CS201

Section: 003

Homework 2

Metehan Saçakçı

ID: 21802788

30.11.2020

Firstly, I writted Algorithm 1 and Algorithm 2. After I finished these algorithms I check these algorithms' execution times for different cases. My procedure and results can be found below.

<u>Algorithm 1 and Case 1</u>

In this situation, all numbers in arr1 are smaller than arr2. I set both arr1's and arr2's sizes as 1000, 2000, 3000, 4000, 5000, respectively and for each different sizes, I set both arrays' elements as random integers. When I measured execution times in Algorithm 1 for each different sizes, I recorded the data and I came out with the table (Figure 1) below.

Sizes of both arr1 and arr2 (n)	1000	2000	3000	4000	5000
Execution Time of Algorithm 1 (milliseconds)	5	23	53	90	141

Figure 1.

After I came out with the table (Figure 1) above, I drew the graph (Figure 2) below with using the recorded data.

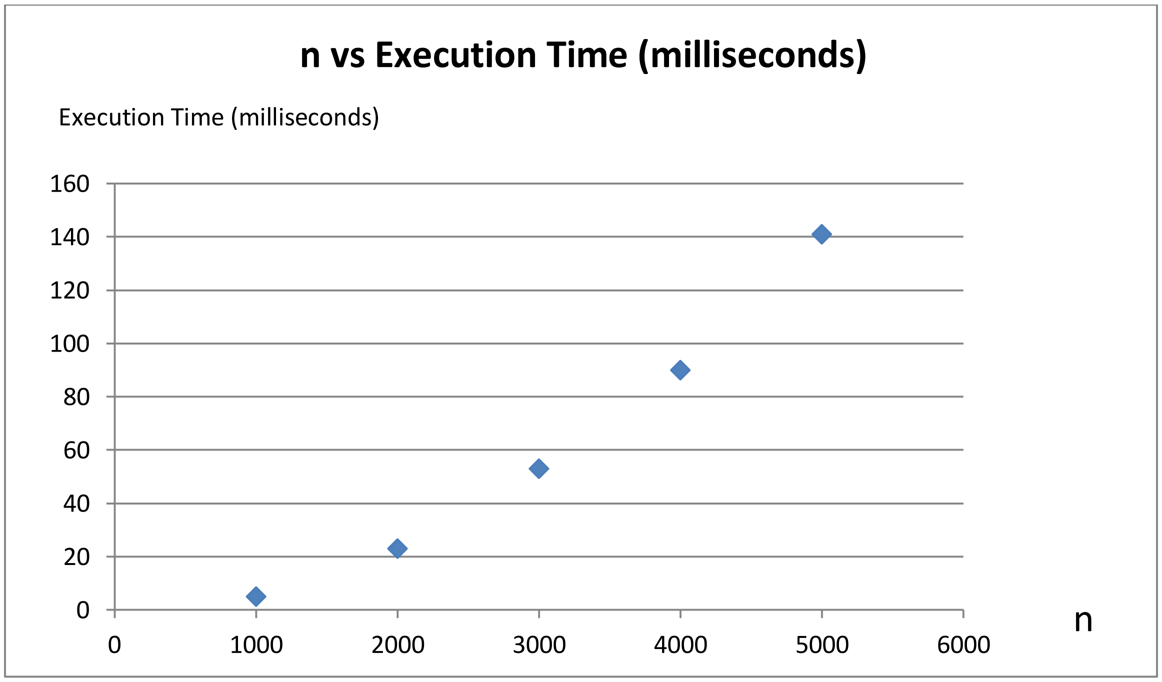

Figure 2.

Figure 2 clearly shows that when n is increased, execution time increases with quadratic growth.

Algorithm 2 and Case 1

In this situation, all numbers in arr1 are smaller than arr2. I set both arr1's and arr2's sizes as 10000000, 20000000, 30000000, 40000000, 50000000, respectively and for each different sizes, I set both arrays' elements as random integers. When I measured execution times in Algorithm 2 for each different sizes, I recorded the data and I came out with the table (Figure 3) below.

Sizes of both arr1 and arr2 (n)	10000000	20000000	30000000	40000000	50000000
Execution Time of Algorithm 2 (milliseconds)	116	228	338	461	570

Figure 3.

