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.

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

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

 $[24, 8, 51, 28, 20, 29, {}_{2}1, 17, 38, 27] is our array.$

-Insertion sort

INITIAL ARRAY

Key = 8

sorted unsorted

201100									
24	8	51	28	20	29	21	17	38	27

PASS 1

Key = 51

sorted	unsorted								
24	51	28	20	20	21	17	38	27	7

PASS 2

Key = 28

8 24 51 28 20 29 21 17 38 27			sorted	unsortea						
	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 $Key = 2$	9								
,				sorted	unsorted				
8	20	24	28	51	29	21	17	38	27
PASS 5 Key = 2	1				sorted	l unsorted			
8	20	24	28	29	51	21	17	38	27
O	20		20		J 1	21	1 /		21
PASS 6 Key = 1	7					sorte	d unsorted		
8	20	21	24	28	29	51	17	38	27
PASS 6 Key =38								unsorted	
8	17	20	21	24	28	29	51	38	27
PASS 7 Key = 2	7	20	21	24	28	29	38	sorted 51	sorted
0	1 /		21			2)] 30	J 1	21
SORTE									sorted
SORTE:	D:	20	21	24	27	28	29	38	sorted 51
-Bubble	17 sort L ARRAY	7						u	nsorted
-Bubble	17	•	21	24	27	28	29		51
-Bubble INITIAL 24 PASS 1	sort ARRAY	51	28	20	29	21	17	u 38	nsorted 27
-Bubble INITIAI	17 sort L ARRAY	7						u 38	nsorted 27
-Bubble INITIAL 24 PASS 1	sort ARRAY	51	28	20	29	21	17	u 38 u 38	nsorted 27
-Bubble INITIAL 24 PASS 1	sort ARRAY	51	28	20	29	21	17	u 38 u 38	nsorted 27 nsorted 27
-Bubble INITIAL 24 PASS 1 24	17 sort ARRAY 8	51 51	28	20 20 20	29	21	17	u 38 u 38 u 38	nsorted 27 nsorted 27 nsorted 27
-Bubble INITIAL 24 PASS 1	17 sort ARRAY 8	51	28	20	29	21	17	u 38 u 38 u 38	nsorted 27 nsorted 27 nsorted 27 27
-Bubble INITIAL 24 PASS 1 24 8	17	51 51 51	28 28 28	20 20 20	29 29 29	21 21 21 21	17 17 17	u 38 u 38 u 38 u 38 u 4	51
-Bubble INITIAL 24 PASS 1 24	17 2 sort 2 ARRAY 8	51 51	28	20 20 20	29 29 29	21 21 21	17 17	38 u 38 u 38 u 38	51
-Bubble INITIAL 24 PASS 1 24 8	17	51 51 51	28 28 28	20 20 20	29 29 29	21 21 21 21	17 17 17	38 u 38 u 38 u 38	51

unsorted

8	24	28	20	29	51	21	17	38	27
								u	nsorted
8	24	28	20	29	21	51	17	38	27
	1	ı		1	I				nsorted
8	24	28	20	29	21	17	51	38	27
0	21	20	20	2)	21	17	<i>J</i> 1] 30	21
								u	nsorted
8	24	28	20	29	21	17	38	51	27
								unsorte	d sorted
8	24	28	20	29	21	17	38	27	51
PASS 2				ı	ı		1	ı	
0		• •							d sorted
8	24	28	20	29	21	17	38	27	51
								unsorte	d sorted
8	24	28	20	29	21	17	38	27	51
						1 - 7	1 20		
								unsorte	d sorted
8	24	28	20	29	21	17	38	27	51
								unsorte	d sorted
8	24	20	28	29	21	17	38	27	51
								unsorte	d sorted
8	24	20	28	29	21	17	38	27	51
								unsorte	d sorted
8	24	20	28	21	29	17	38	27	51
				,			•	unsorte	d sorted
8	24	20	28	21	17	29	38	27	51
		,	,	,	,		,	unsorte	d sorted
8	24	20	28	21	17	29	38	27	51
		ı		1	ı	1	unsorted		
8	24	20	28	21	17	29	27	38	51
PASS 3					1		unsorted		
8	24	20	28	21	17	29	27	38	51
		<u> </u>		_ = =	<u>, </u>		unsorted		,
8	24	20	28	21	17	29	27	38	51
					1 *'		unsorted		
8	20	24	28	21	17	29	27	38	51
		27	20	<u>~1</u>	1 /				<i>J</i> 1

							unsorted	sorted	
8	20	24	28	21	17	29	27	38	51
							unsorted	sorted	
8	20	24	21	28	17	29	27	38	51
					1	_,			
	 						unsorted		
8	20	24	21	17	28	29	27	38	51
							unsorted	sorted	
8	20	24	21	17	28	29	27	38	51
						,			<u>. </u>
0	20	2.4	21	1.7	20		unsorted		7.1
8	20	24	21	17	28	27	29	38	51
						uns	sorted sor	ted	
8	20	24	21	17	28	27	29	38	51
PASS 4						1100	sorted sor	ted	
8	20	24	21	17	28	27	29	38	51
0				1,	1 20	27			
							sorted sor		
8	20	24	21	17	28	27	29	38	51
						une	sorted sor	ted	
8	20	24	21	17	28	27	29	38	51
					• 0		sorted sor		
8	20	21	24	17	28	27	29	38	51
						uns	sorted sor	ted	
8	20	21	17	24	28	27	29	38	51
					•				<u>. </u>
0	20	21	1.7	24	20		sorted sor		5.1
8	20	21	17	24	28	27	29	38	51
						uns	sorted sor	ted	
8	20	21	17	24	27	28	29	38	51
									_
8	20	21	17	24	unsorte 27	ed sorted 28	29	38	51
0		<u> </u>	1/	<u> </u>		20	29	36	31
PASS 5									
			1	1 -	1	ed sorted		Γ -	
8	20	21	17	24	27	28	29	38	51
					linsort	ed sorted			
8	20	21	17	24	27	28	29	38	51
L				1	· · · · · · · · · · · · · · · · · · ·	<u>.</u>			
	20		4 =		1	ed sorted	20	20	
8	20	21	17	24	27	28	29	38	51

