CS202-1

Homework 1

Turan Mert Duran - 21601418

Question 1)

a)

We need to find two positive constants: **c** and **n₀** such that:

$0 \leq 20n^4 + 20n^2 + 5 \leq cn^5$ for all $n \geq n_0$

Choose c = 45 and $n_0 = 1$ => $0 \leq 20n^4 + 20n^2 + 5 \leq 45n^5$ for all $n \geq 1$

b)

Selection Sort:

Beginning- [18, 4, 47, 24, 15, 24, 17, 11, 31, 23]

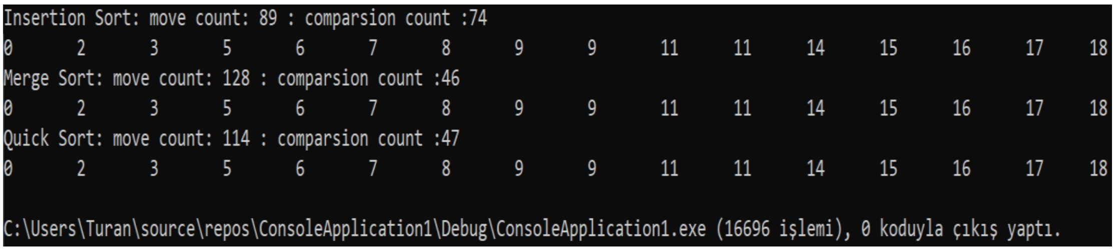
1- [4, 18, 47, 24, 15, 24, 17, 11, 31, 23]
2- [4, 11, 47, 24, 15, 24, 17, 18, 31, 23]
3- [4, 11, 15, 24, 47, 24, 17, 18, 31, 23]
4- [4, 11, 15, 17, 47, 24, 24, 18, 31, 23]
5- [4, 11, 15, 17, 18, 24, 24, 47, 31, 23]
6- [4, 11, 15, 17, 18, 23, 24, 47, 31, 24]
7- [4, 11, 15, 17, 18, 23, 24, 47, 31, 24]
8- [4, 11, 15, 17, 18, 23, 24, 24, 31, 47]

Bubble Sort:

Beginning - [18, 4, 47, 24, 15, 24, 17, 11, 31, 23]

1- [4, 18, 24, 15, 24, 17, 11, 31, 23, 47]
2- [4, 18, 15, 24, 17, 11, 24, 23, 31, 47]
3- [4, 15, 18, 17, 11, 24, 23, 24, 31, 47]
4- [4, 15, 17, 11, 18, 23, 24, 24, 31, 47]
5- [4, 15, 11, 17, 18, 23, 24, 24, 31, 47]
6- [4, 11, 15, 17, 18, 23, 24, 24, 31, 47]

Screen Shot of Makefile Question 2



```
Insertion Sort: move count: 89 : comparsion count :74
0       2       3       5       6       7       8       9       9       11      11      14      15      16      17      18
Merge Sort: move count: 128 : comparsion count :46
0       2       3       5       6       7       8       9       9       11      11      14      15      16      17      18
Quick Sort: move count: 114 : comparsion count :47
0       2       3       5       6       7       8       9       9       11      11      14      15      16      17      18

C:\Users\Turan\source\repos\ConsoleApplication1\Debug\ConsoleApplication1.exe (16696 işlemi), 0 koduyla çıkış yaptı.
```

Question 2)

RANDOMLY CREATED ARRAY

Part c- Time analysis of Insertion Sort

Array Size	Time Elapsed	compCount	moveCount
5000	41 ms	6271235	6281233
10000	58 ms	25121944	25141942
15000	141 ms	56459706	56489704
20000	248 ms	99903281	99943279
25000	370 ms	155497378	155547376
30000	526 ms	224666212	224726210

RANDOMLY CREATED ARRAY

Part c- Time analysis of Merge Sort

Array Size	Time Elapsed	compCount	moveCount
5000	7 ms	55201	123616
10000	8 ms	120415	267232
15000	26 ms	189231	417232
20000	43 ms	260915	574464
25000	93 ms	334016	734464
30000	82 ms	408386	894464

