2021-2022 Fall Semester

Title: Algorithm Efficiency and Sorting

Author: Alper Mumcular

ID: 21902740

Section: 1

Assignment: 1

Description: Report

Question 1

a)

1- $T(n) = 3\,T(n/3) + n$

$= 9\,T(n/9) + 2n$

$= 27\,T(n/27) + 3n$

…

$= 3^k\,T(n/3^k) + kn$ $\quad 3^k = n \Rightarrow k = \log_3 n$

$= n\,T(n/n) + n\log_3 n$

$= n\,T(1) + n\log_3 n$

$= O(n*\log_3 n) = O(n*\log_2 n)$

2- $T(n) = 2\,T(n-1) + n^2$

$= 4\,T(n-2) + 2n^2$

$= 8\,T(n-3) + 3n^2$

…

$= 2^k\,T(n-k) + kn^2$ $\quad n = k + 1$

$= 2^{n-1}\,T(1) + (n-1)*n^2$

$= O(2^{n-1} + (n-1)*n^2) = O(2^n)$

3- $T(n) = 3\,T(n/4) + n*\log n$

$= 9\,T(n/16) + 2*n*\log n$

$= 27\,T(n/64) + 3*n*\log n$

…

$= 3^k\,T(n/4^k) + k*n*\log n$ $\quad n = 4^k \Rightarrow k = \log_2 n$

$= 3^{\log_4 n}\,T(1) + \log_4 n*n*\log n$

$= O(3^{\log_4 n} + \log_4 n*n*\log n) = O(n*\log n*\log n)$

4- $T(n) = 3\,T(n/2) + 1$

$= 9\,T(n/4) + 2$

$= 27\,T(n/8) + 3$

…

$= 3^k\,T(n/2^k) + k \qquad n = 2^k \Rightarrow k = \log_2 n$

$= 3^{\log_2 n}\,T(1) + \log_2 n$

$= O(\,3^{\log_2 n} + \log_2 n\,) \Rightarrow O(3^{\log_2 n}) = O(n^{\log_2 3}) = O(n^2)$

b)

Bubble Sort:

1-	5	6	8	4	10	2	9	1	3	7 |
2-	5	6	8	4	10	2	9	1	3	7 |
3-	5	6	8	4	10	2	9	1	3	7 |
4-	5	6	4	8	10	2	9	1	3	7 |
5-	5	6	4	8	10	2	9	1	3	7 |
6-	5	6	4	8	2	10	9	1	3	7 |
7-	5	6	4	8	2	9	10	1	3	7 |
8-	5	6	4	8	2	9	1	10	3	7 |
9-	5	6	4	8	2	9	1	3	10	7 |
10-	5	6	4	8	2	9	1	3	7 |	10
11-	5	6	4	8	2	9	1	3	7 |	10
12-	5	6	4	8	2	9	1	3	7 |	10
13-	5	4	6	8	2	9	1	3	7 |	10
14-	5	4	6	8	2	9	1	3	7 |	10
15-	5	4	6	2	8	9	1	3	7 |	10

