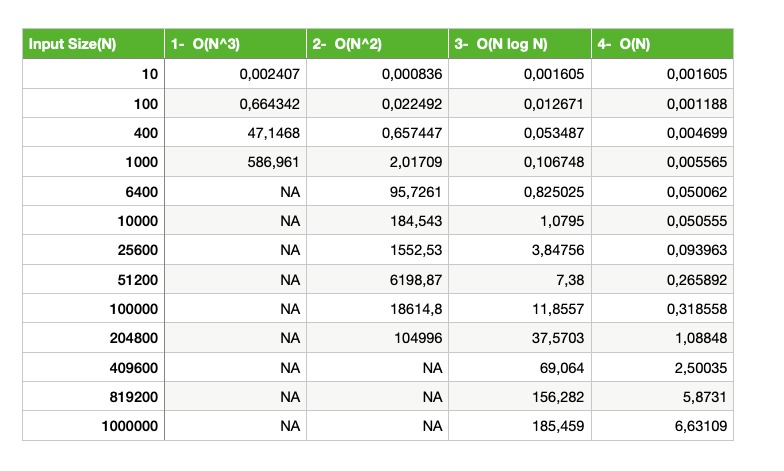
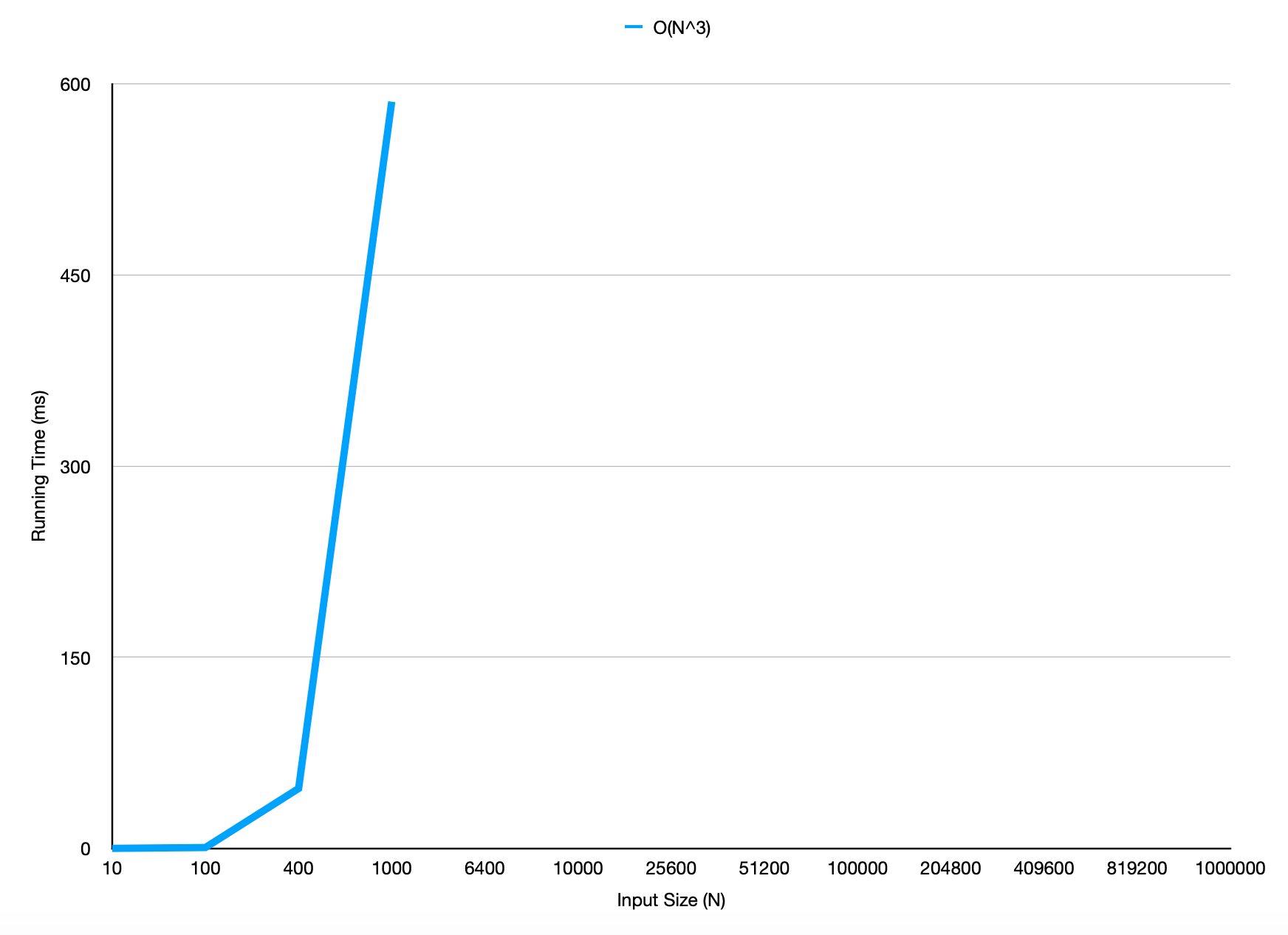
# CS201 Homework 2

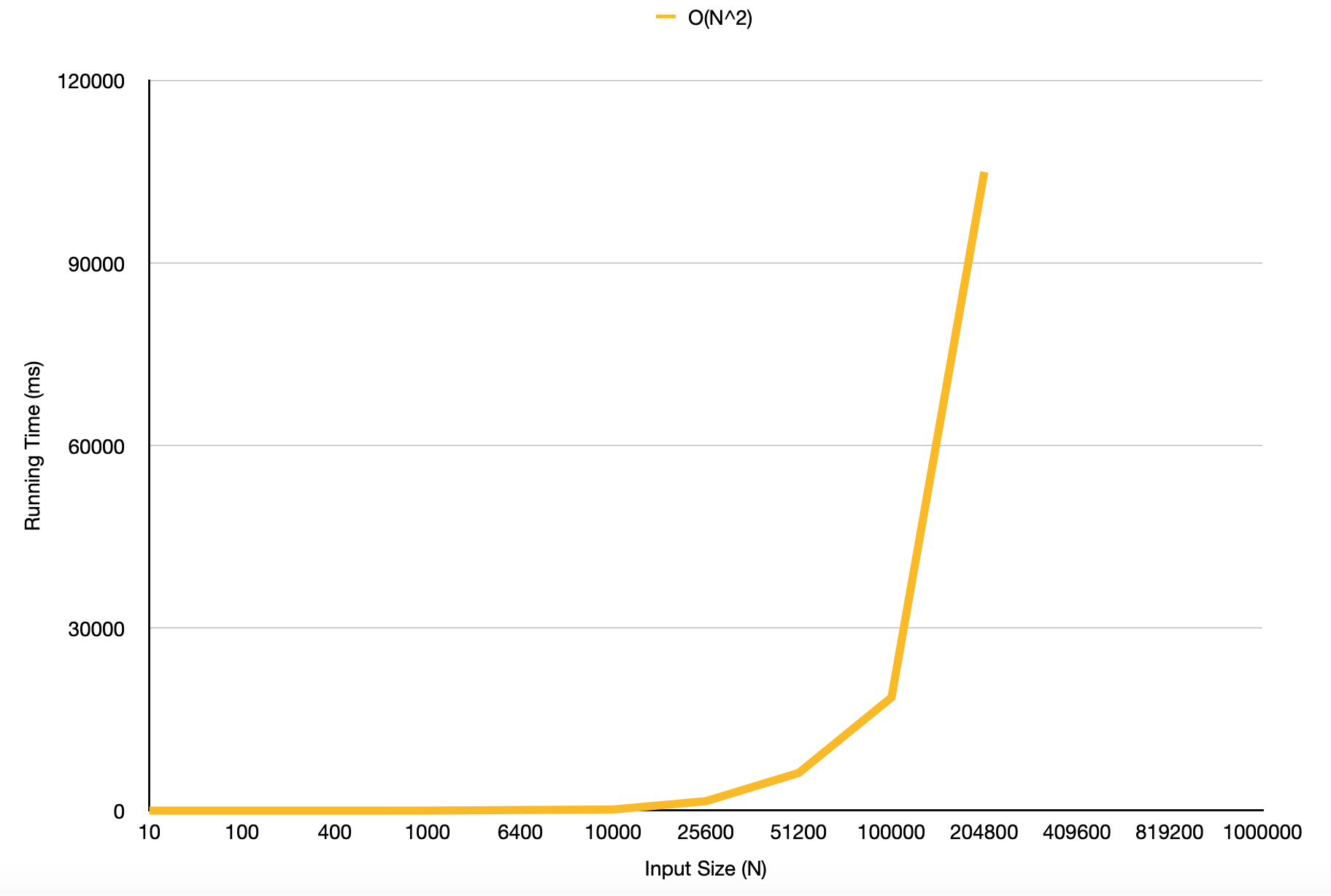
1- In the table below, running times of four algorithms for maximum subsequence sum problem are concluded for different input sizes.



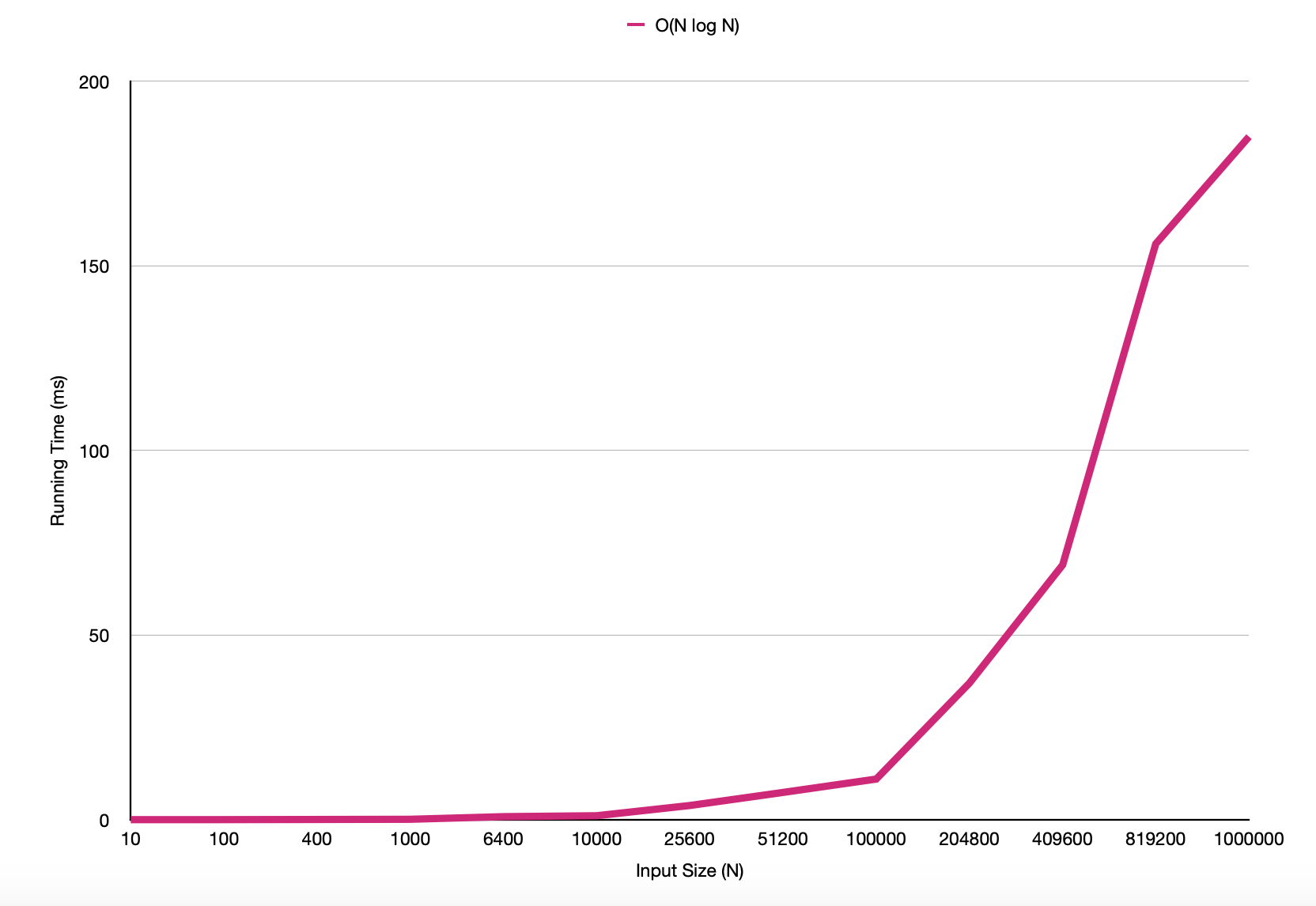
* The input size(N)-running time(ms) graph of the first algorithm:



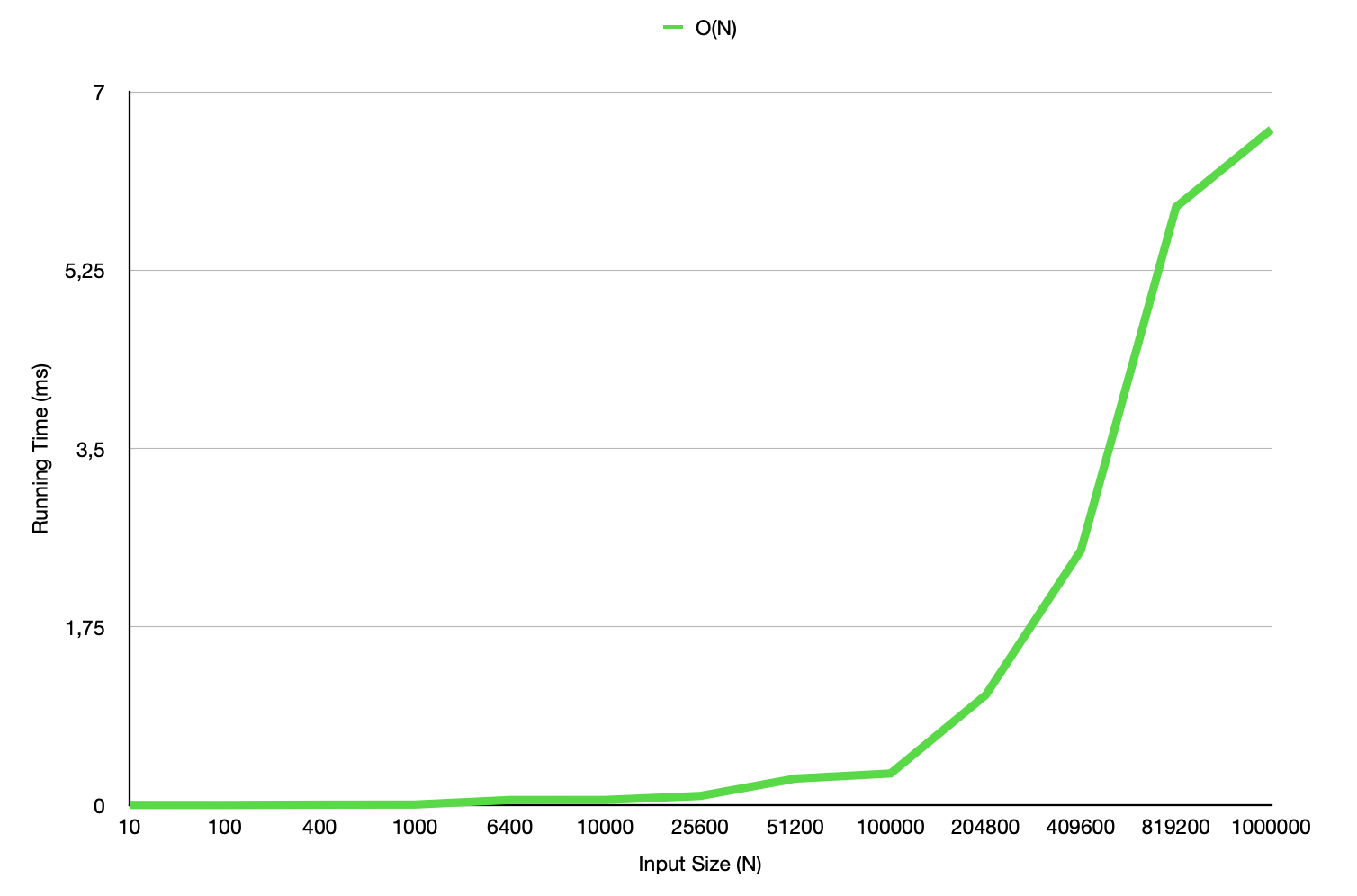
* The input size(N)-running time(ms) graph of the second algorithm:



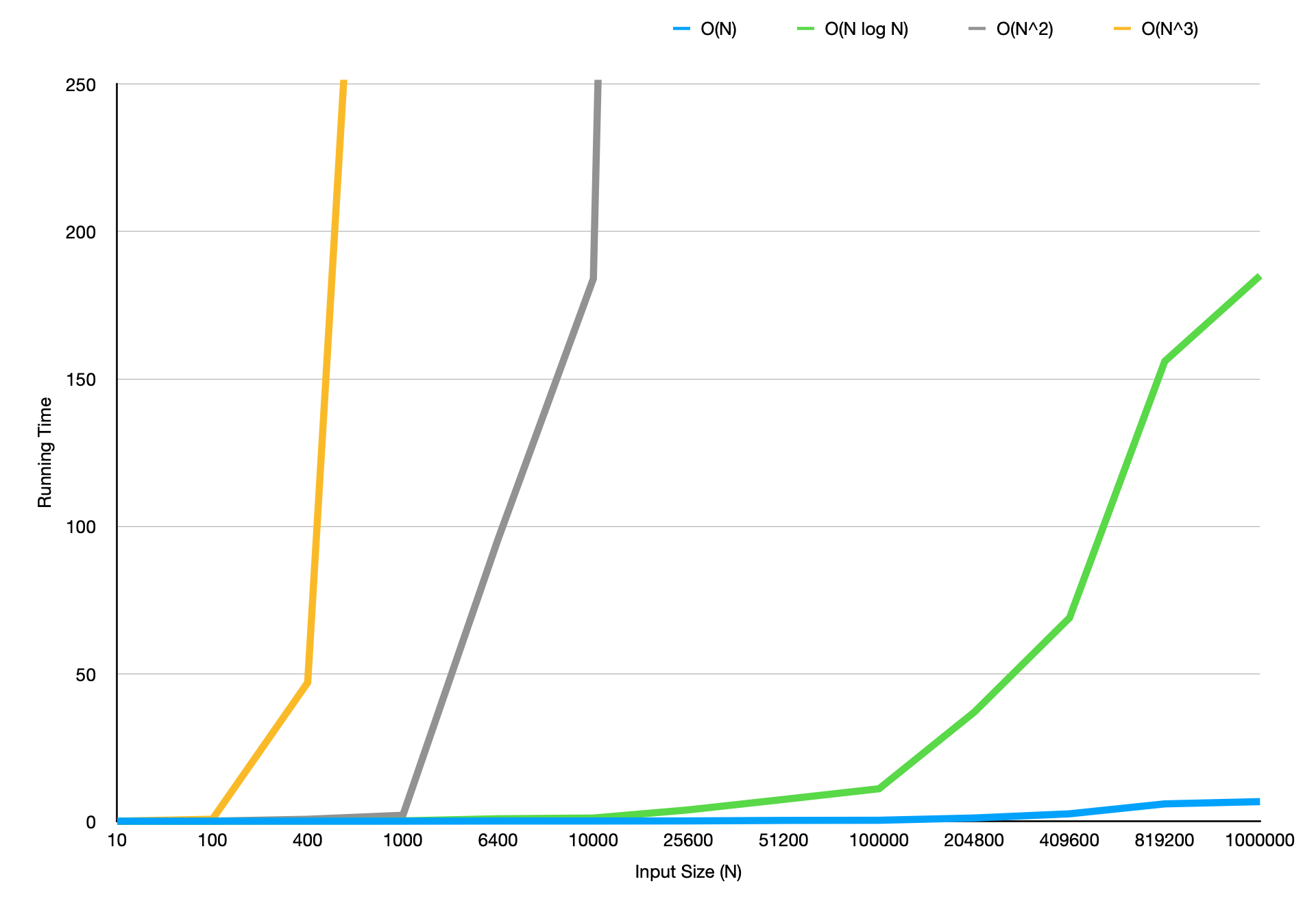
* The input size(N)-running time(ms) graph of the third algorithm:



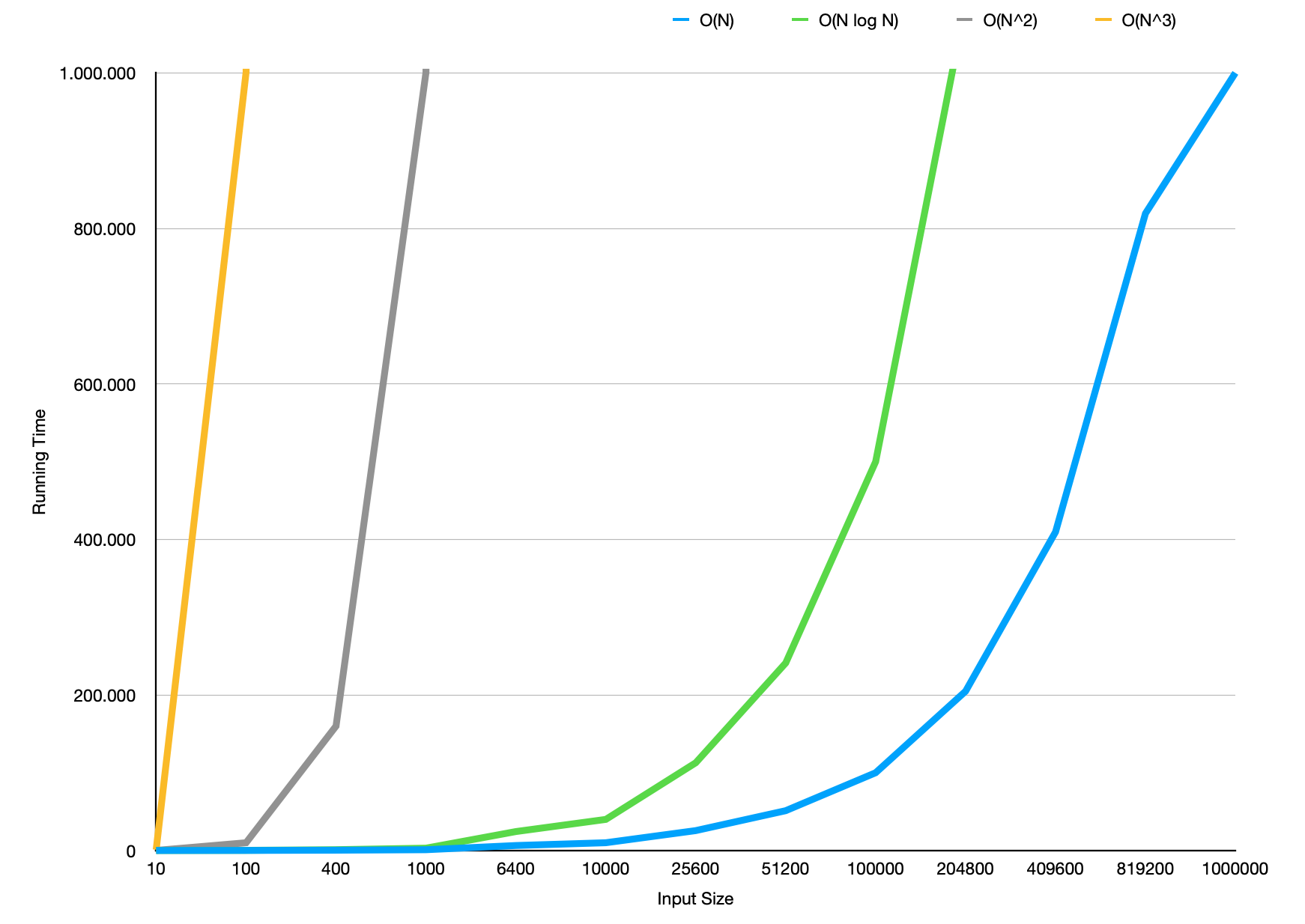
* The input size(N)-running time(ms) graph of the fourth algorithm:



* The input size(N)-running time(ms) graph of all algorithms:



* The graph of expected growth rates:



2- Theoretical Analysis:

The expected growth rate is like the graph above. Since the value ranges on the y-axis are different from each other, the graphs may seem different from each other. However, we can see that the result is quite similar when N values are examined one by one. Even if this increase does not seem obvious enough for small N values, we can see that the difference increases especially after N = 100.

In this example, all algorithms use the worst case to examine the array to the end. There is no such thing as a best case for this example. Because the function needs to compare all possible subsequences with each other in order to find the maximum value. For the first algorithm, since there are 3 nested for loops running from 0 to N, this algorithm runs N^3 times for the worst case. So, the time complexity of this function is O(N^3). In the expected graph and the experimental graph, we can see that N^3 increases exponentially relative to other functions when we change the value of N. Also, the value table shows that from an N value between 1000 and 6400, the running time increases too much and we cannot get the output. This means that the first algorithm has the worst performance.

For the second algorithm, since there are 2 nested for loops running from 0 to N, this algorithm runs N^2 times for the worst case. So, the time complexity of this function is O(N^2).

Even though it is faster than O(N^3) for big N values, the running time of the N^2 function also increases exponentially. This means that this function is more effective than the first function.

For the third algorithm which is a recursive function, we divide the array into two until we find the maximum sum. This process repeats N times. So, the time complexity of this function is O(N log N). As we can see in both expected and obtained results, this function is more effective than the first two functions. However, for big N values, dividing a large array into two until finding the maximum sum value is not effective.

The last algorithm which contains only one for loop, performs best. Because the time complexity of this function is O(N). The graph is linear. As we can see from my results, even if N value is 1 million, running time is quite small and it is a more effective function than others.

3- The Specifications of my computer:

