Alara Zeybek
22102544
CS201-002

CS201 HOMEWORK 2

Time Complexity and Algorithm Analysis

In the second homework of CS201 course, we study time complexity of 3 algorithms. Each algorithm permits to find the median of an array. I have done these observations in a computer with Intel(R) Core (TM) i7-10870H CPU @ 2.20GHz, 16.0 GB (15.8 GB usable), 64-bit operating system, x64-based processor, Windows 11.

General Information About 3 Algorithms:

First algorithm sorts the input array with selection sort (O(n^2)) and returns the median (O(1)). Second algorithm is using QuickSort to sort the array (O(nlogn)). Third algorithm is Quickselect algorithm and has O(nlogn) complexity in average, O(n) complexity in worst case.

Execution:

Number of elements in array	Time to calculate Algorithm 1 (milliseconds)	Time to calculate Algorithm 2 (milliseconds)	Time to calculate Algorithm 3 (milliseconds)
10	0.009	0.007	0.008
100	0.016	0.012	0.011
1000	1.079	0.850	0.123
5000	19.056	2.188	0.336
10000	76.863	2.800	1.002
50000	1696.789	8.730	3.156
100000	7102.300	15.219	6.895
500000	179484.200	89.649	28.056

As we see from the table, in every case, algorithm 1 is the slowest, algorithm 3 is the fastest and algorithm 2 is almost equal to algorithm 3.

Algorithm 1:

First algorithm uses one while loop and one for loop, thus its complexity is O(n^2): It is basically sorts the array with selection sort and returns the middle element. First while loop is executed until the int variable “count” reaches at the middle index of the array. In this part, the element that limits the execution time is array’s length/2. Since it is a determined number such as n, the time augmented linearly with the augmentation of the array’s size. Thus, the complexity of the first part of this algorithm is O(n). Inside the first part, there is a for loop to sort the array in descending numbers. Since the boundary of this loop is the size of the array, it has a linear relation. For this reason, second part of the first algorithm also has a complexity of O(n). At the end, function returns the middle element of a sorted array which is the median. Hence, second part of the algorithm is inside the first part, the total complexity of the first algorithm is O(n^2).

Alara Zeybek
22102544
CS201-002

In Figure A, you can see the experimental execution time versus array size graph for algorithm 1 and in Figure B, you can see the comparison of the previous graph (Figure A) and the theorical O(n^2) graph. (The unit of the execution time is milliseconds.)

Figure A:

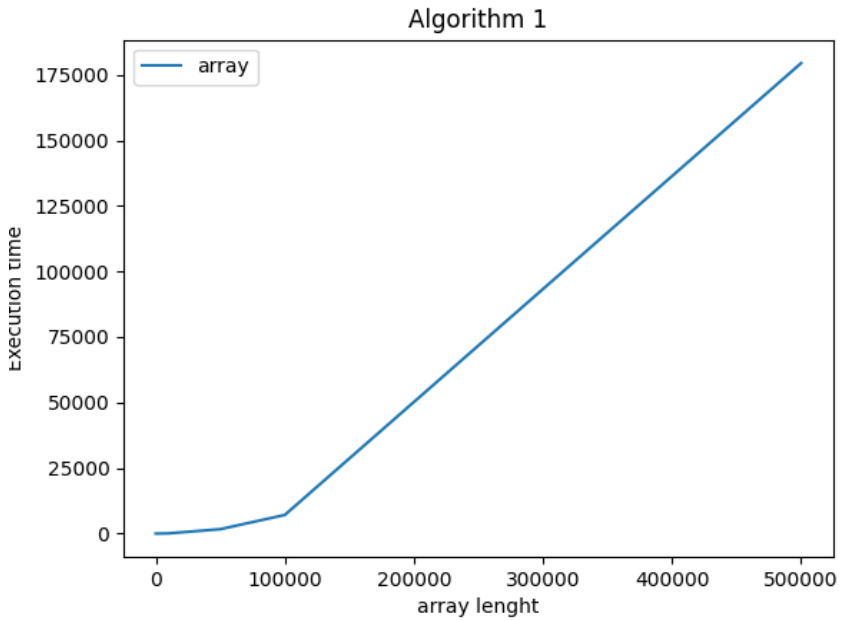
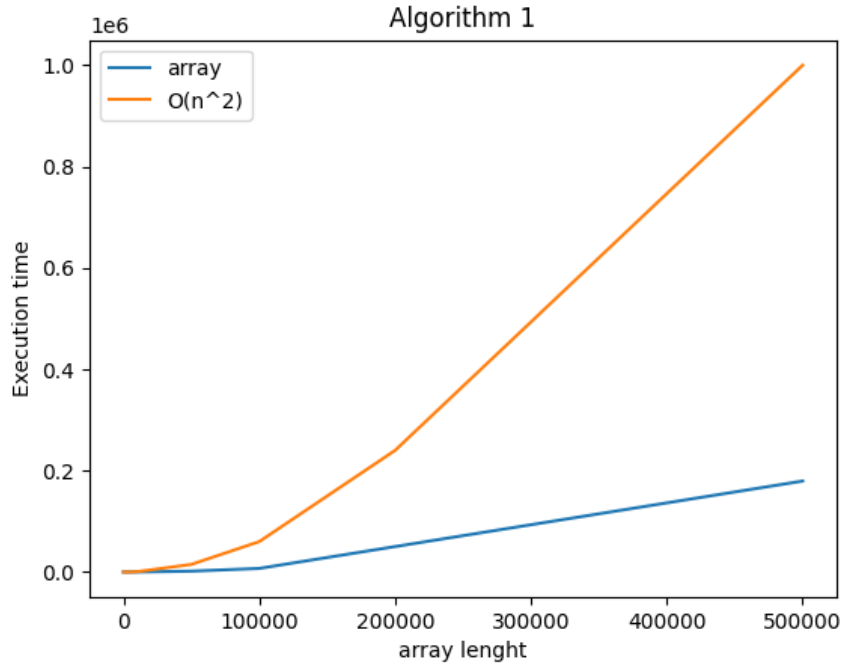


