Student Name: Gün Taştan

Section: 3

Student ID: 22101850

CS-201 HOMEWORK 2 REPORT

1. Description of Algorithms

Algorithm 1

In the first algorithm, we loop over the array and find the maximum value in it. To find the maximum value in the array, we visit each element of this n-element array once. Visiting each element once takes O(n) time (Also known as linear running time and take proportionally grows as the input grows). Then, we eliminate this value by putting it at the beginning of the array. Then, since the goal is to find the median of the array, we need to find the largest n/2th element. For this, if we repeat the above-mentioned maximum finding process n/2 times, we assign the largest n/2th element to the beginning of the array, which shows that the last element to be subtracted is the median. This part of the algorithm takes O(n*n/2) time since the maximum search process takes O(n) time and this process is repeated n/2 times. At the end, the algorithm takes a total of $O((n^2/2)+1)$ time since the loop is entered once again for the median. When the coefficients and lesser priority terms are removed, $O((n^2)/2 + 1)$ time equals $O(n^2)$ time. So the first algorithm takes $O(n^2)$ time.

Algorithm 2

In the second algorithm, we sort the array in descending order using quick sort, and then we reach the middle element of the array (which is sorted) to reach the median, we select it. The Quicksort algorithm takes O(n*logn) time and it takes O(1) time (constant time) to reach a specific index of the sorted array. This means that the second algorithm takes the same time as the quicksort. As we learned, Quick Sort algorithm uses divide and conquer technique/mechanics. The Quick Sort algorithm starts by choosing a pivot in the array. Then we go through/traverse the array and place the smaller ones (than pivot) to the left of the pivot and the larger ones to the right of the pivot. Thus, all numbers smaller than the pivot/element we selected are on the left, and large numbers are on the right. Repeating this process for every partition recursively, a sorted array will be obtained. The process of choosing a pivot and placing the larger and smaller elements from the pivot to the right and left of the pivot is called partition, and this partitioning process takes O(n) time. The reason why it takes O(n) time is that in a for loop, it compares the elements of the array according to the pivot and swaps it when needed. So partition operation takes O(n) time. The best case of Quick Sort occurs if pivot is selected as mean element. If the pivot is selected as the mean element, the height of the recursion tree (how many times it is called) will be logn and it will travel over all elements at the level. Since it takes O(n) time to go through all the elements and this operation is called logn times, the time taken will be O(n*logn) in the best case. The reason for this is that if the pivot is selected as the mean element, the height of the recursion tree will be least since the input array will be divided into equal-sized branches. The reason for this is that if the pivot mean is selected as the element, the height of the recursion tree will be minimal since the input array will be divided into equal-sized branches. The case where the quick sort is the slowest will be when the pivot is selected as the largest or smallest element in the array. If we consider this situation in the recursion tree, the length of this tree will be n because the smallest or the largest element is chosen as the pivot in the partition operation, so the array will end up decreasing in size one by one. This means that quick sort takes O(n*n) = O(n^2) time in worst case. Lastly, hen examined and all cases are taken into account, Quick sort average takes O(n*logn) time.

Algorithm 3

The median algorithm of medians is an algorithm developed to provide a good pivot to the quick select algorithm used to find the kth smallest element of an array. In this algorithm, the array taken as input is divided into small groups of five. In other words, the elements in the input array are divided into small subgroups of five. Then, the medians of these five subgroups are found by comparing each other 6 times within themselves. That is, when finding the medians of these small subgroups, 6 comparisons are made for each subgroup. Considering that the input array is n in size and the array is divided into groups of five, this means a total of 6n/5 comparisons. Then, the median of each group is put into a newly created array and the median of this array consisting of medians is found using the same algorithm (recursively). Then, after the median of this array is found, the median of these medians is ready to be used as a pivot for the input array. The rest of the algorithm is partitioned using the medians of the medians. We can calculate the time taken by the algorithm as follows: First of all, we have to go through all the elements of the array while making subgroups of five from the input array. This process will take O(n) time. Then we found the median of each subgroup in O(1) time (constant time). Then it will take O(n/5) time to find the median of these medians. The reason for this is to apply the quick select algorithm on the median of the medians, and this operation can be represented by the expression T(n/5). Finally, if we add the probability that the median with the worst case time is in the partition of size 7n/10 (I prefer not to mention the mathematical calculations because explanation would be so long and complicated. Calculations can be seen from the link provided) , we can see that the algorithm works ,by removing the coefficients and lower order expressions, taking O(n) time.