					unsorte	ed sorted			
8	20	17	21	24	27	28	29	38	51
	ı			,	•	'		'	ı
						dsorted			
8	20	17	21	24	27	28	29	38	51
						114.1			
8	20	17	21	24	27	d sorted	29	38	51
	20	1 /	21	27	21	20	2)		31
				unso	rted sorted	1			
8	20	17	21	24	27	28	29	38	51
PASS 6					. 11				
O	20	17	21	unso 24	rted sorted	28	29	38	51
8	20	1 /	21	24	21	20	29	36	51
				unso	rted sorted	1			
8	20	17	21	24	27	28	29	38	51
					1				
				unso	rted sorted	1			
8	17	20	21	24	27	28	29	38	51
					4 11 4	1			
8	17	20	21	unso 24	rted sorted	28	29	38	51
0	1 /	20	21	24	2.1	20	23	36	31
				unso	rted sorted	1			
8	17	20	21	24	27	28	29	38	51
			unsorted						
8	17	20	21	24	27	28	29	38	51
PASS 7									
TASS /			unsorted	sorted					
8	17	20	21	24	27	28	29	38	51
			1		1	'		'	
			unsorted	İ				1	
8	17	20	21	24	27	28	29	38	51
			. 1	1					
8	17	20	unsorted 21	sorted 24	27	28	29	38	51
O	1 /	20	21	∠ +		20	<i>43</i>		J1
			unsorted	sorted					
8	17	20	21	24	27	28	29	38	51
		•						•	
			ted sorted						
8	17	20	21	24	27	28	29	38	51
PASS 8									
rass 8		lineor	ted sorted						
8	17	20	21	24	27	28	29	38	51
9	- 1								, J

unsorted | sorted

8	17	20	21	24	27	28	29	38	51
		unsort	ed sorted						
8	17	20	21	24	27	28	29	38	51
	4-1	14.1							
8	unsorted 17	sorted 20	21	24	27	28	29	38	51
	1/	20	<u> </u>	<u> </u>	21	28		38	31
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
SORTE	D:								
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:

	ANDOM ORDERED AR	RAY	
Wait 10 seconds			
Analysis of Sel			
10 (10 m to	Elapsed time		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 Mer	ge Sort		
Array Size		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
30000	± 0	400001	001101
Analysis of Qui	ck Sort		
Array Size		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
	7.14286		
26000		457887	732519
30000	8.33333	530871	904696
7			
Analysis of Rad			
Array Size			
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:

	IN ASCENDING ARRAY		
Wait 10 seco	onds		
	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		
	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	
30000	8.33333	221128	894464
Analysis of			1000
Array Size			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		
30000			

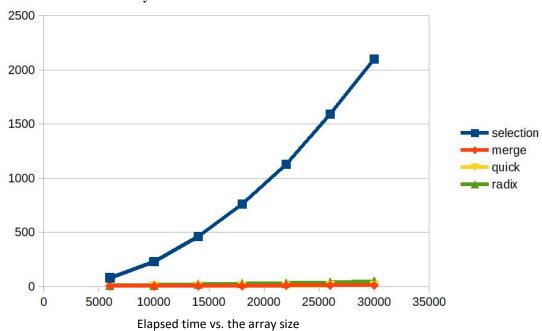
Running performance analysis method with descending ordered arrays in Dijkstra server:

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
30000	00001
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
7,23107 213001	031101
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	

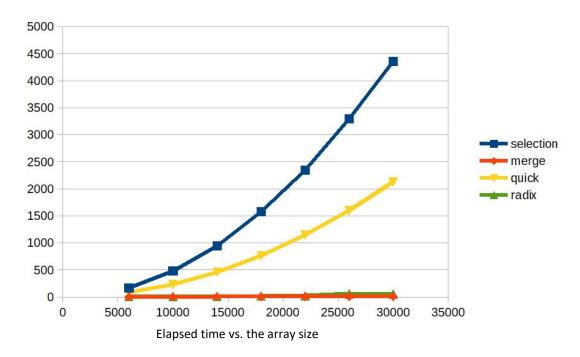
QUESTION 3Experimental Results Obtained from Question 2

Graphs

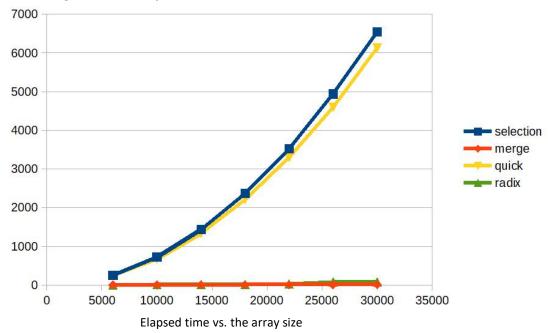
Random Ordered Array



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 $O(n\log n)$ 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 $O(n\log n)$.

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 $(O(n\log n))$ and it's worst case is similar to selection sort $(O(n^2))$.