16- 5 4 6 2 8 9 1 3 7 | 10

17- 5 4 6 2 8 1 9 3 7 | 10

18- 5 4 6 2 8 1 3 9 7 | 10

19- 5 4 6 2 8 1 3 7 | 9 10

20- 5 4 6 2 8 1 3 7 | 9 10

21- 4 5 6 2 8 1 3 7 | 9 10

22- 4 5 6 2 8 1 3 7 | 9 10

23- 4 5 2 6 8 1 3 7 | 9 10

24- 4 5 2 6 8 1 3 7 | 9 10

25- 4 5 2 6 1 8 3 7 | 9 10

26- 4 5 2 6 1 3 8 7 | 9 10

27- 4 5 2 6 1 3 7| 8 9 10

28- 4 5 2 6 1 3 7| 8 9 10

29- 4 5 2 6 1 3 7| 8 9 10

30- 4 2 5 6 1 3 7| 8 9 10

31- 4 2 5 6 1 3 7| 8 9 10

32- 4 2 5 1 6 3 7| 8 9 10

33- 4 2 5 1 3 6 7| 8 9 10

34- 4 2 5 1 3 6 | 7 8 9 10

35- 4 2 5 1 3 6 | 7 8 9 10

36- 2 4 5 1 3 6 | 7 8 9 10

37- 2 4 5 1 3 6 | 7 8 9 10

38- 2 4 1 5 3 6 | 7 8 9 10

39- 2 4 1 3 5 6 | 7 8 9 10

40- 2 4 1 3 5 | 6 7 8 9 10

41- 2 4 1 3 5 | 6 7 8 9 10

42- 2 4 1 3 5 | 6 7 8 9 10

43- 2 1 4 3 5 | 6 7 8 9 10

44- 2 1 3 4 5 | 6 7 8 9 10

45- 2 1 3 4| 5 6 7 8 9 10

46- 2 1 3 4| 5 6 7 8 9 10

47- 1 2 3 4| 5 6 7 8 9 10

48- 1 2 3 4| 5 6 7 8 9 10

49- 1 2 3| 4 5 6 7 8 9 10

50- 1 2 3| 4 5 6 7 8 9 10

51- 1 2 3| 4 5 6 7 8 9 10

52- 1 2| 3 4 5 6 7 8 9 10

53- 1 2| 3 4 5 6 7 8 9 10

54- 1| 2 3 4 5 6 7 8 9 10

55- |1 2 3 4 5 6 7 8 9 10

Selection Sort:

1- 5 6 8 4 10 2 9 1 3 7 | largest = 5

2- 5 6 8 4 10 2 9 1 3 7 | largest = 6

3- 5 6 8 4 10 2 9 1 3 7 | largest = 8

4- 5 6 8 4 10 2 9 1 3 7 | largest = 8

5- 5 6 8 4 10 2 9 1 3 7 | largest = 10

6- 5 6 8 4 10 2 9 1 3 7 | largest = 10

7- 5 6 8 4 10 2 9 1 3 7 | largest = 10

8-	5	6	8	4	10	2	9	1	3	7 |		largest = 10
9-	5	6	8	4	10	2	9	1	3	7 |		largest = 10
10-	5	6	8	4	10	2	9	1	3	7 |		largest = 10
11-	5	6	8	4	7	2	9	1	3|	10		swap
12-	5	6	8	4	7	2	9	1	3|	10		largest = 5
13-	5	6	8	4	7	2	9	1	3|	10		largest = 6
14-	5	6	8	4	7	2	9	1	3|	10		largest = 8
15-	5	6	8	4	7	2	9	1	3|	10		largest = 8
16-	5	6	8	4	7	2	9	1	3|	10		largest = 8
17-	5	6	8	4	7	2	9	1	3|	10		largest = 8
18-	5	6	8	4	7	2	9	1	3|	10		largest = 9
19-	5	6	8	4	7	2	9	1	3|	10		largest = 9
20-	5	6	8	4	7	2	9	1	3|	10		largest = 9
21-	5	6	8	4	7	2	3	1|	9	10		largest = 9
22-	5	6	8	4	7	2	3	1|	9	10		swap
23-	5	6	8	4	7	2	3	1|	9	10		largest = 6
24-	5	6	8	4	7	2	3	1|	9	10		largest = 8
25-	5	6	8	4	7	2	3	1|	9	10		largest = 8
26-	5	6	8	4	7	2	3	1|	9	10		largest = 8
27-	5	6	8	4	7	2	3	1|	9	10		largest = 8
28-	5	6	8	4	7	2	3	1|	9	10		largest = 8
29-	5	6	8	4	7	2	3	1|	9	10		largest = 8
30-	5	6	1	4	7	2	3|	8	9	10		swap
31-	5	6	1	4	7	2	3|	8	9	10		largest = 5