<u>Note</u>

Quick select is a selection algorithm used to find the kth smallest element in an unordered array. Working with a good average time, this algorithm (O(n)) is similar to quicksort in that it uses partition logic. The main difference from Quicksort is that instead of recurring for both sides of the array after the pivot, it recurs for the side containing the kth smallest element.

2. Table(s)

Table 1. Table of runtimes of different algorithms on finding medians of an arrays

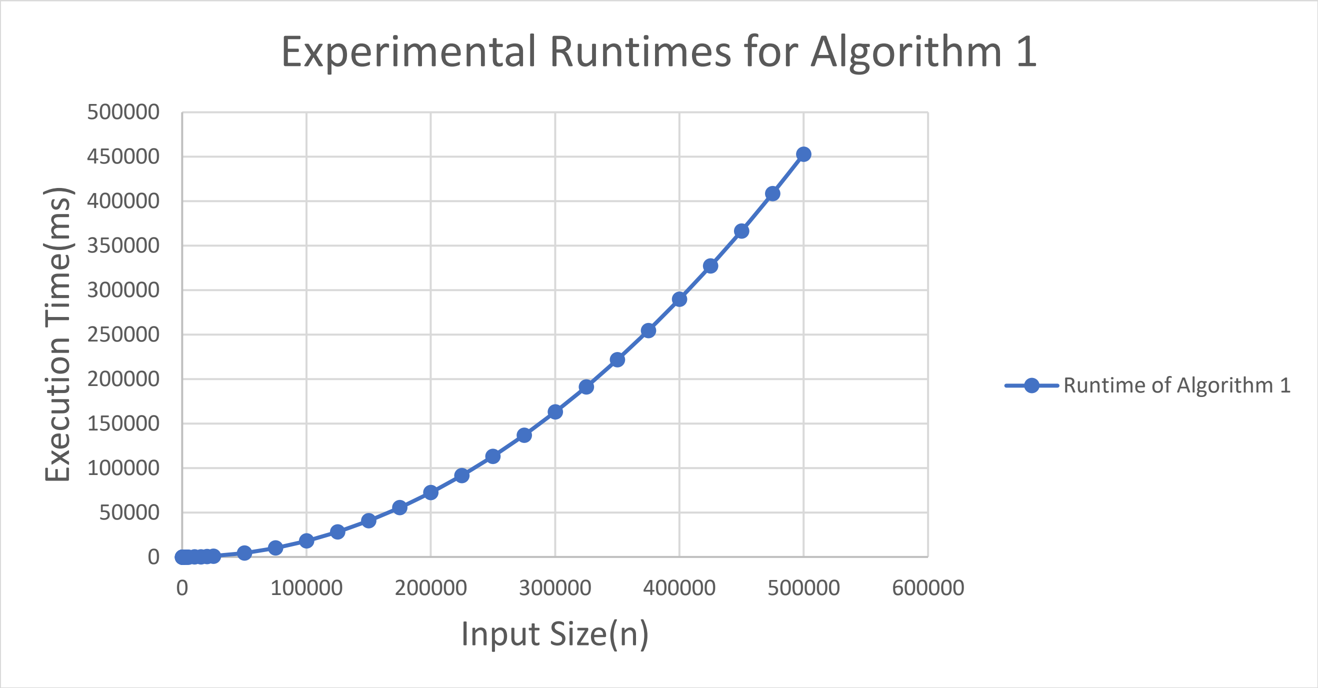
Runtimes of Different Algorithms on Finding Medians of an Arrays			
Array Size	Algorithm 1(ms)	Algorithm 2(ms)	Algorithm 3(ms)
0	0.000273	0.000157	0.000204
100	0.023339	0.015248	0.025885
200	0.084412	0.032956	0.035681
300	0.178966	0.057132	0.053824
400	0.312573	0.07211	0.06115
500	0.485397	0.091532	0.078849
600	0.684261	0.129676	0.083481
700	0.929416	0.136584	0.111184
800	1.20588	0.159147	0.126071
900	1.51634	0.191735	0.146245
1000	1.89244	0.21985	0.154556
2000	7.30428	0.457896	0.298752
3000	16.3057	0.678487	0.474531
4000	28.898	0.961516	0.635223
5000	45.0483	1.23507	0.688789
10000	180.702	2.6828	1.34811
15000	408.582	4.08097	2.05904
20000	726.72	5.57261	2.93352
25000	1134.79	7.41893	3.43457
50000	4533.41	15.2761	6.84293
75000	10197.6	23.7609	10.3755
100000	18121.7	32.0927	13.5676
125000	28315.2	41.1554	18.4751
150000	40823.4	50.1547	21.3371
175000	55532.1	58.1356	23.9459
200000	72525.9	67.9981	26.9747
225000	91780.6	76.2664	30.5002
250000	113277	86.5231	33.6351
275000	137023	95.4742	37.6951
300000	163105	109.77	41.2227
325000	191224	114.352	44.2904
350000	221882	124.099	47.5078
375000	254755	133.465	51.2587
400000	289869	142.394	55.0528
425000	327176	153.688	57.8363
450000	366697	162.527	62.0803
475000	408821	171.178	65.319
500000	452936	180.406	68.5308