After I came out with the table (Figure 3) above, I drew the graph (Figure 4) below with using the recorded data.

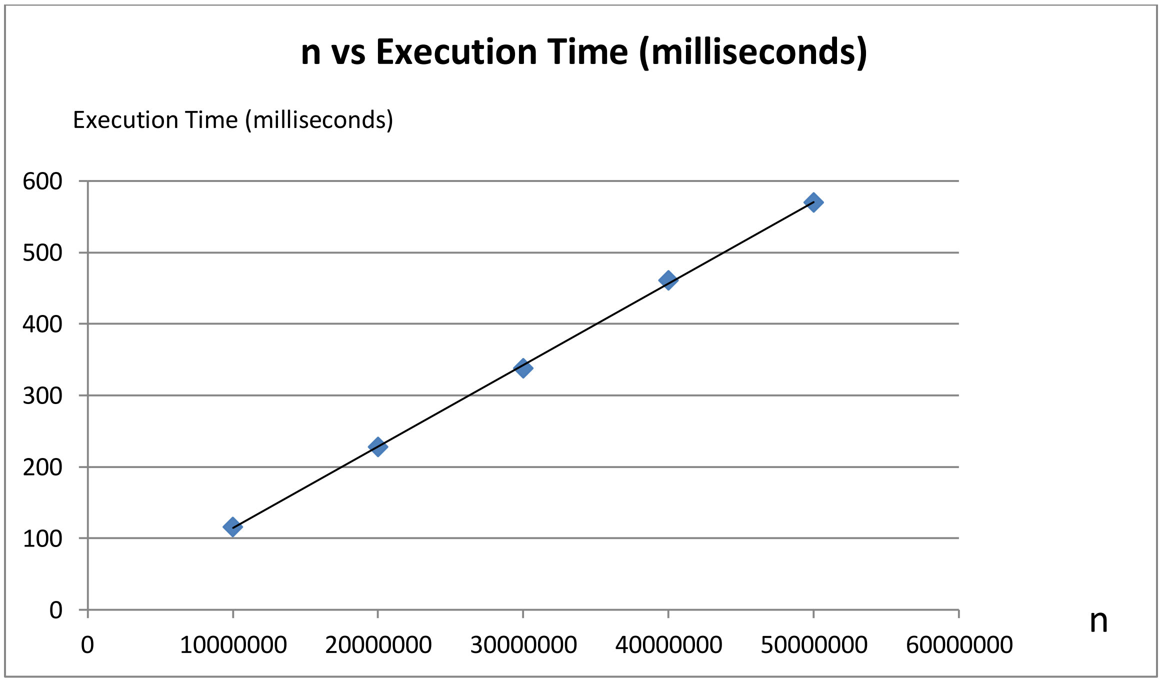

Figure 4.

Figure 4 clearly shows that when n is increased, execution time increases with linear growth.

Algorithm 1 and Case 2

In this situation, all numbers in arr2 are smaller than arr1. I set both arr1's and arr2's sizes as 1000, 2000, 3000, 4000, 5000, respectively and for each different sizes, I set both arrays' elements as random integers. When I measured execution times in Algorithm 1 for each different sizes, I recorded the data and I came out with the table (Figure 5) below.

Sizes of both arr1 and arr2 (n)	1000	2000	3000	4000	5000
Execution Time of Algorithm 1 (milliseconds)	12	21	81	141	217

Figure 5.

After I came out with the table (Figure 5) above, I drew the graph (Figure 6) below with using the recorded data.