32- 5 6 1 4 7 2 3| 8 9 10 largest = 6

33- 5 6 1 4 7 2 3| 8 9 10 largest = 6

34- 5 6 1 4 7 2 3| 8 9 10 largest = 6

35- 5 6 1 4 7 2 3| 8 9 10 largest = 7

36- 5 6 1 4 7 2 3| 8 9 10 largest = 7

37- 5 6 1 4 7 2 3| 8 9 10 largest = 7

38- 5 6 1 4 3 2| 7 8 9 10 swap

39- 5 6 1 4 3 2| 7 8 9 10 largest = 5

40- 5 6 1 4 3 2| 7 8 9 10 largest = 6

41- 5 6 1 4 3 2| 7 8 9 10 largest = 6

42- 5 6 1 4 3 2| 7 8 9 10 largest = 6

43- 5 6 1 4 3 2| 7 8 9 10 largest = 6

44- 5 6 1 4 3 2| 7 8 9 10 largest = 6

45- 5 2 1 4 3| 6 7 8 9 10 swap

46- 5 2 1 4 3| 6 7 8 9 10 largest = 5

47- 5 2 1 4 3| 6 7 8 9 10 largest = 5

48- 5 2 1 4 3| 6 7 8 9 10 largest = 5

49- 5 2 1 4 3| 6 7 8 9 10 largest = 5

50- 5 2 1 4 3| 6 7 8 9 10 largest = 5

51- 3 2 1 4| 5 6 7 8 9 10 swap

52- 3 2 1 4| 5 6 7 8 9 10 largest = 3

53- 3 2 1 4| 5 6 7 8 9 10 largest = 3

54- 3 2 1 4| 5 6 7 8 9 10 largest = 3

55- 3 2 1 4| 5 6 7 8 9 10 largest = 4

56- 3 2 1| 4 5 6 7 8 9 10 swap

57- 3 2 1| 4 5 6 7 8 9 10 largest = 3

58- 3 2 1| 4 5 6 7 8 9 10 largest = 3

59- 3 2 1| 4 5 6 7 8 9 10 largest = 3

60- 1 2| 3 4 5 6 7 8 9 10 swap

61- 1 2| 3 4 5 6 7 8 9 10 largest = 1

62- 1 2| 3 4 5 6 7 8 9 10 largest = 2

63- 1| 2 3 4 5 6 7 8 9 10 swap

64- 1| 2 3 4 5 6 7 8 9 10 largest = 1

65- |1 2 3 4 5 6 7 8 9 10

c)

The worst case is that the pivot divides the list of size n into two sublists of sizes 0 and n-1.

$T(N) = T(N-1) + T(0) + N$ [$T(0) = 1$]

$= T(N-1) + N + 1$

$= T(N-2) + 2\ (N+1)$

$= T(N-3) + 3\ (N+1)$

…

$= T(N-k) + k\ (N+1)$ $k = n$

$= T(N-N) + N\ (N+1)$

$= T(0) + N\ (N+1)$

$= O(1+N^2+N) = O(N^2)$ → Worst case.

