*HW1-Algorithm Efficiency and Sorting

*Alper Kandemir

*21703062

1) Find the asymptotic running times in big O notation of the following recurrence equations by using the repeated substitution method.

-T(n) = 3T (n/3) + n, where T (1) = 1 and n is an exact power of 3.

T (n/3) = 3T(n/9) + n/3

T(n) = 3(3T(n/9) + n/3) + n = 3^2 T(n/9) + n + n

T(n/9) = 3T (n/27) + n/9

T(n) = 3^2(3T(n/27) + n/9) + n + n = 3^3 T(n/27) + n + n + n

…

3^k T(n/3^k) +kn

T (1) = 1 so, n/3^k =1, k=$\log_3 n$

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

T(n)=O(nlogn)

-T(n) = 3T (n/2) + 1, where T (1) = 1 and n is an exact power of 2.

T(n/2) = 3T(n/4) + 1

T(n) = 3^2T (n/4) + 3 + 1

T(n/4) = T(n/8) + 1

T(n) = 3^3T (n/8) + 3^2 + 3 + 1

….

3^k T(n/2^k) + (3^k-1 +3 ^k-2 +… 3 + 1)

n/2^k=1, k=$\log_2 n$

T(n) = (1+3+3^2+ …+ 3^k-1)

T(n) = ((3^k – 1)/ 3-1)

$T(n) = 1/2\,(3^{\log_2 n} - 1)$

$T(n) = 1/2\,(n^{\log_2 3} - 1)$

Complexity is $O(n^{\log_2 3})$

1-1) Sort the array [5, 6, 8, 4, 10, 2, 9, 1, 3, 7] in ascending order.

-) Bubble Sort:

1st:

<u>5, 6</u>, 8, 4, 10, 2, 9, 1, 3, 7

5, <u>6, 8</u>, 4, 10, 2, 9, 1, 3, 7

5, 6, <u>8, 4</u>, 10, 2, 9, 1, 3, 7

5, 6, 4, 8, 10, 2, 9, 1, 3, 7

5, 6, 4, <u>8, 10</u>, 2, 9, 1, 3, 7

5, 6, 4, 8, <u>10, 2</u>, 9, 1, 3, 7

5, 6, 4, 8, 2, 10, 9, 1, 3, 7

5, 6, 4, 8, 2, <u>10, 9</u>, 1, 3, 7

5, 6, 4, 8, 2, 9, 10, 1, 3, 7

5, 6, 4, 8, 2, 9, <u>10, 1</u>, 3, 7

5, 6, 4, 8, 2, 9, 1, 10, 3, 7

5, 6, 4, 8, 2, 9, 1, <u>10, 3</u>, 7

5, 6, 4, 8, 2, 9, 1, 3, 10, 7

5, 6, 4, 8, 2, 9, 1, 3, <u>10, 7</u>

5, 6, 4, 8, 2, 9, 1, 3, 7, |10

2nd:

<u>5, 6</u>, 4, 8, 2, 9, 1, 3, 7, |10

5, <u>6, 4</u>, 8, 2, 9, 1, 3, 7, |10

5, 4, 6, 8, 2, 9, 1, 3, 7, |10

5, 4, <u>6, 8</u>, 2, 9, 1, 3, 7, |10

5, 4, 6, 8, 2, 9, 1, 3, 7, |10

5, 4, 6, 2, 8, 9, 1, 3, 7, |10

5, 4, 6, 2, 8, 9, 1, 3, 7, |10

5, 4, 6, 2, 8, 9, 1, 3, 7, |10

5, 4, 6, 2, 8, 1, 9, 3, 7, |10

5, 4, 6, 2, 8, 1, 9, 3, 7, |10

5, 4, 6, 2, 8, 1, 3, 9, 7, |10

5, 4, 6, 2, 8, 1, 3, 9, 7, |10

5, 4, 6, 2, 8, 1, 3, 7, 9, |10

5, 4, 6 , 2, 8, 1, 3, 7, |9, 10

3rd:

5, 4, 6, 2, 8, 1, 3, 7, |9, 10

4, 5, 6, 2, 8, 1, 3, 7, |9, 10

4, 5, 6, 2, 8, 1, 3, 7, |9, 10

4, 5, 6, 2, 8, 1, 3, 7, |9, 10

4, 5, 2, 6, 8, 1, 3, 7, |9, 10

4, 5, 2, 6, 8, 1, 3, 7, |9, 10

4, 5, 2, 6, 1, 8, 3, 7, |9, 10

4, 5, 2, 6, 1, 8, 3, 7, |9, 10

4, 5, 2, 6, 1, 3, 8, 7, |9, 10

4, 5, 2, 6, 1, 3, 8, 7, |9, 10

4, 5, 2, 6, 1, 3, 7, 8, |9, 10

4, 5, 2, 6, 1, 3, 7, |8, 9, 10

4th:

4, 5, 2, 6, 1, 3, 7, |8, 9, 10

4, 5, 2, 6, 1, 3, 7, |8, 9, 10

4, 2, 5, 6, 1, 3, 7, |8, 9, 10

4, 2, 5, 6, 1, 3, 7, |8, 9, 10

4, 2, 5, 6, 1, 3, 7, |8, 9, 10

4, 2, 5, 1, 6, 3, 7, |8, 9, 10

4, 2, 5, 1, 6, 3, 7, |8, 9, 10

4, 2, 5, 1, 3, 6, 7, |8, 9, 10

4, 2, 5, 1, 3, 6, 7, |8, 9, 10

4, 2, 5, 1, 3, 6, |7, 8, 9, 10

5th:

4, 2, 5, 1, 3, 6, |7, 8, 9, 10

2, 4, 5, 1, 3, 6, |7, 8, 9, 10

2, 4, 5, 1, 3, 6, |7, 8, 9, 10

2, 4, 5, 1, 3, 6, |7, 8, 9, 10

2, 4, 1, 5, 3, 6, |7, 8, 9, 10

2, 4, 1, 5, 3, 6, |7, 8, 9, 10

2, 4, 1, 3, 5, 6, |7, 8, 9, 10

2, 4, 1, 3, 5, 6, |7, 8, 9, 10

2, 4, 1, 3, 5, |6, 7, 8, 9, 10

6th:

2, 4, 1, 3, 5, |6, 7, 8, 9, 10

2, 4, 1, 3, 5, |6, 7, 8, 9, 10

2, 1, 4, 3, 5, |6, 7, 8, 9, 10

2, 1, 4, 3, 5, |6, 7, 8, 9, 10

2, 1, 3, 4, 5, |6, 7, 8, 9, 10

2, 1, 3, 4, 5, |6, 7, 8, 9, 10

2, 1, 3, 4, |5, 6, 7, 8, 9, 10

7th:

2, 1, 3, 4, |5, 6, 7, 8, 9, 10

1, 2, 3, 4, |5, 6, 7, 8, 9, 10

1, 2, 3, 4, |5, 6, 7, 8, 9, 10

1, 2, 3, 4, |5, 6, 7, 8, 9, 10

1, 2, 3, |4, 5, 6, 7, 8, 9, 10

8th:

1, 2, 3, |4, 5, 6, 7, 8, 9, 10

1, 2, 3, |4, 5, 6, 7, 8, 9, 10

1, 2, | 3, 4, 5, 6, 7, 8, 9, 10

9th:

1, 2, | 3, 4, 5, 6, 7, 8, 9, 10

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

-) Selection Sort:

5, 6, 8, 4, 10, 2, 9, 1, 3, 7

5, 6, 8, 4, 7, 2, 9, 1, 3, |10

5, 6, 8, 4, 7, 2, 3, 1, |9, 10

5, 6, 8, 4, 7, 2, 3, 1, |9, 10

5, 6, 1, 4, 3, 2, |7, 8, 9, 10

5, 2, 1, 4, 3, |6, 7, 8, 9, 10

3, 2, 1, 4, |5, 6, 7, 8, 9, 10

3, 2, 1, |4, 5, 6, 7, 8, 9, 10

1, 2, |3, 4, 5, 6, 7, 8, 9, 10

1, |2, 3, 4, 5, 6, 7, 8, 9, 10

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

1-3) Write the recurrence relation of quick sort algorithm for the worst case, and solve it. Show all the steps clearly.