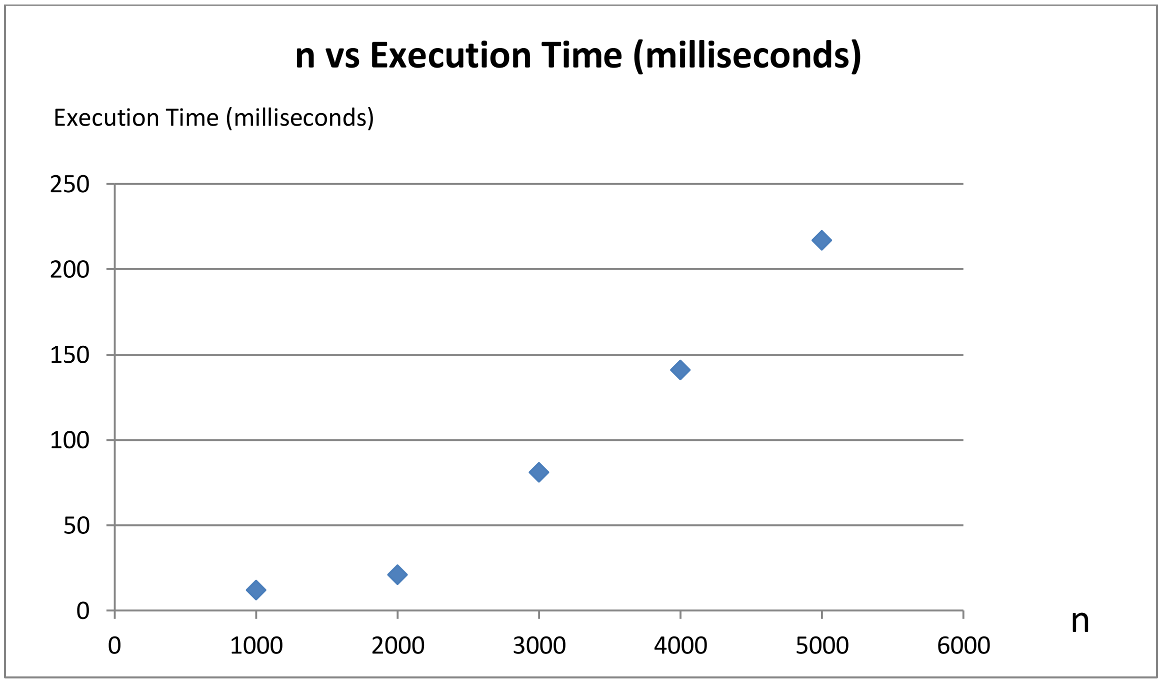

Figure 6.

Figure 6 clearly shows that when n is increased, execution time increases with quadratic growth.

Algorithm 2 and Case 2

In this situation, all numbers in arr2 are smaller than arr1. I set both arr1's and arr2's sizes as 10000000, 20000000, 30000000, 40000000, 50000000, respectively and for each different sizes, I set both arrays' elements as random integers. When I measured execution times in Algorithm 2 for each different sizes, I recorded the data and I came out with the table (Figure 7) below.

Sizes of both arr1 and arr2 (n)	10000000	20000000	30000000	40000000	50000000
Execution Time of Algorithm 2 (milliseconds)	130	261	392	530	655

Figure 7.

After I came out with the table (Figure 7) above, I drew the graph (Figure 8) below with using the recorded data.