Question 2

```
Array: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
Array: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
Array: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
Array: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
-------------------------------------------------------
Part a - Time Analysis of Insertion Sort
Array Size             Time Elapsed          compCount          moveCount
2000                        4.00 ms           90696              91056
6000                       29.00 ms           810552             811632
10000                      80.22 ms           2223585            2225385
14000                     171.67 ms           4383166            4385686
18000                     259.67 ms           7268937            7272177
22000                     420.33 ms           10876430           10880390
26000                     525.67 ms           15111241           15115921
30000                     618.89 ms           20223960           20229359
-------------------------------------------------------
Part b - Time Analysis of Merge Sort
Array Size             Time Elapsed          compCount          moveCount
2000                        0.35 ms           19443              34072
6000                        2.12 ms           67841              117171
10000                       3.74 ms           120379             206031
14000                       4.89 ms           175491             298101
18000                       6.64 ms           231768             392814
22000                       7.98 ms           289984             490712
26000                      10.78 ms           348936             589161
30000                      12.32 ms           408568             687936
-------------------------------------------------------
Part c - Time Analysis of Quick Sort
Array Size             Time Elapsed          compCount          moveCount
2000                        0.22 ms           11242              39034
6000                        0.73 ms           50597              167823
10000                       1.25 ms           77624              259524
14000                       1.92 ms           120730             399606
18000                       2.52 ms           151327             502345
22000                       3.12 ms           176089             587511
26000                       3.63 ms           218898             726630
30000                       4.44 ms           247254             822846
-------------------------------------------------------
Part d - Time Analysis of Radix Sort
Array Size             Time Elapsed
2000                        0.68 ms
6000                        1.94 ms
10000                       3.11 ms
14000                       4.63 ms
18000                       5.83 ms
22000                       7.64 ms
26000                       8.62 ms
30000                       9.76 ms

--------------------------------
Process exited after 30.09 seconds with return value 0
Press any key to continue . . .
```