Table 2. Table of expected runtimes of different algorithms on finding medians of an arrays

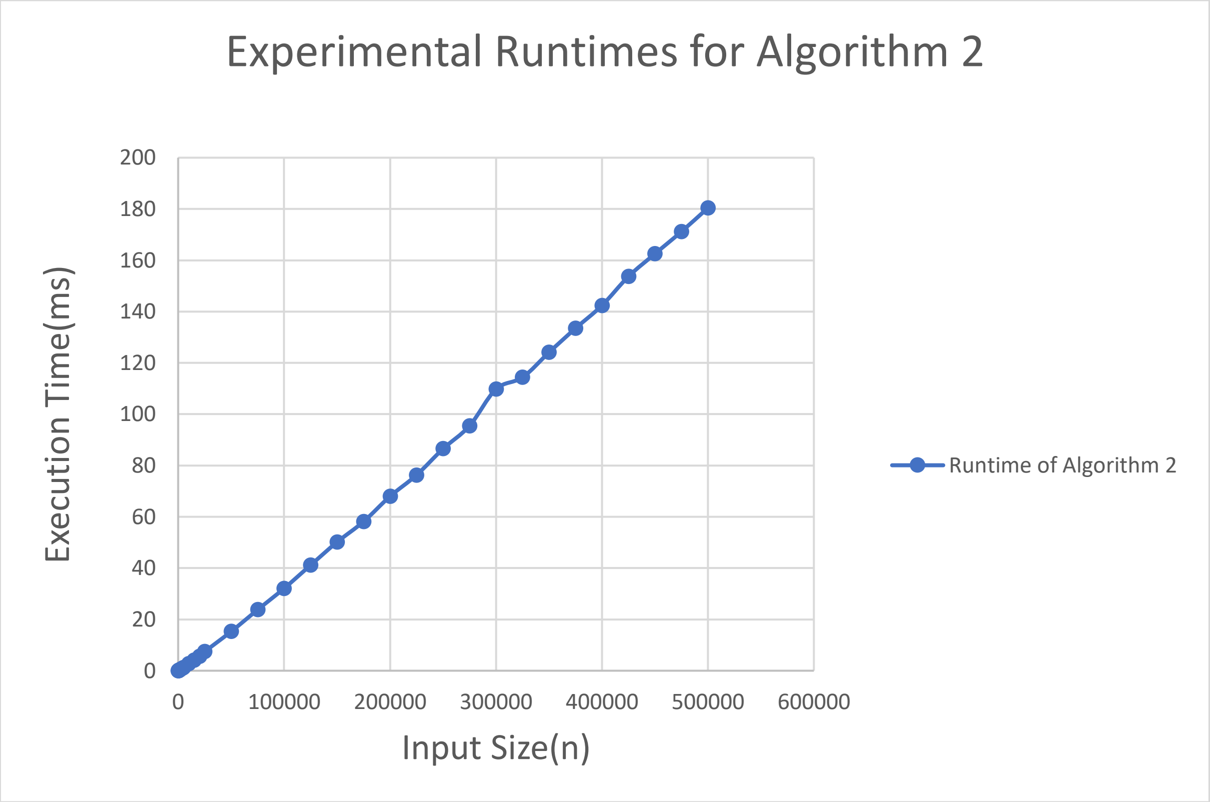
Expected Runtimes Times of Different Algorithms on Finding Medians of an Arrays			
Array Size	Algorithm 1(ms)	Algorithm 2(ms)	Algorithm 3(ms)
0	0.000273	0.000157	0.000204
100	0.023339	0.015248	0.025885
200	0.093356	0.035086105	0.05177
300	0.210051	0.056656717	0.077655
400	0.373424	0.079352421	0.10354
500	0.583475	0.102884737	0.129425
600	0.840204	0.127083751	0.15531
700	1.143611	0.151837192	0.181195
800	1.493696	0.177065264	0.20708
900	1.890459	0.202708304	0.232965
1000	2.3339	0.22872	0.25885
2000	9.3356	0.503341054	0.5177
3000	21.0051	0.795287173	0.77655
4000	37.3424	1.098484215	1.0354
5000	58.3475	1.410047366	1.29425
10000	233.39	3.0496	2.5885
15000	525.1275	4.775777964	3.88275
20000	933.56	6.558210537	5.177
25000	1458.6875	8.382473657	6.47125
50000	5834.75	17.91247366	12.9425
75000	13128.1875	27.8756003	19.41375
100000	23339	38.12	25.885
125000	36467.1875	48.57355242	32.35625
150000	52512.75	59.19377964	38.8275
175000	71475.6875	69.95261365	45.29875
200000	93356	80.83010537	51.77
225000	118153.69	91.81133892	58.24125
250000	145868.75	102.8847366	64.7125
275000	176501.19	114.0410493	71.18375
300000	210051.00	125.2727173	77.655
325000	246518.19	136.5734459	84.12625
350000	285902.75	147.9379117	90.5975
375000	328204.69	159.3615539	97.06875
400000	373424.00	170.8404215	103.54
425000	421560.69	182.3710581	110.01125
450000	472614.75	193.9504149	116.4825
475000	526586.19	205.5757824	122.95375
500000	583475.00	217.2447366	129.425