Figure B:



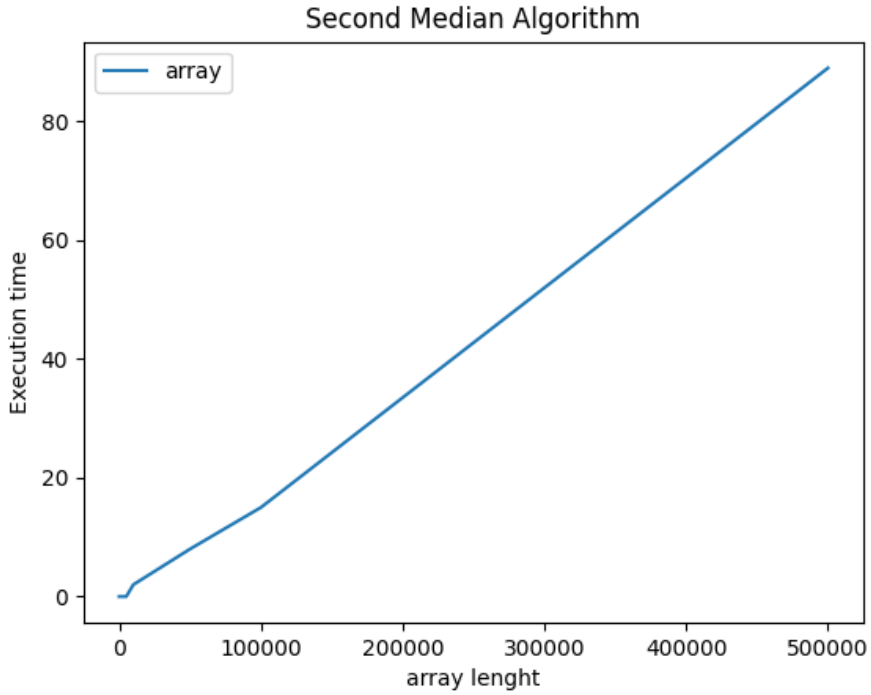
Alara Zeybek
22102544
CS201-002

Algorithm 2:

Second algorithm does one quicksort and then, it picks the middle element. Thus, its complexity is O(nlogn). The function of FIND_MEDIAN_2, calls randomizedQuickSort(…) and returns input[…]. Return part has O(1) complexity, so the total complexity of second algorithm depends on the complexity of randomizedQuickSort function.

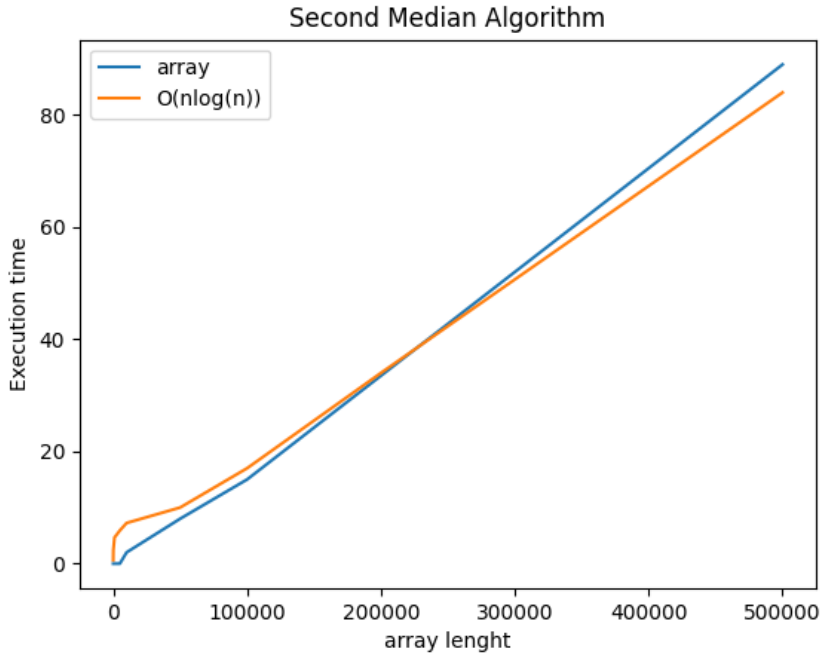
In Figure C, you can see the experimental execution time versus array size graph for algorithm 2 and in Figure D, you can see the comparison of the previous graph (Figure C) and the theorical O(nlogn) graph. (The unit of the execution time is milliseconds.)

