Name: Burak Yetiştiren

ID: 21802608

Section: 01

Question 1

a) We must have $cn^5 \geq 20n^4 + 20n^2 + 5 \geq 0$ for $n \geq n_0$, c, $n_0 > 0$

If we take $c = 20$,

$$20n^5 \geq 20n^4 + 20n^2 + 5$$

If we take $n_0 = 4$,

$$20.4^5 \geq 20.4^4 + 20.4^2 + 5$$

Hence, we have,

$$f(n) = 20n^4 + 20n^2 + 5 = O(n^5)$$

b)

Selection Sort

Trace for "Selection Sort" algorithm, largest element in unsorted sublist is marked with yellow, red mark is the border for sorted and unsorted sublists on the left side of the border we have the unsorted sublist, as for the right side we have the sorted sublist.

Initial array	18	4	47	24	15	24	17	11	31	23
After 1st swap	18	4	24	15	24	17	11	31	23	47
After 2nd swap	18	4	24	15	24	17	11	23	31	47
After 3rd swap	18	4	15	24	17	11	23	24	31	47
After 4th swap	18	4	15	17	11	23	24	24	31	47
After 5th swap	18	4	15	17	11	23	24	24	31	47
After 6th swap	4	15	17	11	18	23	24	24	31	47
After 7th swap	4	15	11	17	18	23	24	24	31	47
After 8th swap	4	11	15	17	18	23	24	24	31	47
After 9th swap	4	11	15	17	18	23	24	24	31	47

Bubble Sort

Trace for "Bubble Sort" algorithm, elements to be compared are marked with yellow, red mark is the border for sorted and unsorted sublists on the left side of the border we have the unsorted sublist, as for the right side we have the sorted sublist.

Pass 1

18	4	47	24	15	24	17	11	31	23
4	18	47	24	15	24	17	11	31	23
4	18	47	24	15	24	17	11	31	23
4	18	24	47	15	24	17	11	31	23
4	18	24	15	47	24	17	11	31	23
4	18	24	15	24	47	17	11	31	23
4	18	24	15	24	17	47	11	31	23
4	18	24	15	24	17	11	47	31	23
4	18	24	15	24	17	11	31	47	23
4	18	24	15	24	17	11	31	23	47

Pass 2

4	18	24	15	24	17	11	31	23	47
4	18	24	15	24	17	11	31	23	47
4	18	24	15	24	17	11	31	23	47
4	18	15	24	24	17	11	31	23	47
4	18	15	24	24	17	11	31	23	47
4	18	15	24	17	24	11	31	23	47
4	18	15	24	17	11	24	31	23	47
4	18	15	24	17	11	24	31	23	47
4	18	15	24	17	11	24	23	31	47

Pass 3

4	18	15	24	17	11	24	23	31	47
4	18	15	24	17	11	24	23	31	47
4	15	18	24	17	11	24	23	31	47
4	15	18	24	17	11	24	23	31	47
4	15	18	17	24	11	24	23	31	47
4	15	18	17	11	24	24	23	31	47
4	15	18	17	11	24	24	23	31	47
4	15	18	17	11	24	23	24	31	47

Pass 4

4	15	18	17	11	24	23	24	31	47
4	15	18	17	11	24	23	24	31	47
4	15	18	17	11	24	23	24	31	47
4	15	17	18	11	24	23	24	31	47
4	15	17	11	18	24	23	24	31	47
4	15	17	11	18	24	23	24	31	47
4	15	17	11	18	23	24	24	31	47

Pass 5

4	15	17	11	18	23	24	24	31	47
4	15	17	11	18	23	24	24	31	47
4	15	17	11	18	23	24	24	31	47
4	15	11	17	18	23	24	24	31	47
4	15	11	17	18	23	24	24	31	47
4	15	11	17	18	23	24	24	31	47