3. Plot(s)/Graphs

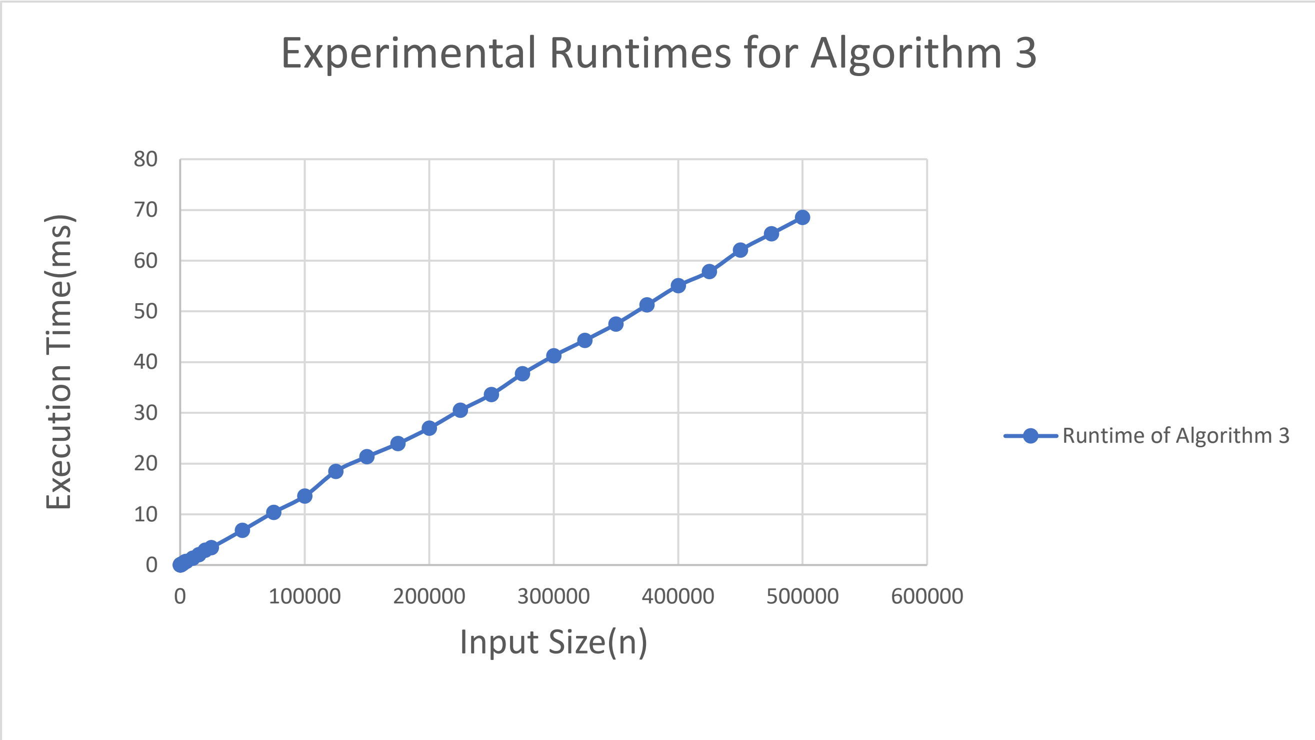
Graph 1. Graph of experimental runtimes of Algorithm 1 plotted by the data gathered from Table 1