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

$T(n-1) = T(n-2) + n-1$

$T(n) = T(n-2) + n-1 + n$

$T(n-2) = T(n-3) + n-2$

$T(n) = T(n-3) + n-2 + n-1 + n$

…

$T(n)=T(n-k) + (n-(k-1)) + (n-(k-2)) + \ldots + (n-1) + n$

Assume $n-k =0$, $n=k$

$T(n) = T\ 0) + 1 +2 + 3 \ldots + (n-1) + n$

$T(n) = n(n+1)/2$ ---> $O(n^2)$

Analysis of insertionSort:

Array Size	Time elapsed	countComp	countMove
2000	1ms	1009741	1007742
6000	20ms	9012135	9006136
10000	63ms	24716506	24706507
14000	110ms	48715854	48701855
18000	181ms	80783972	80765973
22000	271ms	120871229	120849230
26000	375ms	167928683	167902684
30000	502ms	224740667	224710668

Analysis of mergeSort:

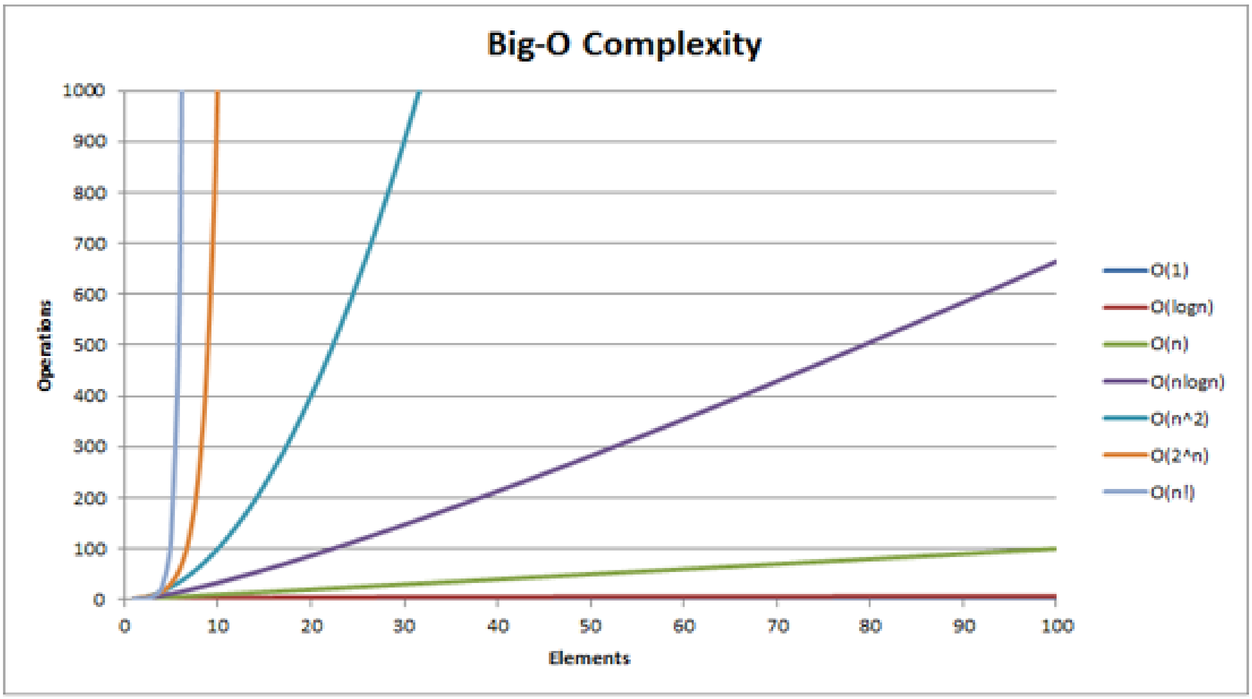
Array Size	Time elapsed	countComp	countMove
2000	4 ms	19443	43904
6000	10 ms	67841	151616
10000	16 ms	120379	267232
14000	23 ms	175491	387232
18000	29 ms	231768	510464
22000	35 ms	289984	638464
26000	42 ms	348936	766464
30000	48 ms	408568	894464

```
Analysis of quickSort:

Array Size        Time elapsed        countComp        countMove
2000              1 ms                58438            92541
6000              1 ms                89233            144576
10000             1 ms                242865           154452
14000             1 ms                239021           350268
18000             2 ms                286022           472947
22000             3 ms                384466           653826
26000             4 ms                438603           721308
30000             5.40731e-317 ms          554447           970872
------------------------------------------------------------------
```