RANDOMLY CREATED ARRAY

Part c- Time analysis of Quick Sort

Array Size	Time Elapsed	compCount	moveCount
5000	2 ms	32634	111278
10000	5 ms	73101	246063
15000	10 ms	127033	421163
20000	12 ms	162441	541211
25000	16 ms	209483	695853
30000	20 ms	259829	860695

ALREADY SORTED ARRAY

Part c- Time analysis of Insertion Sort

Array Size	Time Elapsed	compCount	moveCount
5000	0.018163 ms	571817533	575733032
10000	0.036778 ms	571849537	575896642
15000	0.053578 ms	571918545	576233868
20000	0.135155 ms	572024909	576751094
25000	0.089183 ms	572172925	577455552
30000	0.109592 ms	572361401	578350010

ALREADY SORTED ARRAY

Part c- Time analysis of Merge Sort

Array Size	Time Elapsed	compCount	moveCount
5000	1 ms	571849537	575876644
10000	7 ms	572024909	576711096
15000	7 ms	572172925	577405554
20000	8 ms	572172925	577405554
25000	10 ms	572361401	578290012
30000	12 ms	572589129	579364470

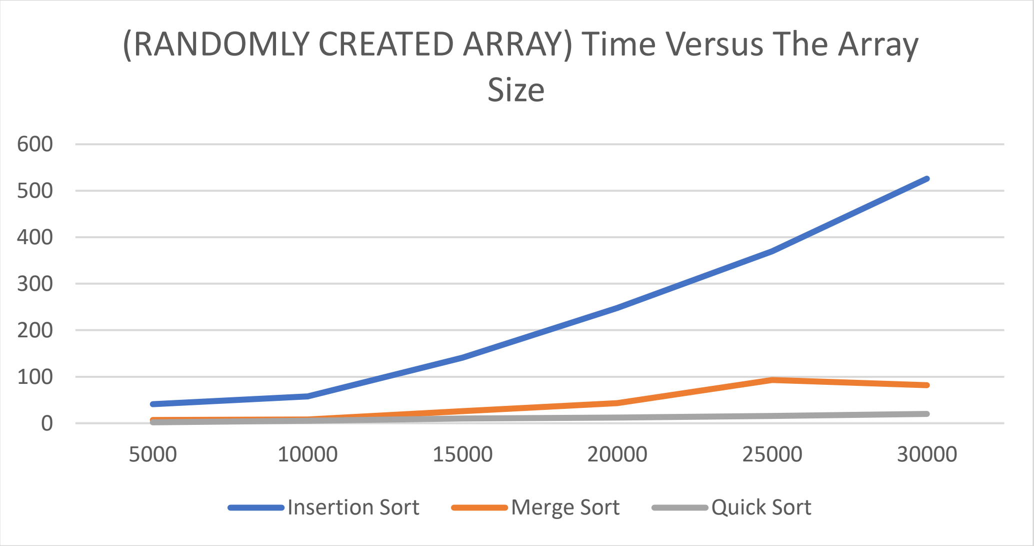
ALREADY SORTED ARRAY

Part c- Time analysis of Quick Sort

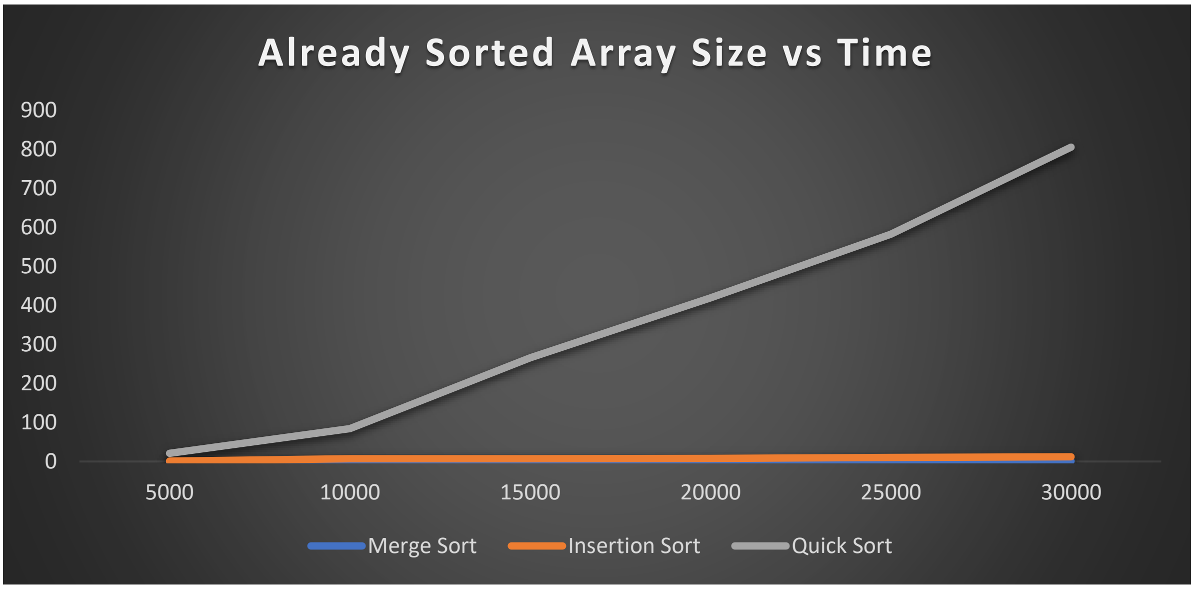
Array Size	Time Elapsed	compCount	moveCount
5000	21 ms	571817533	575753028
10000	84 ms	571849537	575936638
15000	266 ms	571918545	576293864
20000	419 ms	572024909	576831090
25000	583 ms	572172925	577555548
30000	806 ms	572361401	578470006

As we can observe from the results that quickest sorting algorithm was the quick sort algorithm among the insertion and merge sort in our empirical results for randomly created arrays. As we expected, insertion sort was the slowest sorting algorithm among these in randomly created arrays. It can be seen that, as array size increase in randomly created arrays, execution time of insertion sort is increasing because it is directly proportional to the array size. When the array size reach the 30000 in randomly created arrays sorting time of insertion sort is approximately 5 times slower than the other two algorithms. However, in already sorted arrays quick sort is the worst sorting algorithm when we comparet it with others. These results show us, we should choose our sorting algorithms as which is most appropriate for our arrays. This result shows us that sorting algorithms are important for optimization and effeciency of our program. Theoretically, i expected not much difference between quick and merge sort in randomly created arrays because of their time complexity is same in average. However, there were a little bit difference in sorting times between merge and quick sort. We obtained that quick sort was faster than merge sort in randomly created arrays. In my opinion, because we created arrays randomly, these arrays were more convenient for quick sort algorithm that is why quick sorting algorithm worked a little bit faster than the merge sort.

