Bilkent University
Cs201 HW2 Report

Hacı Çakın
21802641
Section 01

1)Specifications

- CPU: 9th gen Core i7
- RAM: 16GB
- Operating System: macOS BigSur
- IDE: IntelliJ Clion

2)Data

A) For the all numbers in arr1 are smaller than arr2

A.1) For the first algorithm

Execution took 3.089 milliseconds. This is for the first algorithm with size 1000

Execution took 12.4 milliseconds. This is for the first algorithm with size 2000

Execution took 27.41 milliseconds. This is for the first algorithm with size 3000

Execution took 44.521 milliseconds. This is for the first algorithm with size 4000

Execution took 70.322 milliseconds. This is for the first algorithm with size 5000

Execution took 88.079 milliseconds. This is for the first algorithm with size 6000

Execution took 112.502 milliseconds. This is for the first algorithm with size 7000

Execution took 147.48 milliseconds. This is for the first algorithm with size 8000
Execution took 183.433 milliseconds. This is for the first algorithm with size 9000
Execution took 243.679 milliseconds. This is for the first algorithm with size 10000
A.2) For the second algorithm
Execution took 0.009 milliseconds. This is for the second algorithm with size 1000
Execution took 0.016 milliseconds. This is for the second algorithm with size 2000
Execution took 0.03 milliseconds. This is for the second algorithm with size 3000
Execution took 0.034 milliseconds. This is for the second algorithm with size 4000
Execution took 0.049 milliseconds. This is for the second algorithm with size 5000
Execution took 0.049 milliseconds. This is for the second algorithm with size 6000
Execution took 0.057 milliseconds. This is for the second algorithm with size 7000
Execution took 0.064 milliseconds. This is for the second algorithm with size 8000
Execution took 0.076 milliseconds. This is for the second algorithm with size 9000
Execution took 0.085 milliseconds. This is for the second algorithm with size 10000

B) For the all numbers in arr2 are smaller than arr1

B.1) For the first algorithm
Execution took 3.206 milliseconds. This is for the first algorithm with size 1000
Execution took 13.408 milliseconds. This is for the first algorithm with size 2000
Execution took 30.82 milliseconds. This is for the first algorithm with size 3000
Execution took 55.631 milliseconds. This is for the first algorithm with size 4000
Execution took 83.851 milliseconds. This is for the first algorithm with size 5000
Execution took 112.488 milliseconds. This is for the first algorithm with size 6000
Execution took 135.507 milliseconds. This is for the first algorithm with size 7000
Execution took 181.414 milliseconds. This is for the first algorithm with size 8000
Execution took 222.085 milliseconds. This is for the first algorithm with size 9000
Execution took 275.51 milliseconds. This is for the first algorithm with size 10000

B.2) For the second algorithm
Execution took 0.009 milliseconds. This is for the second algorithm with size 1000

Execution took 0.015 milliseconds. This is for the second algorithm with size 2000
Execution took 0.021 milliseconds. This is for the second algorithm with size 3000
Execution took 0.027 milliseconds. This is for the second algorithm with size 4000
Execution took 0.035 milliseconds. This is for the second algorithm with size 5000
Execution took 0.041 milliseconds. This is for the second algorithm with size 6000
Execution took 0.049 milliseconds. This is for the second algorithm with size 7000
Execution took 0.055 milliseconds. This is for the second algorithm with size 8000
Execution took 0.062 milliseconds. This is for the second algorithm with size 9000
Execution took 0.068 milliseconds. This is for the second algorithm with size 10000

C) For the there is no such ordering between these array

C.1) For the first algorithm

Execution took 2.435 milliseconds. This is for the first algorithm with size 1000
Execution took 9.791 milliseconds. This is for the first algorithm with size 2000
Execution took 22.231 milliseconds. This is for the first algorithm with size 3000
Execution took 40.953 milliseconds. This is for the first algorithm with size 4000
Execution took 63.639 milliseconds. This is for the first algorithm with size 5000
Execution took 93.059 milliseconds. This is for the first algorithm with size 6000
Execution took 134.489 milliseconds. This is for the first algorithm with size 7000
Execution took 162.689 milliseconds. This is for the first algorithm with size 8000
Execution took 201.521 milliseconds. This is for the first algorithm with size 9000
Execution took 251.978 milliseconds. This is for the first algorithm with size 10000

C.2) For the second algorithm

Execution took 0.016 milliseconds. This is for the second algorithm with size 1000
Execution took 0.049 milliseconds. This is for the second algorithm with size 2000
Execution took 0.056 milliseconds. This is for the second algorithm with size 3000
Execution took 0.058 milliseconds. This is for the second algorithm with size 4000
Execution took 0.105 milliseconds. This is for the second algorithm with size 5000
Execution took 0.086 milliseconds. This is for the second algorithm with size 6000
Execution took 0.132 milliseconds. This is for the second algorithm with size 7000

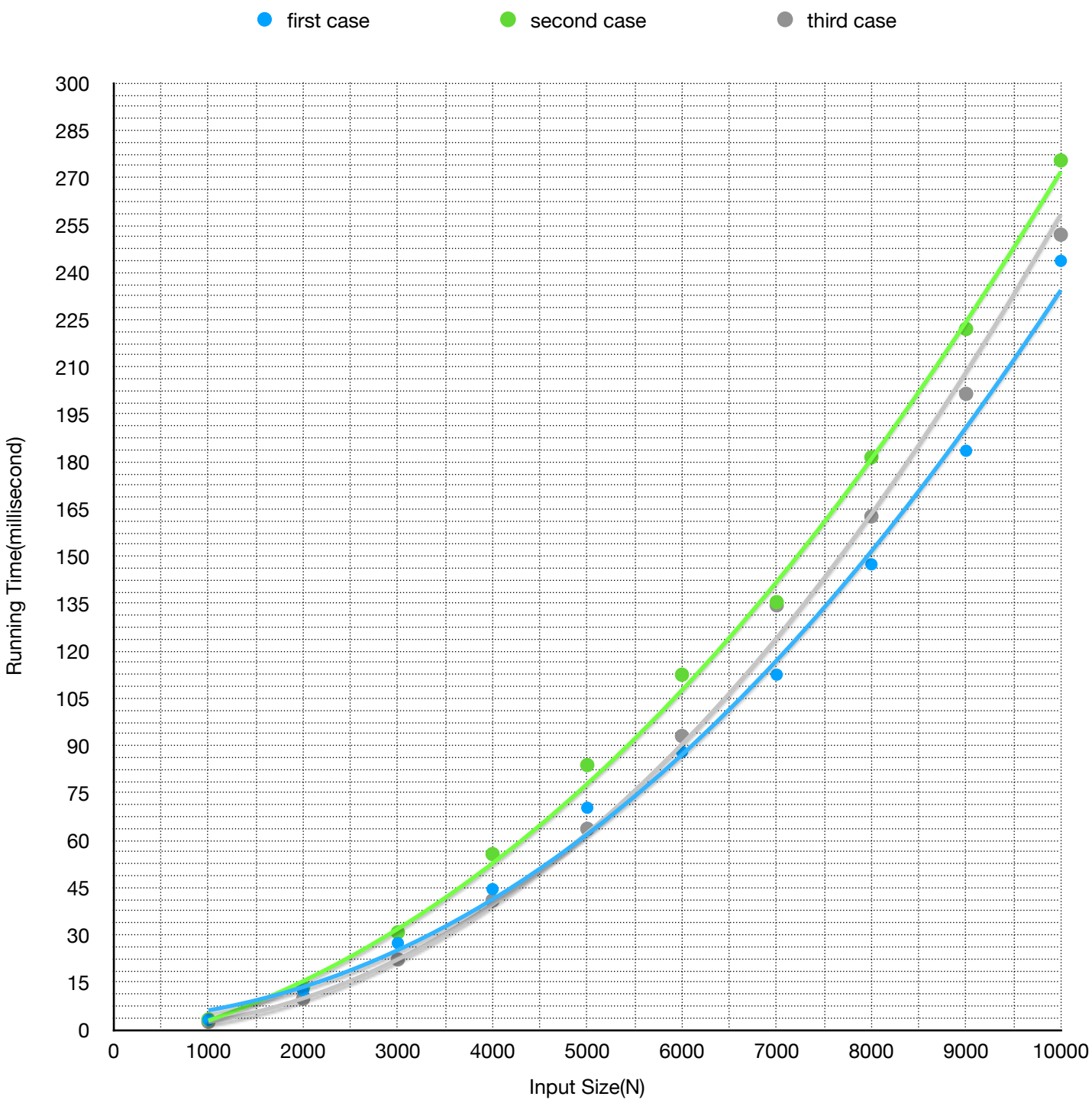
Execution took 0.146 milliseconds. This is for the second algorithm with size 8000

Execution took 0.16 milliseconds. This is for the second algorithm with size 9000

Execution took 0.17 milliseconds. This is for the second algorithm with size 10000

3)Plots

3.1) For the first algorithm



- Best Case: It is first case.
- Worst Case: It is second case
- Average Case: It is third case

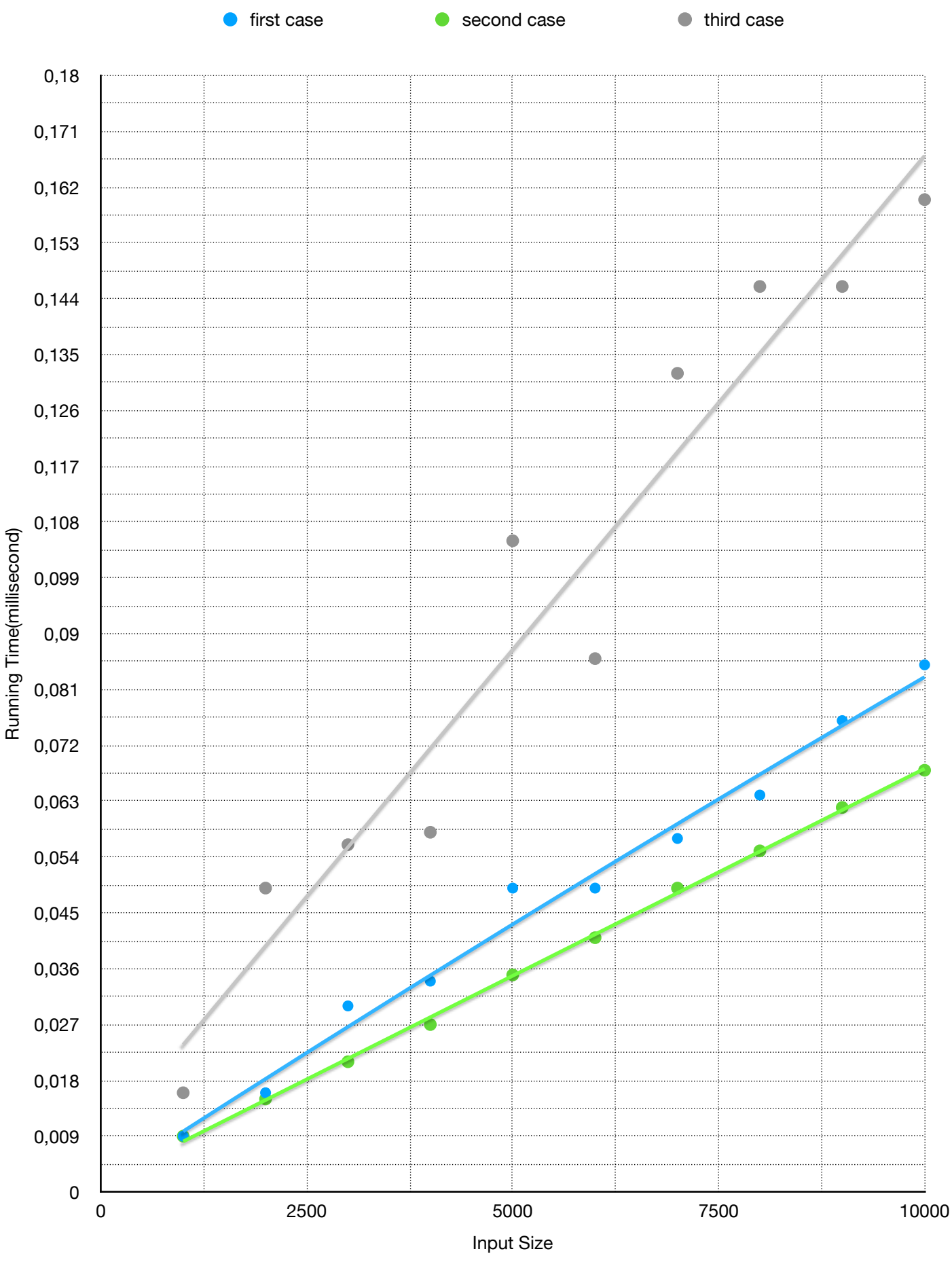
All of the cases has same grow rate which is O(N^2). Their coefficients are different but this does not matter. The reason of why second case is best case is that in my code if arr3' elements are smaller then from the arr2, it enters the if block which has a continue statement. However, finally it enters else block in algorithm1. This block has another for loop so it is the reason of it's growth rate(O(N^2)). For the worst case, reverse case happened and elements which are in arr2 directly shift the list and added. In the for loop which is in the else block always runs with maximum number. Therefore, even it has same growth rate, it is bigger than others. As it is expected third case which is between them has bigger growth rate than first case, smaller growth rate than the second case.

So

Big-Oh O(N^2)
Big-omega Ω(N^2)

Big-theta Θ(N^2)

3.2) For the second algorithm



- Best Case: It is second case.
- Worst Case: It is third case
- Average Case: It is first case

Each of cases for the algorithm2 have growth rate N. Main reason of this is that for each case loop runs with number of 2n. Therefore, it's growth rate is N. A reason of the differences between first case and second case is caused because of the if block. They have same number of operations but first case always control one more if block. Therefore, its coefficient is bigger but it is not shown in the notation. Therefore, it can not beat the second case(in my code). However, these are not valid for the third case. Because it is randomly selected, it can be placed anywhere for each trial. Which means that its coefficient will change each trial while depending on the random lists.

So

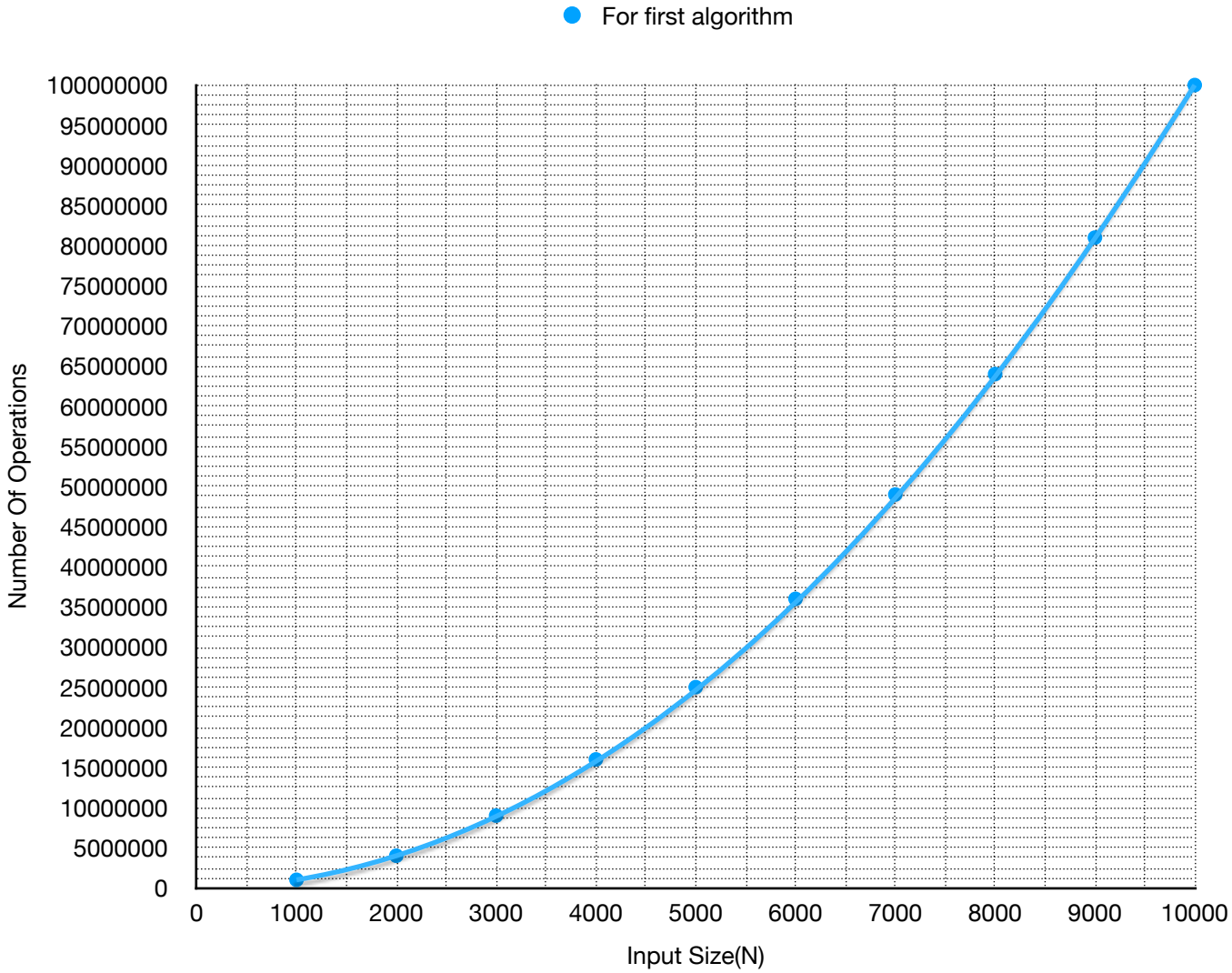
Big-Oh O(N)
Big-omega Ω(N)

Big-theta Θ(N)