Figure C:



Alara Zeybek
22102544
CS201-002

Figure D:



Algorithm 3:

Third algorithm is an algorithm called Quickselect. Its average complexity is O(nlogn) and its worst case's complexity is O(n). In this algorithm, the worst case is when the size of the array is bigger than 5 and it is sorted in ascending order because is select(…) function which FIND_MEDIAN_3 returns, does the sorting according to descending order. For this reason, using algorithm 3 for ascending ordered array takes more time than using this algorithm for random created array.

In Figure E, you can see the experimental execution time versus array size graph for algorithm 2 and in Figure F, you can see the comparison of the previous graph (Figure E) and the theorical O(nlogn) graph. (The unit of the execution time is milliseconds.)

Alara Zeybek
22102544
CS201-002

Figure E:

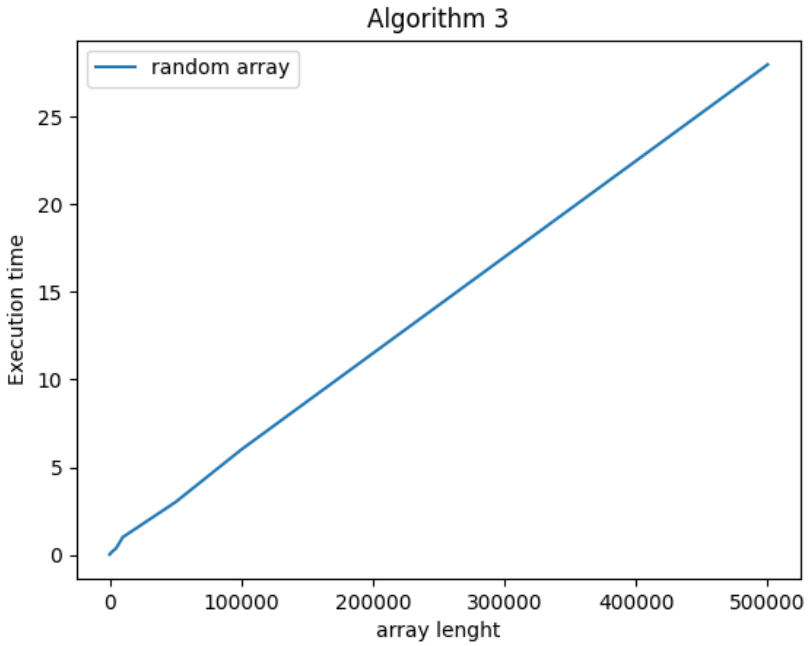
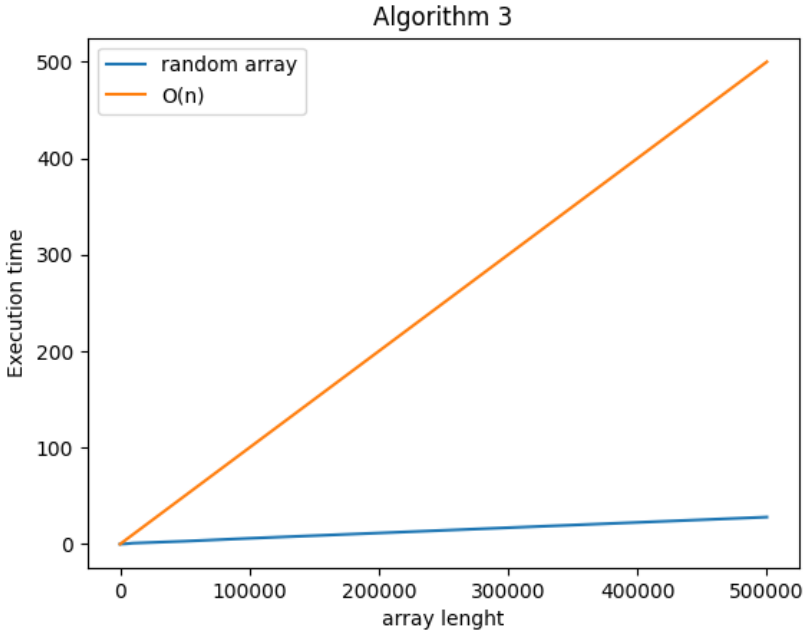