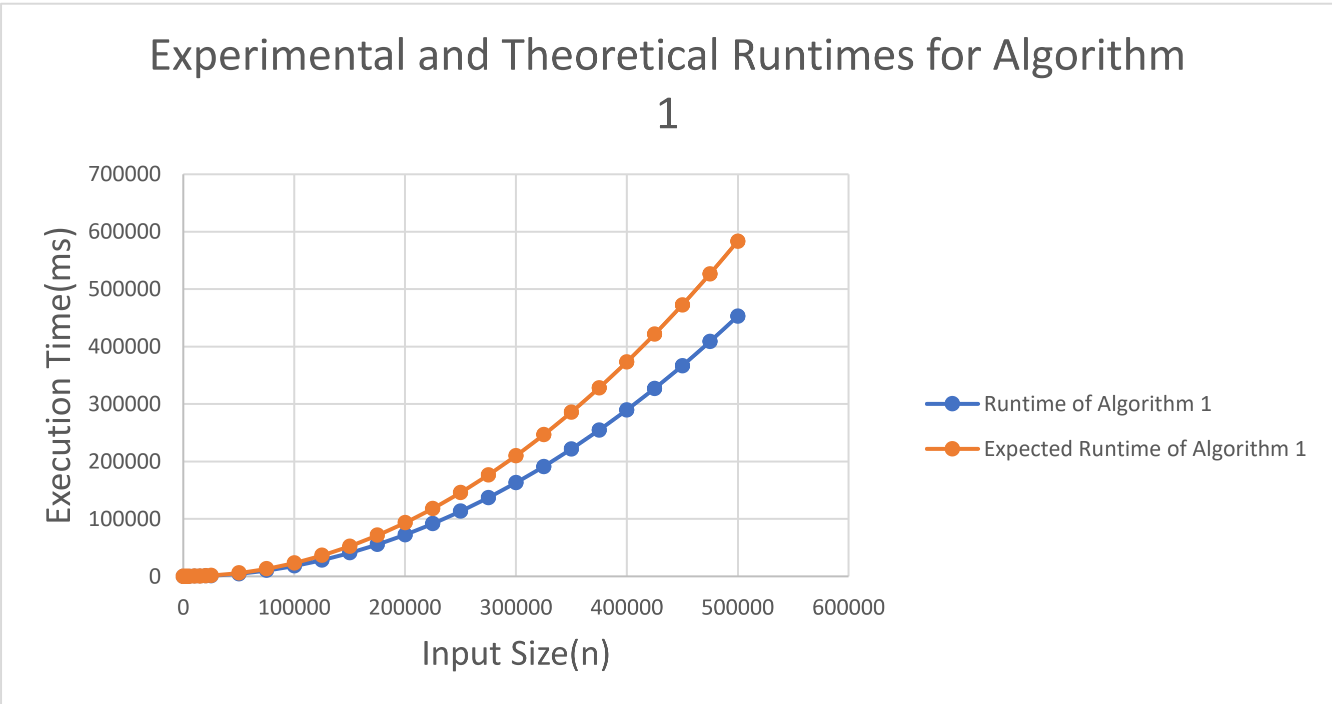
Graph 2. Graph of experimental runtimes of Algorithm 2 plotted by the data gathered from Table 1