Pass 6

4	15	11	17	18	23	24	24	31	47
4	15	11	17	18	23	24	24	31	47
4	11	15	17	18	23	24	24	31	47
4	11	15	17	18	23	24	24	31	47
4	11	15	17	18	23	24	24	31	47

Pass 7

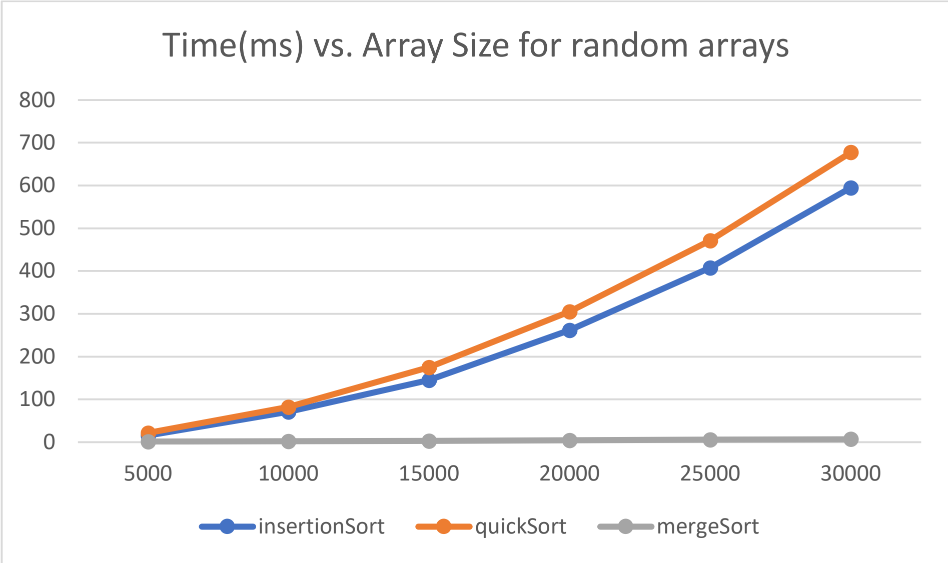
4	11	15	17	18	23	24	24	31	47
4	11	15	17	18	23	24	24	31	47
4	11	15	17	18	23	24	24	31	47
4	11	15	17	18	23	24	24	31	47

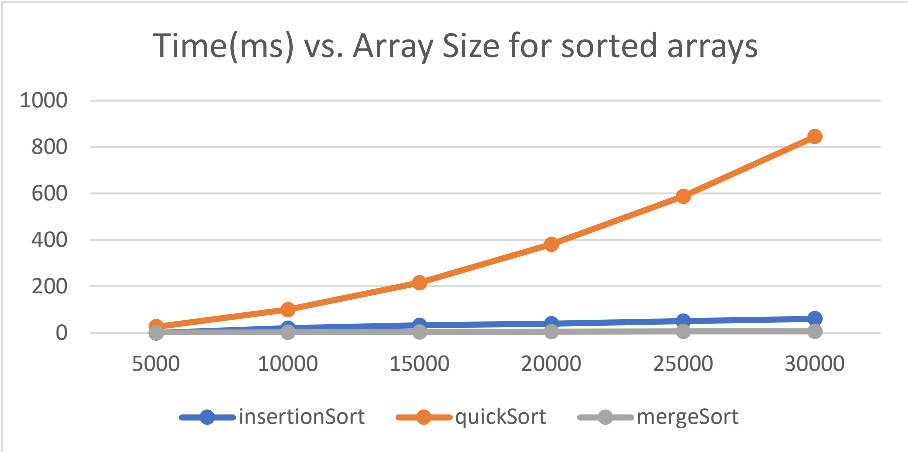
Pass 8

4	11	15	17	18	23	24	24	31	47
4	11	15	17	18	23	24	24	31	47
4	11	15	17	18	23	24	24	31	47

Pass 9

4	11	15	17	18	23	24	24	31	47
4	11	15	17	18	23	24	24	31	47





The theoretical time complexity for insertionsort algorithm is O(n) in best case where the array is already sorted and $O(n^2)$ in worst case and average case. When we look at our results in the first graph, we can observe that the structure of the line for insertionSort looks like the n^2 graph, and we can interpret from this result that the case with the data in our experiment was not for the best case, but it was an average case. As for the second graph we know that the array is already sorted and we expect the time complexity to be O(n). When we look at the graph this situation is proved, also when contrasted with the first graph, we can see that the time complexity is reduced too.