As a result, all algorithms have different most appropriate usages. Which sorting algorithm would be best for sorting array depends on the size of array and situations of elements in the array. It can’t be said that one of them is most powerful than other in all situations.



(Y axis is execution time in ms, X axis is array size)



(Y axis is execution time in ms, X axis is array size)

Question 3)

In nearly sorted arrays, the most effective sorting algorithm depends on items distance from their target location. In insertion sort, if items were close to their target location sorting takes much more smaller time when we compare with other sorting algorithms. For example, when items were approximately 5 index away from their target location in a 10000 size array, insertion sort takes only 0.39 ms in our experiments whereas quick sort takes 30 ms and merge sort takes 1.6 ms. However, when we look at another experiment that items are approximately 3000 index away from their target location in 10000 size array, insertion sort takes much more time when we compare with others. In this experiment insertion sort took 64 ms to arrange items whereas others took much more less time (Quick sort = 1.6 ms, Merge sort = 1.7 ms). It is observed that in average case merge sort would be much more efficient than the insertion sort and quick sort regardless of items average distance from their target location. Although, insertion sort can be used for nearly sorted arrays much more faster, in the unsorted arrays this algorithm is too slow when we compared with others. We can obtain from the results that, if average distance of items from their target location is small insertion sort would be most efficient, if average distance of items from their target location is middle merge sort is most efficient and if average distance of items from their target location is large quick sort is the most efficient sort so, choosing most suitable sort algorithm for array would be change because of items stiuation of array.