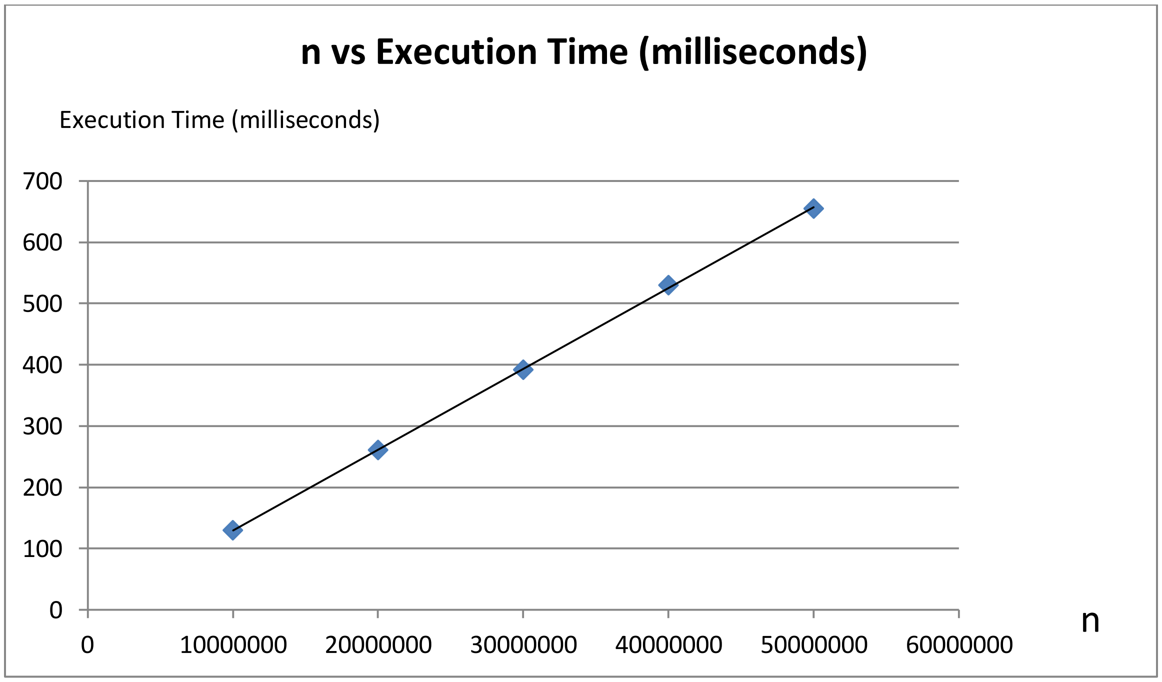

Figure 8.

Figure 8 clearly shows that when n is increased, execution time increases with linear growth.

Algorithm 1 and Case 3

In this situation, all numbers in arr2 are smaller than arr1. I set both arr1's and arr2's sizes as 1000, 2000, 3000, 4000, 5000, respectively and for each different sizes, I set both arrays' elements as random integers. When I measured execution times in Algorithm 1 for each different sizes, I recorded the data and I came out with the table (Figure 9) below.

Sizes of both arr1 and arr2 (n)	1000	2000	3000	4000	5000
Execution Time of Algorithm 1 (milliseconds)	9	23	64	114	178

Figure 9.

After I came out with the table (Figure 9) above, I drew the graph (Figure 10) below with using the recorded data.

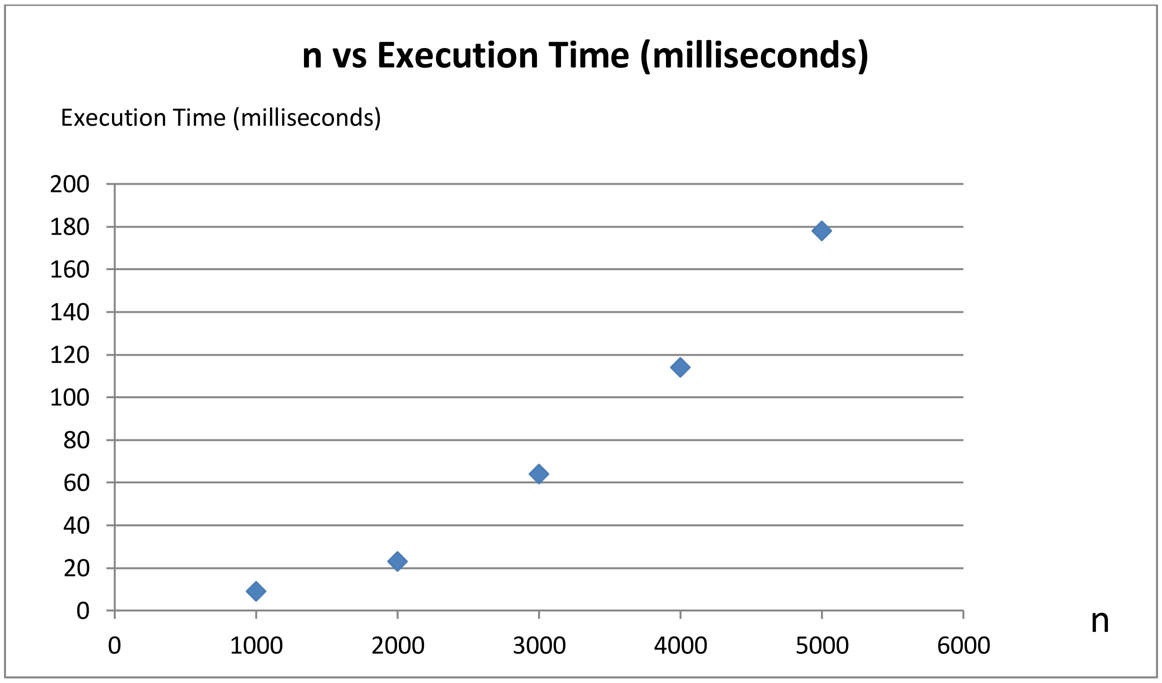

Figure 10.