The theoretical time complexity for quicksort algorithm is O(n*logn) in best case and average case, $O(n^2)$ in worst case, according to the selection of the pivot. In our results, both graphs look like they have the structure of the line n^2, we know that the array in the second graph was already sorted and the result we have gotten is indeed accurate, as for the results in the first graph, we can say that the pivot was close to the beginning, so that our graph looks like the graph for n^2. But we know that the pivot was not the first item, as the second graph has a bigger time complexity when compared with the first graph.

The theoretical time complexity for mergesort algorithm is O(n*logn) in worst case and average case. Indeed, when we look at the both graphs, it can be seen that they share the same growth rate close to O(n*logn).

For nearly sorted arrays theoretically, most efficient algorithms are lined as:

1. Insertionsort
2. Mergesort
3. Quicksort

As insertionsort algorithm works in O(n) time complexity, whereas the quicksort algorithm works in $O(n^2)$ time complexity and the mergesort works in O(n*logn) time complexity with a nearly sorted array as an input.

I have conducted two experiments, in each there were 6 different array sizes: 5000, 10000, 15000, 20000, 25000, 30000. For the first experiment I have chosen K as the 0.1% of the array size and for the second I have chosen K as 1% of the array size, not to make the array far from being sorted.

```
Array Size      Elapsed Time
     5000         0.0400 ms
    10000        16.0000 ms
    15000        16.0000 ms
    20000        15.0000 ms
    25000        47.0000 ms
    30000        49.0000 ms
--------------------------------
Part c - Performance Analysis fo
Array Size      Elapsed Time
     5000        23.9000 ms
    10000        84.8000 ms
    15000       180.3000 ms
    20000       313.6000 ms
    25000       486.7000 ms
    30000       688.1000 ms
Part c - ----------------------
Performance Analysis for mergeSo
Array Size      Elapsed Time
     5000         0.0000 ms
    10000         3.1000 ms
    15000         3.1000 ms
    20000         4.7000 ms
    25000         4.7000 ms
    30000         7.1000 ms
```

Elapsed time table for K = 0.1% of the array size

(First part for insertionsort, second part for quicksort, third part for mergesort)

```
Array Size      Elapsed Time
     5000         0.0250 ms
    10000        18.0000 ms
    15000        31.0000 ms
    20000        47.0000 ms
    25000        53.0000 ms
    30000        78.0000 ms
-------------------------------
Part c - Performance Analysis f
Array Size      Elapsed Time
     5000        22.7000 ms
    10000        81.5000 ms
    15000       175.7000 ms
    20000       308.0000 ms
    25000       471.0000 ms
    30000       677.6000 ms
Part c - ---------------------
Performance Analysis for mergeS
Array Size      Elapsed Time
     5000         0.0000 ms
    10000         1.6000 ms
    15000         3.1000 ms
    20000         4.6000 ms
    25000         5.6000 ms
    30000         7.8000 ms
```

Elapsed time table for K = 1% of the array size

(First part for insertionsort, second part for quicksort, third part for mergesort)

From these results, the efficiency of the algorithms to sort nearly sorted arrays line as:

1. Mergesort
2. Insertionsort
3. Quicksort

The results prove that the quicksort algorithm works in the $O(n^2)$ time complexity and the inseritonsort algorithm works in the O(n) time complexity, but fail to prove the case for mergesort algorithm.