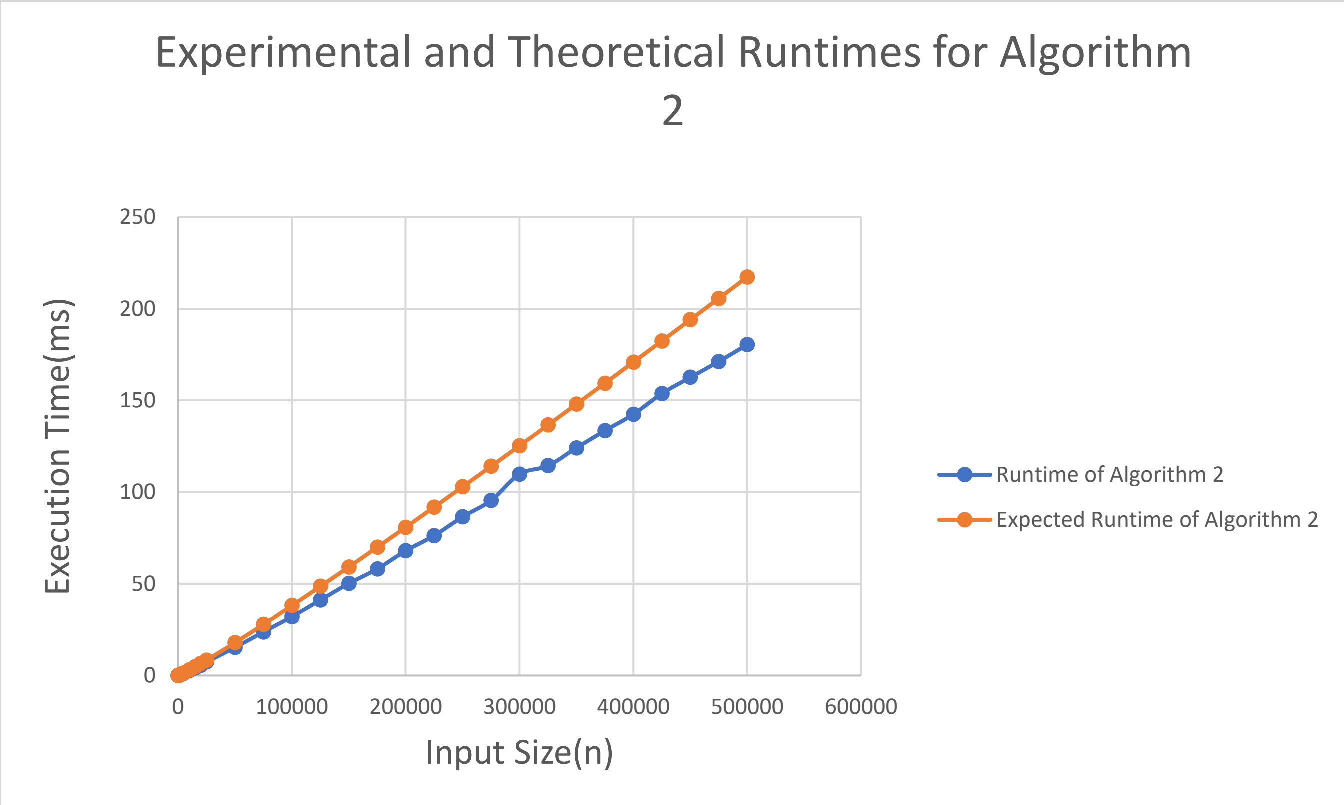
Graph 3. Graph of experimental runtimes of Algorithm 3 plotted by the data gathered from Table 1



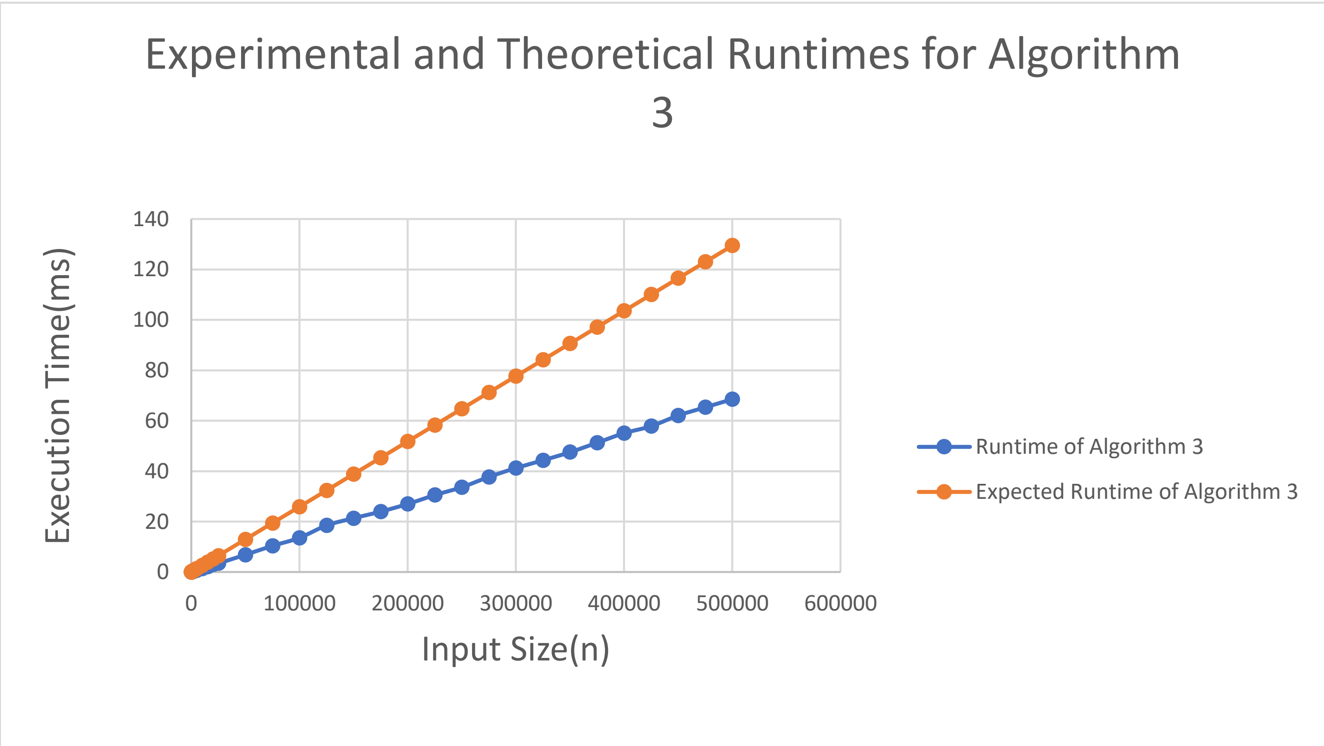
Graph 4. Graph of experimental and expected (theoretical) runtimes of Algorithm 1 plotted by the data gathered from Table 2



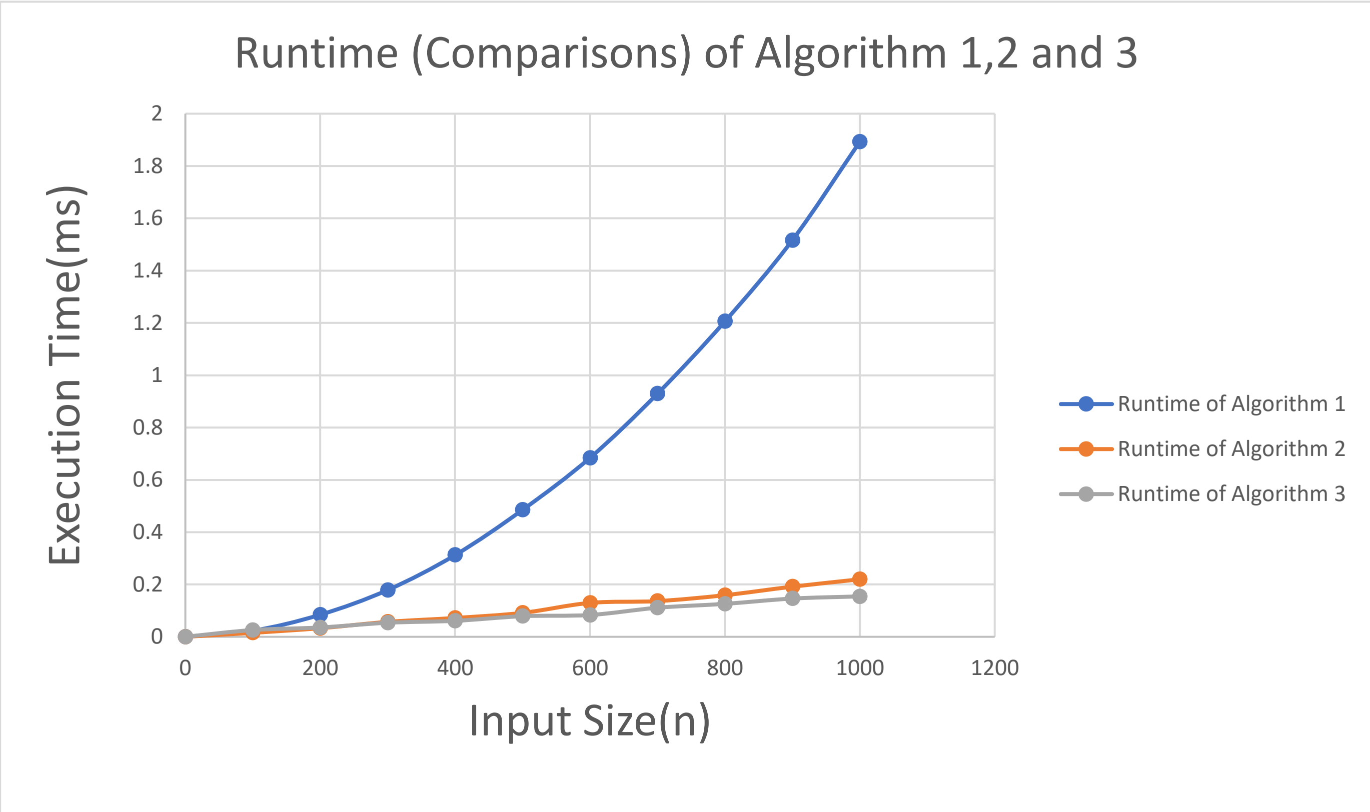
Graph 5. Graph of experimental and expected (theoretical) runtimes of Algorithm 2 plotted by the data gathered from Table 2



Graph 6. Graph of experimental and expected (theoretical) runtimes of Algorithm 3 plotted by the data gathered from Table 2



Graph 7. Graph of Comparison of the Runtimes of Different Three Algorithms



4. Discussion

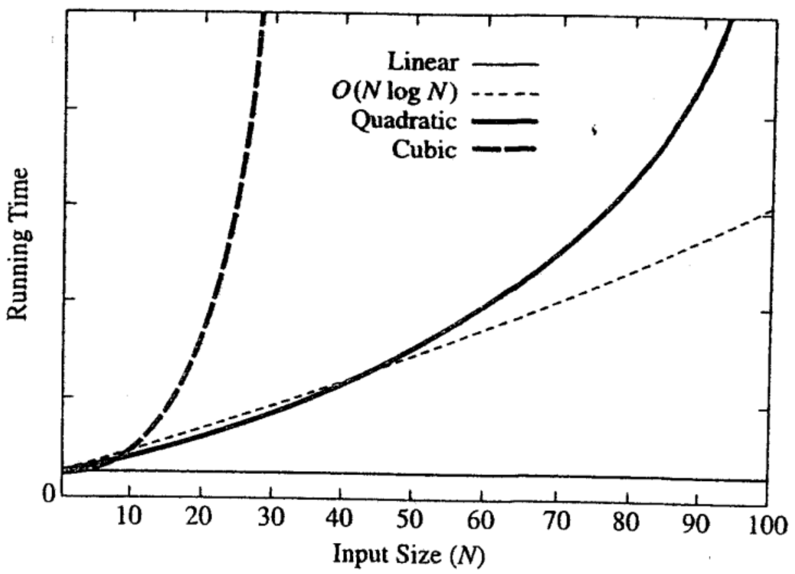


Figure 2.3 Plot (*N* vs. time) of various algorithms

According to the graphs I created from the data I received, the actual experimental data and theoretical runtimes were close to what happened. As can be seen from the graphs, the running time trend of each algorithm moves in the same trend as expected theoretically. The first point that stands out here is that the expected graph is not the same as the actual data graphs. The reason for this is that the data we calculated with Big O Notation calculates the worst cases of the algorithms. So Big O never gives us a result in seconds, milliseconds or nanoseconds. It only shows how fast the runtime of the algorithm can run according to the number of inputs.

Another important point here is that Graph 7, one of the graphs I drew, is quite similar to Figure 2.3 in the handout we worked on in the lesson.

5. Computer Specification

- **CPU (Processor):** Intel(R) Core(TM) i3-5005U CPU @ 2 GHz
- **RAM:** 4.00 GB
- **Operating System Specification:** 64-bit operating system, x64-based processor

  Edition: Windows 10 Home

  (Laptop’s brand is Casper Nirvana C350.5005-4D00X)