4)Worst Cases

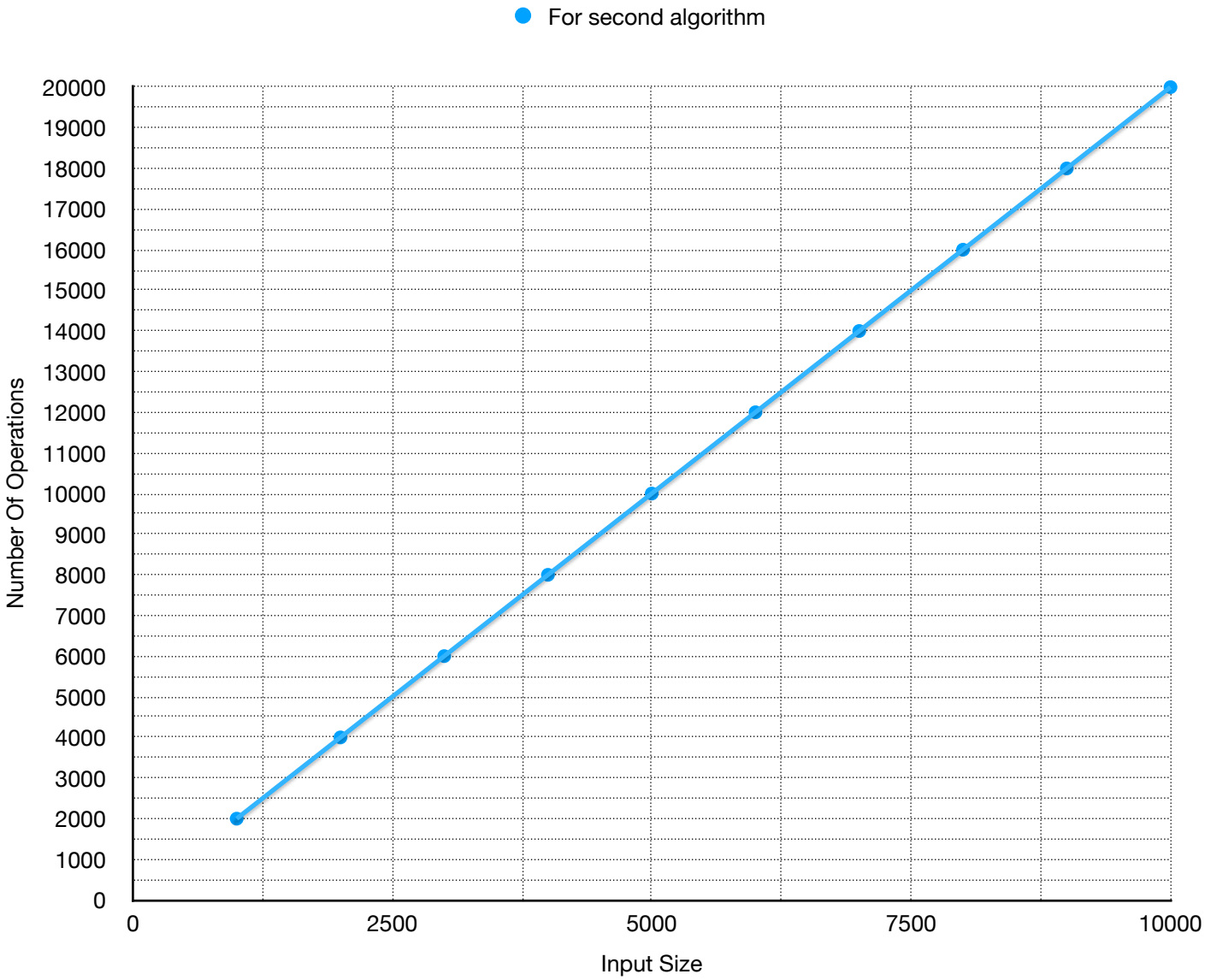
4.1) For the first algorithm

This plot has same character with the worst case in the first algorithm. As I said before, their growth rate is direct proportional to N^2. The only difference between them is the coefficients. However, coefficients are not shown in the notation of expectation. The



reason of the why number of operations are direct proportional to N^2 is that for each n elements method runs loop n times(worst). In the worst case, this will be bigger than because it always exceed exact N^2 which means that its coefficient will be always bigger than 1. What I try to say is that in the worst case there are n elements and for each elements loop runs with number x>n(valid for my code because in the second loop, number of turns is depending on how much element are already added). However, coefficients are not that matter, therefore the theoretical analysis matches the experimental result.

4.2) For the second algorithm



In this plot, there is a pattern which is similar to the worst case in the second algorithm. As I said before, their growth rate is direct proportional to N. The only difference between them is the coefficients again. However, coefficients are not shown in the notation of expectation. The reason of the why number of operations are direct proportional to N is that for each n elements method runs loop 2 times(worst). In the worst case, growth rate will be direct proportional to this line which means that slope of the line will give the derivative of the growth rate(if we calculated the growth rate with its coefficient). On the other hand, we don't look the coefficients, so experimental result and theatricals assumption are similar to each other.

Important Note : I uploaded and run it in the server. My code works fine but, server does not show decimals. Therefore, for the second algorithm, it shows 0 as a time(but values are not 0, they are 0.008 etc.). I also tried test with much bigger values for just second algorithm but both server and my IDE do not accept the that tremendous values. I really tried it too much. Therefore, I added my IDE's output.