Figure 10 clearly shows that when n is increased, execution time increases with quadratic growth.

<u>Algorithm 2 and Case 3</u>

In this situation, all numbers in arr2 are smaller than arr1. I set both arr1's and arr2's sizes as 10000000, 20000000, 30000000, 40000000, 50000000, respectively and for each different sizes, I set both arrays' elements as random integers. When I measured execution times in Algorithm 2 for each different sizes, I recorded the data and I came out with the table (Figure 11) below.

Sizes of both arr1 and arr2 (n)	10000000	20000000	30000000	40000000	50000000
Execution Time of Algorithm 2 (milliseconds)	112	218	328	446	557

Figure 11.

After I came out with the table (Figure 11) above, I drew the graph (Figure 12) below with using the recorded data.

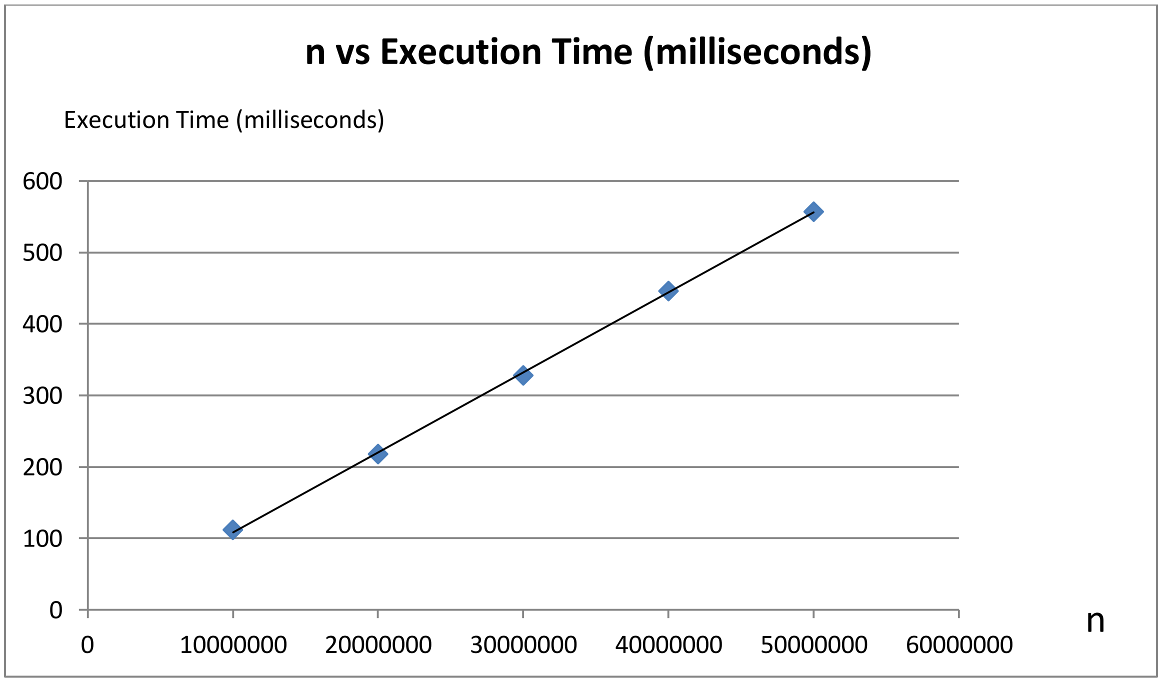

Figure 12.

Figure 12 clearly shows that when n is increased, execution time increases with linear growth.

Comparison of Algorithm 1 With Using Each Cases

When all of the graphs about Algorithm 1 (Figure 2, Figure 6, Figure 10) one are combined, I come out with the graph (Figure 13) below.

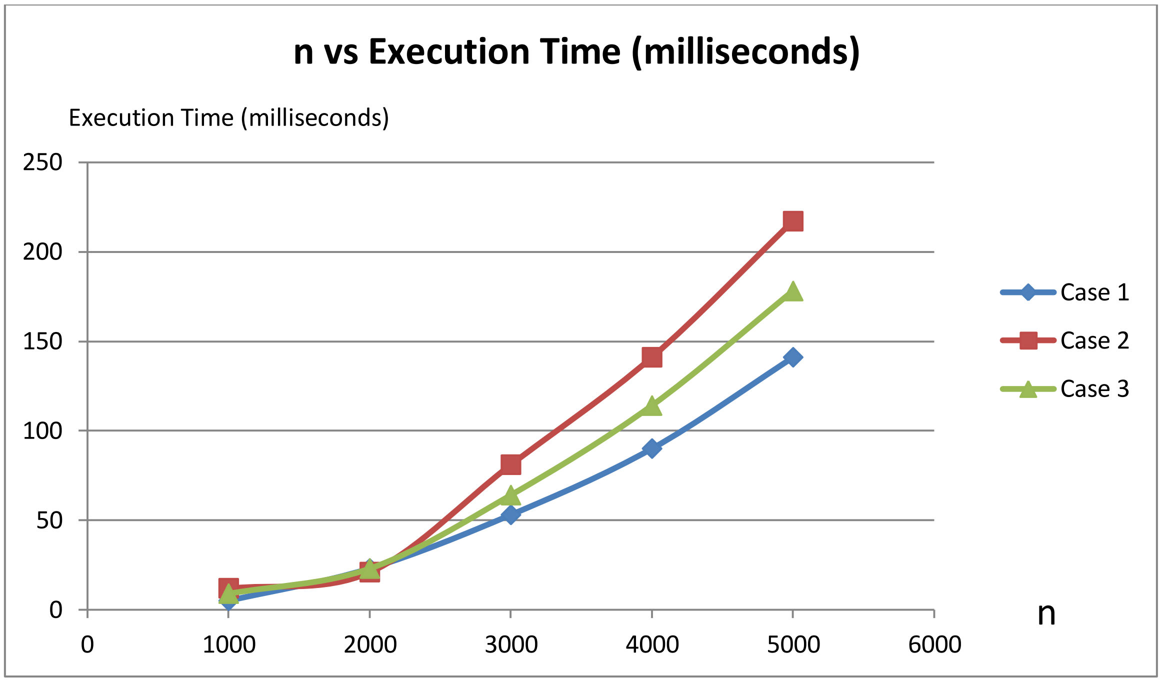

Figure 13.

When Algorithm 1 is compared with each case individually, it can be seen that for $n = 5000$, case 1 took 141 milliseconds, case 2 took 217 milliseconds and case 3 took 178 milliseconds. This observation, the graph (Figure 13) above and other observations that comes from other input values show that:

- Case 1 is the best case. -> O(n^2)
- Case 2 is the worst case. -> O(n^2)
- Case 3 is the average case. -> O(n^2)

Because there is a quadratic growth when input size is increased for all of the cases, their growth must be O(n^2).

Comparison of Algorithm 2 With Using Each Cases

When all of the graphs about Algorithm 2 (Figure 4, Figure 8, Figure 12) one are combined, I come out with the graph (Figure 14) below.

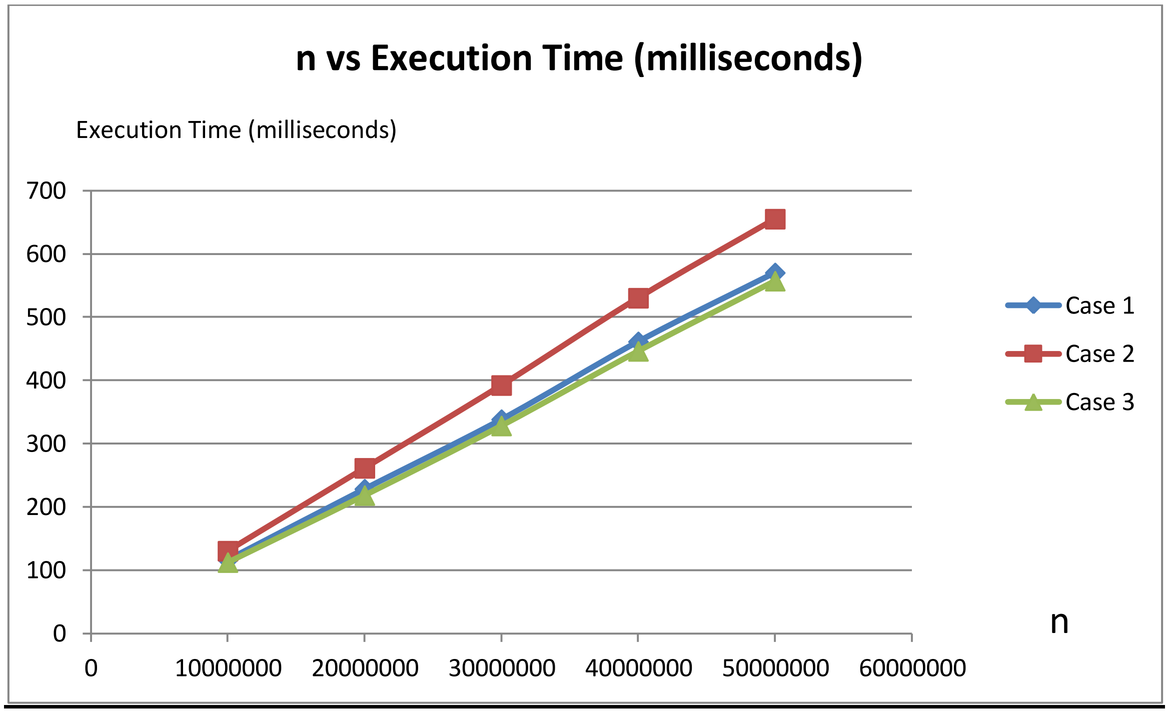

Figure 14.

When Algorithm 2 is compared with each case individually, it can be seen that for $n = 50000000$, case 1 took 570 milliseconds, case 2 took 655 milliseconds and case 3 took 557 milliseconds. This observation, the graph (Figure 14) above and other observations that comes from other input values show that:

- Case 3 is the best case. -> O(n)
- Case 2 is the worst case. -> O(n)
- Case 1 is the average case. -> O(n)

Because there is a linear growth when input size is increased for all of the cases, their growth must be O(n).

Detailed Information About The Setup That Test Is Done

This test is done in the computer that has:

Processor: Intel® Core™ i7-3770 CPU @ 3.40GHz

RAM: 16.0 GB

Operating System: Windows 10

Expected Worst Case Growth Rate of Algorithm 1

For Algorithm 1, I determined that worst case's growth rate should be O(n^2). That is why if I put n sized array, n^2 operations should occur. The expected graph (Figure 15) with the input sizes, that I used for Algorithm 1, can be found below.

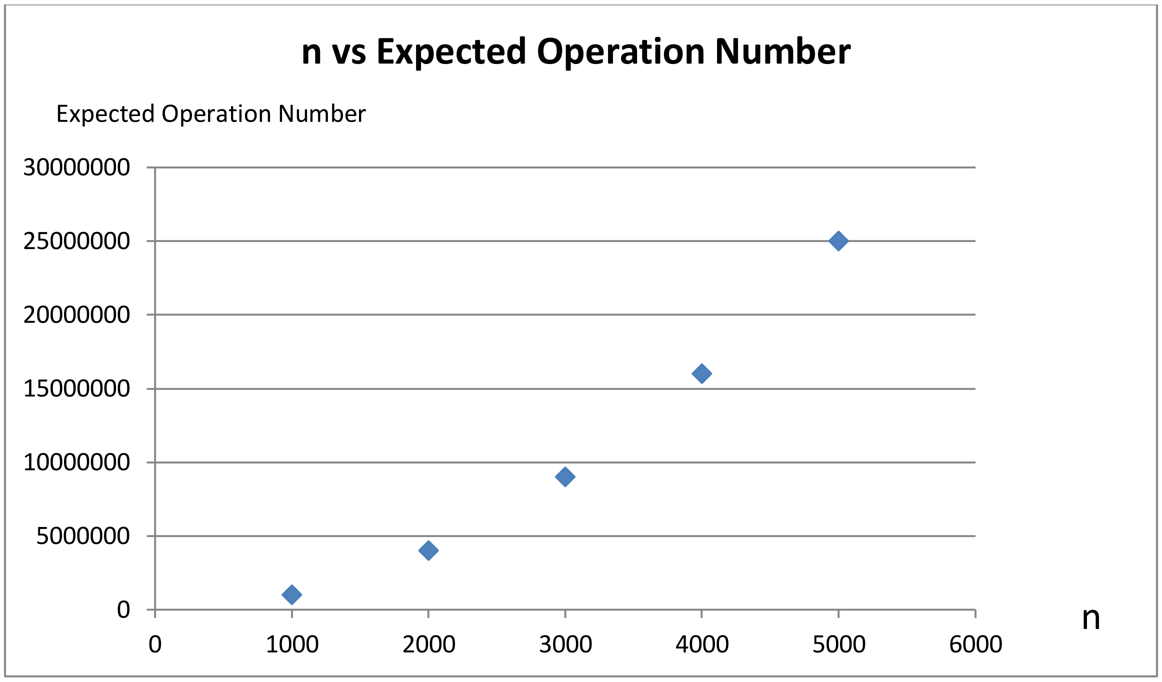

Figure 15.

Expected Worst Case Growth Rate of Algorithm 2

For Algorithm 2, I determined that worst case's growth rate should be O(n). That is why if I put n sized array, n operations should occur. The expected graph (Figure 16) with the input sizes, that I used for Algorithm 2, can be found below.

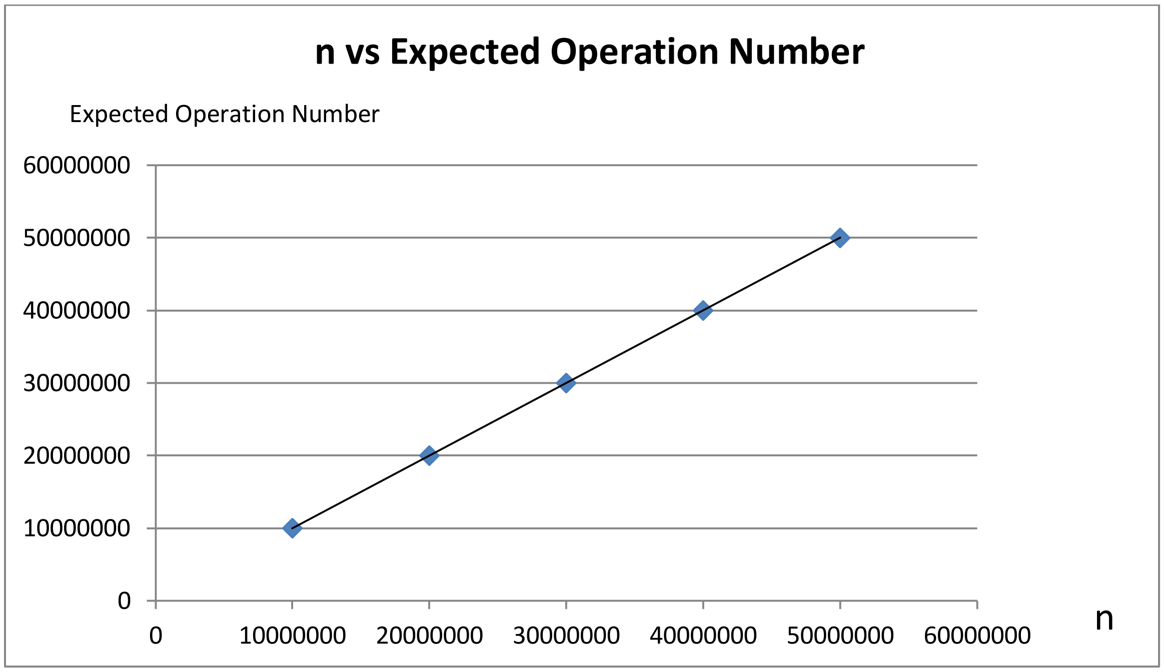

Figure 16.

Conclusion for Algorithm 1

In Algorithm 1, according to the result of all of the cases, it can be concluded that the worst case scenario has quadratic growth rate. That is why it should be expected that in this algorithm, there will be operation for squared of n value times. Because the graphs Figure 13 and Figure 15 that comes from observation and expectation respectively are accurate, I concluded that my deduction is correct for Algorithm 1.

Conclusion for Algorithm 2

In Algorithm 2, according to the result of all of the cases, it can be concluded that the worst case scenario has linear growth rate. That is why it should be expected that in this algorithm, there will be operation for n value times. Because the graphs Figure 14 and Figure 16 that comes from observation and expectation respectively are accurate, I concluded that my deduction is correct for Algorithm 2.