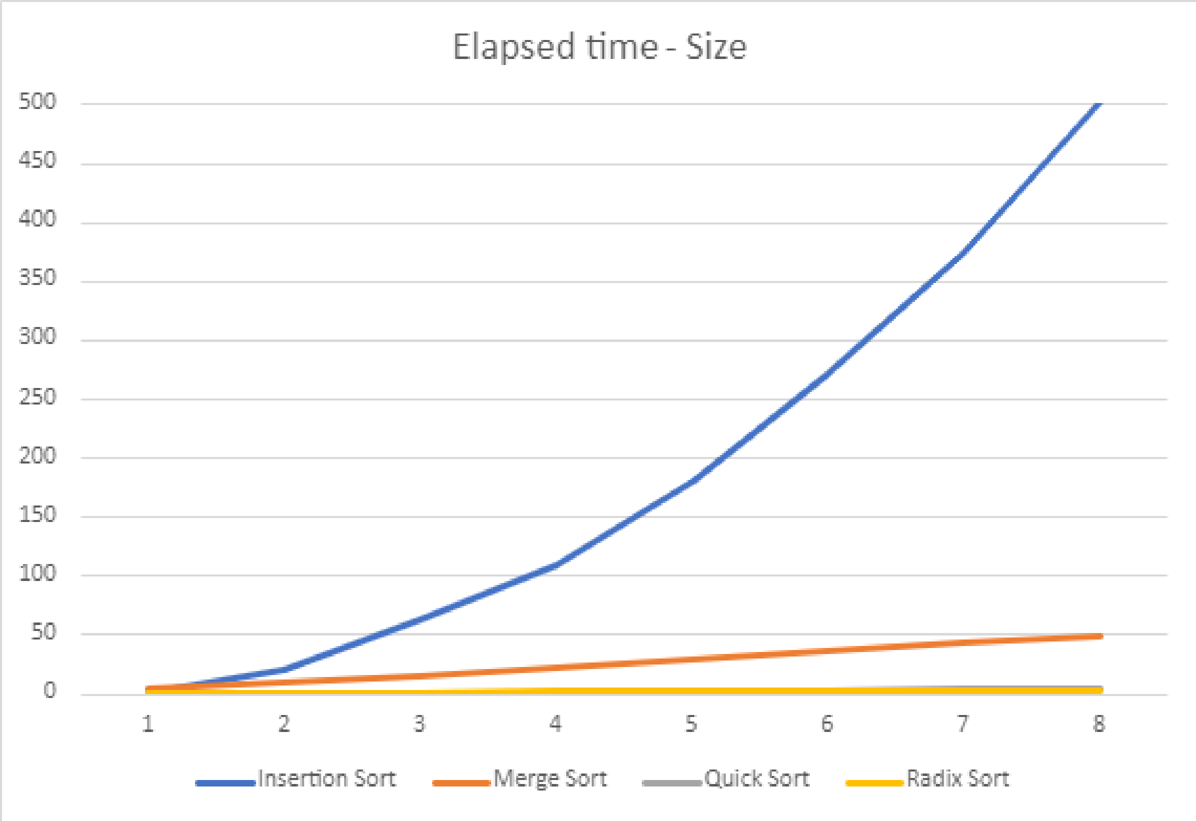
```
Analysis of radixSort:

Array Size        Time elapsed
2000              1 ms
6000              1 ms
10000             1 ms
14000             2 ms
18000             2 ms
22000             2 ms
26000             2 ms
30000             3 ms
```

Question 3

Theoretical complexity:



Experimental complexity:



The theoretical complexity of insertion sort is O(n^2) for the worst-case and average case. The array used in the experiment was a random sorted integer array, so the expected case is the average case that is O(n^2). Experimental results of insertion sort behave like O(n^2) so, theoretical and empirical results are similar for insertion sort. The theoretical complexity of merge sort is O(nlogn) for the worst-case and average case. The expected case is the average case that is O(nlogn). Although the line is not clear to understand O(nlogn) from the graph, it can be understood when compared to the insertion sort's line and when examined alone in another graph. The theoretical complexity of quicksort is O(n^2) for the worst case and O(nlogn) for the average case. The expected case is the average case that is O(nlogn). However, the sizes of the random sorted integer array are not big enough the observe O(nlogn) for quicksort. The theoretical complexity of radix sort is O(n) for the worst case and the average case. However, the sizes of the random sorted integer array are not big enough the observe O(n) for radix sort. If the applied array will be an array of increasing numbers instead of randomly generated numbers, insertion sort behaves like O(n) because it is best case, and the elapsed time will be lesser because all sorting functions sort the array in ascending order so, no time waste for the swap. Merge sort will behave like O(nlogn) because it is the best case for merge sort, and the elapsed time will be lesser because of fewer iterations. Quicksort is slow when the array is already sorted, and choosing the first element as a pivot will increase the elapsed time because the smallest element is selected as a pivot. Radix sort will be the same because it does not use key comparisons, so it does not depend on the array of increasing numbers.