Figure F:



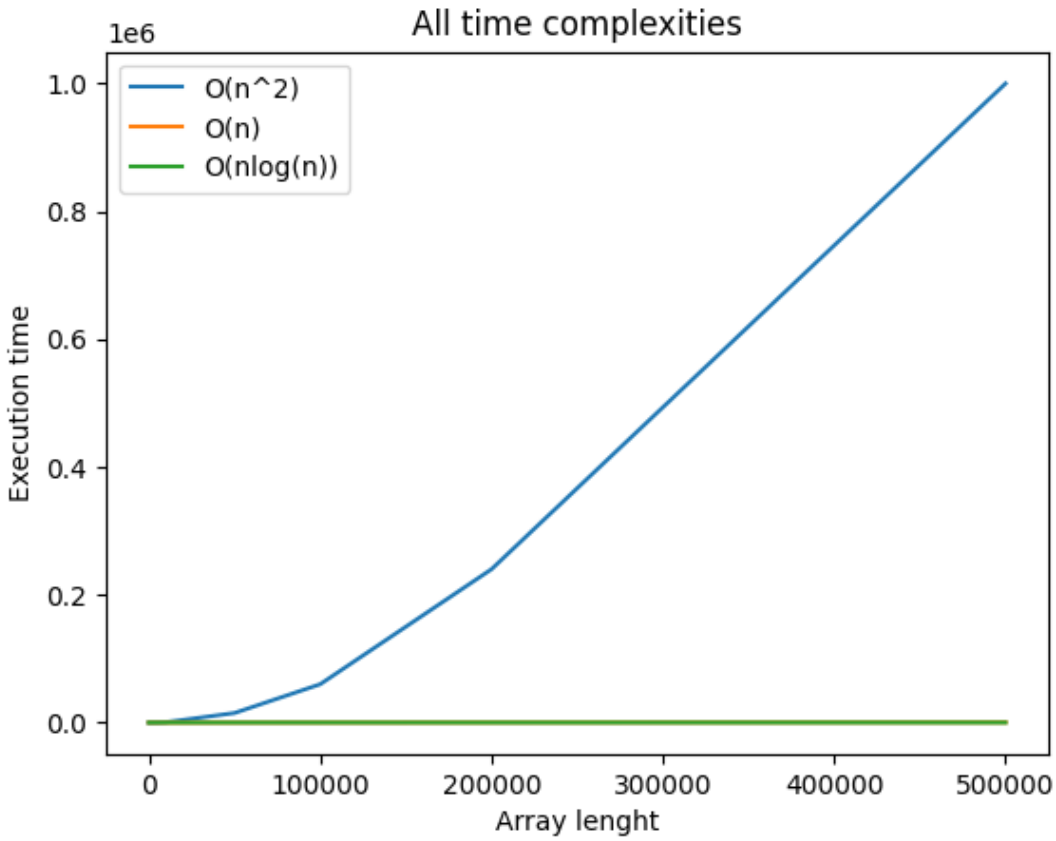
Alara Zeybek
22102544
CS201-002

Conclusion:

Taking into consideration of these algorithms' complexities, I expected that algorithm 1 would be slower than algorithm 2 (O(n^2) > O(nlogn)) and algorithm 3 (O(n^2) > O(n)). If I consider the average case of algorithm 3, I expect that it would take equal time as algorithm 2(O(nlogn) = O(nlogn)). But, if I consider algorithm 3's worst case, I expect that it would be slower than algorithm 2(O(nlogn) > O(n)). I was right about my expectations: In the Figure G, it can be seen the function behaviors of the complexities and in the Figure H, it can be seen how the algorithm behaves according to the time it takes. As I expected before, O(n^2) is slower than O(nlogn) and it is slower than O(n). Furthermore, algorithm 1 is slower than algorithm2 and it is slower than algorithm 3. Since the execution time is really small (milliseconds), orange and green lines look like collide.

To conclude, if we are struggling with large datasets, it would be better to use algorithm 3 or algorithm 2. On the other hand, if we are dealing with slightly smaller datasets (in my execution, the obvious difference occurs at array length 5000, so if we want to find the median of an array that has a length less than 5000), we can also consider algorithm 1. There wouldn't be too much difference but it would be still less efficient than other two algorithms.

Figure G:



Alara Zeybek
22102544
CS201-002

Figure H: