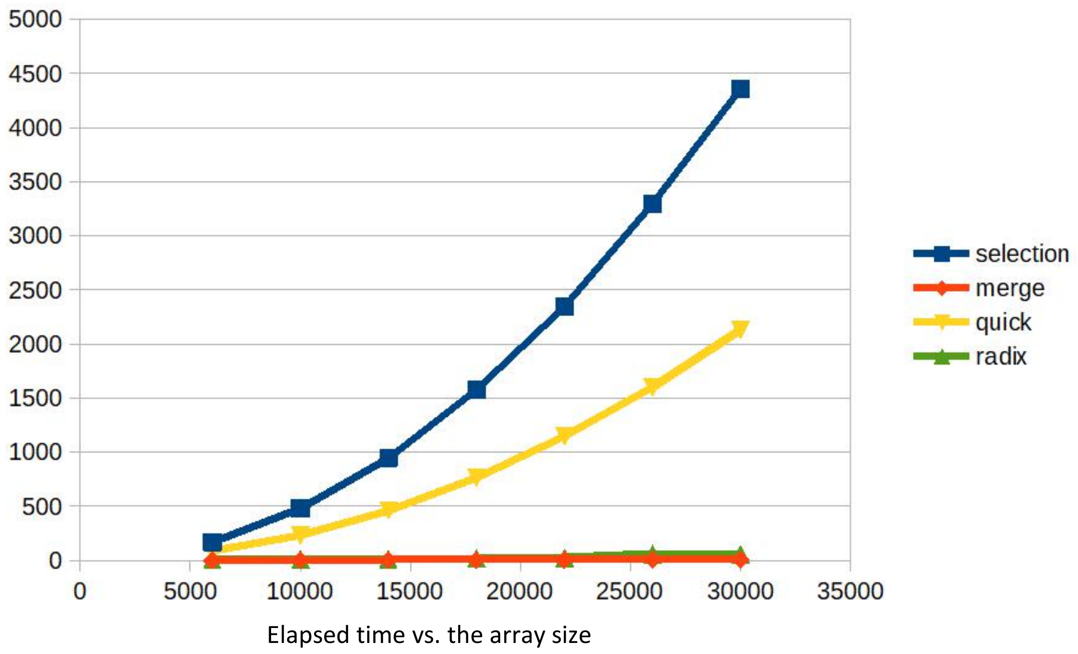
QUESTION 3
Experimental Results Obtained from Question 2

Graphs

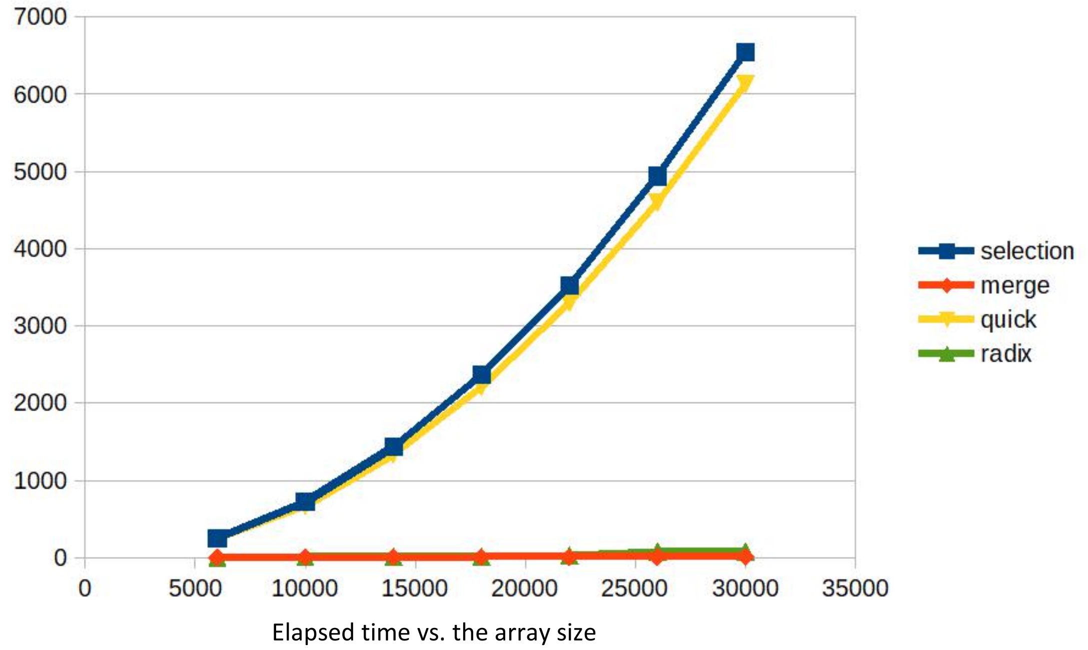
Random Ordered Array



Ascending Ordered Array



Descending Ordered Array



Discussion

Comparison between experimental results and theoretical results:

Theoretical Results

- Average Case

Selection Sort = O(n^2)
Merge Sort = O(nlogn)
Quick Sort = O(nlogn)
Radix Sort = O(n)

- Worst Case

Selection Sort = O(n^2)
Merge Sort = O(nlogn)
Quick Sort = O(n^2)
Radix Sort = O(n)

Experimental Results

In theoretical results, we see that selection sort is O(n^2) and this result matches with the empirical results that we see in the graphs. For merge sort and radix sort, in the graphs they seem as they are similar, but in theory, merge sort is O(nlogn) and radix sort is O(n). We cannot see this clearly in the graphs because of the scale of the graphs. Also, we can see quick sort's average case and worst case as in descending ordered it is similar to selection sort (O(n^2)) and in random ordered, it is similar to O(nlogn).

If we choose already sorted arrays instead of random ordered arrays:

In the graphs, we see that merge, selection and radix sort does not depend on different ordering of the arrays, their time complexity came out similar in the graphs. Choosing already sorted arrays instead of random ordered arrays does not made any difference in selection, merge and radix sort but it made a great difference in quick sort.

As we see in the graphs, worst case time complexity for quick sort with **already sorted arrays case** came out similar as the selection sort with already sorted arrays. For the **random ordered arrays case**, quick sort was similar to merge and radix sort. So we can say from the graphs that empirical results of quick sort matches with the theoretical results as it's average case is similar to merge and radix sort (O(nlogn)) and it's worst case is similar to selection sort (O(n^2)).