Question 3

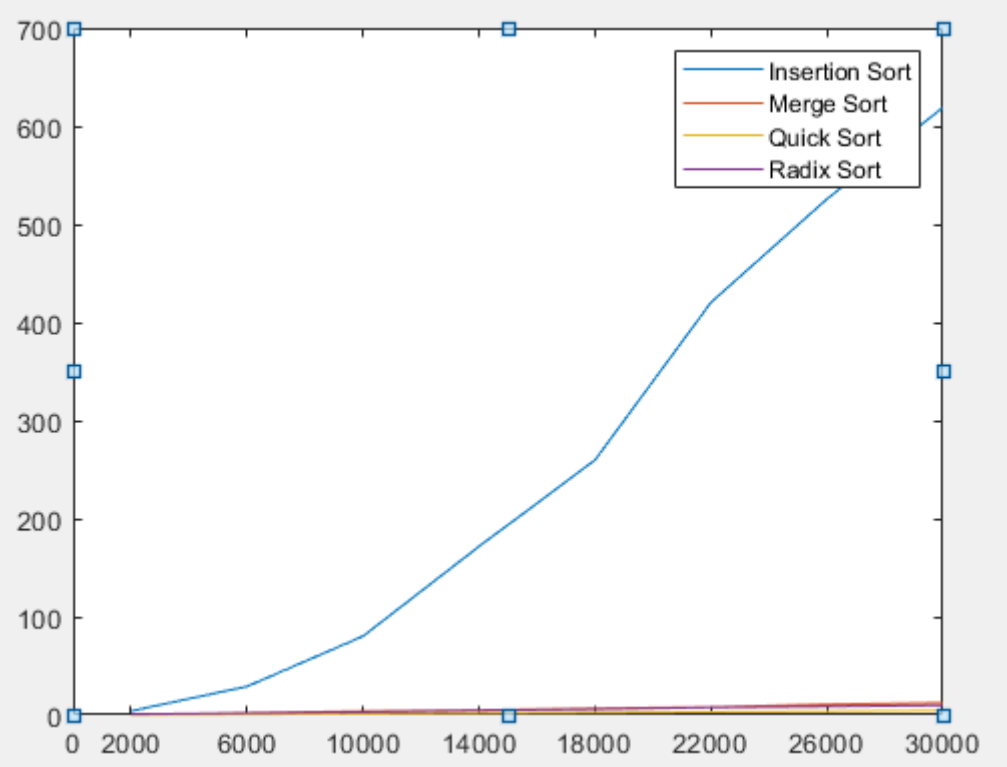

Figure 1: Graph of 4 Sorting Algorithms

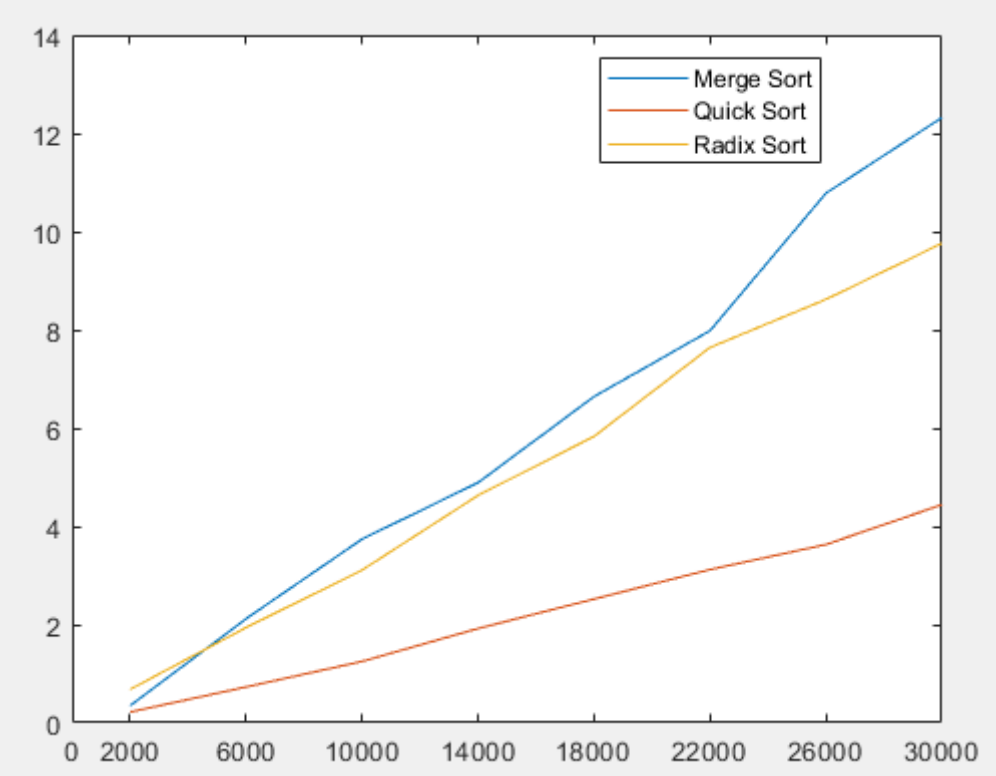

Figure 2: Graph of Sorting Algorithms Except Insertion Sort

Comments

*****We can see that Insertion Sort requires much more time than the other three sorting algorithms in Graph 1.

*****Graph of Insertion Sort is similar to $O(n^2)$, elapsed time of Merge Sort is similar to O(n*logn), elapsed time of Quick Sort is similar to O(n*logn), elapsed time of Radix Sort is similar to O(n). According to the graph, making comments of other sorting algorithms except Insertion Sort is hard.

*****Theoretically, Radix Sort should be the most efficient sorting algorithm, but empirically, Quick Sort requires less time than the others.

*****According to the experiment, Quick Sort is better than the Merge Sort whereas both are O(n*logn). The reason of this can be Merge Sort is not an in-place algorithm and requires extra memory. Creating extra memory will increase the elapsed time in Merge Sort.

Question: How would the time complexity of your program change if you applied the sorting algorithms to an array of increasing numbers instead of randomly generated numbers?

***** Insertion Sort: Array is already sorted. Therefore, inner loop will not be executed. For this reason, Insertion Sort will be O(n). In my program, the elapsed time of Insertion sort will decrease.

***** Merge Sort: Best case of Merge Sort is O(n*logn). Therefore, elapsed time will not be affected too much, but it will decrease because in merge operations, one array will be placed in the original array then all elements in the other array will be placed in the original array.

***** Quick Sort: This will be the worst case of Quick Sort because pivot element divides the list of size n into two sublists of sizes zero and n-1. Therefore, time complexity of Quick Sort will become $O(n^2)$. Hence, the running time of Quick Sort will increase.

***** Radix Sort: Time complexity of Radix Sort is O(n). The elapsed time in Radix Sort will not be affected too much because Radix Sort does not use key comparisons to sort an array.