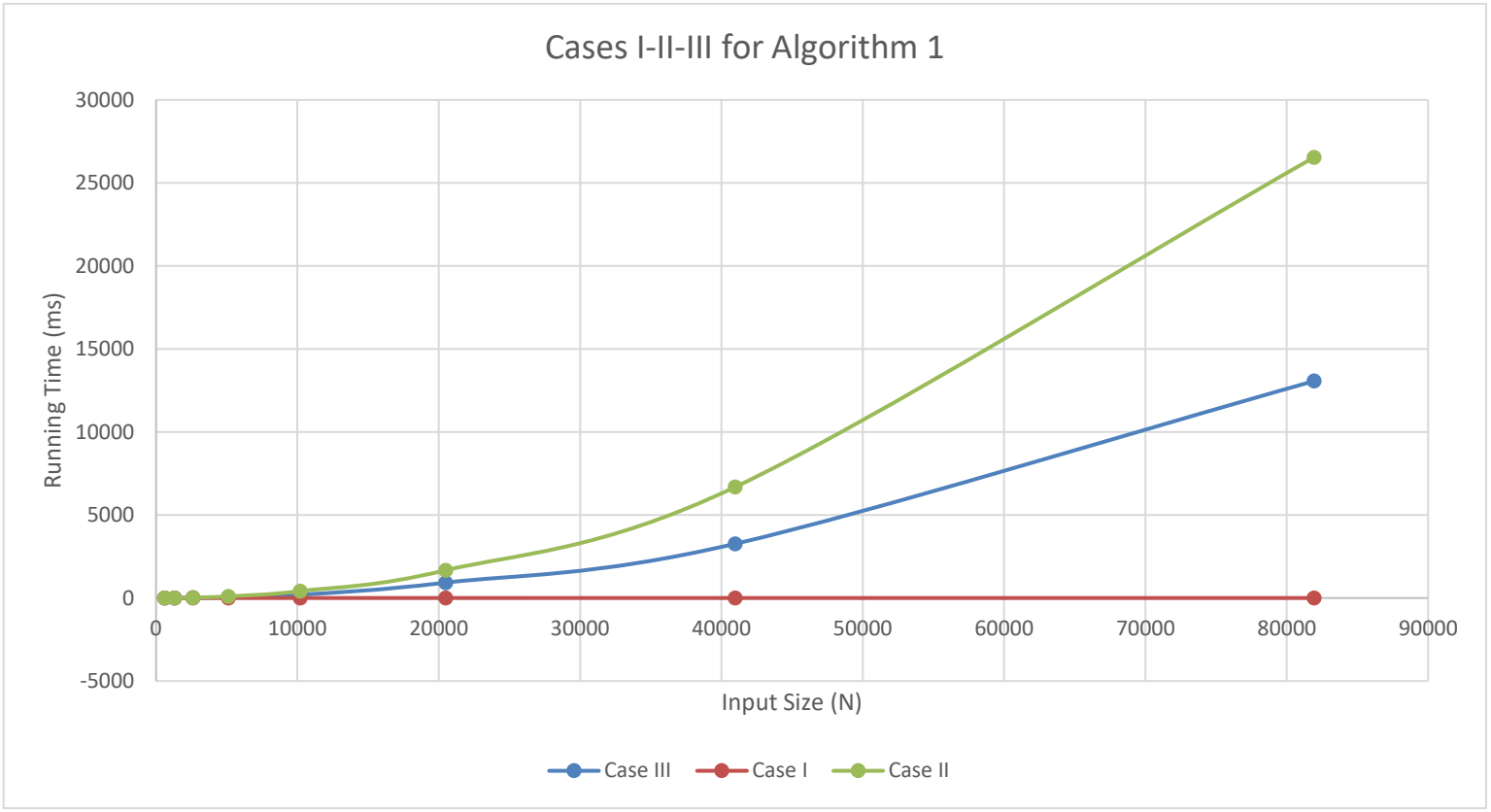
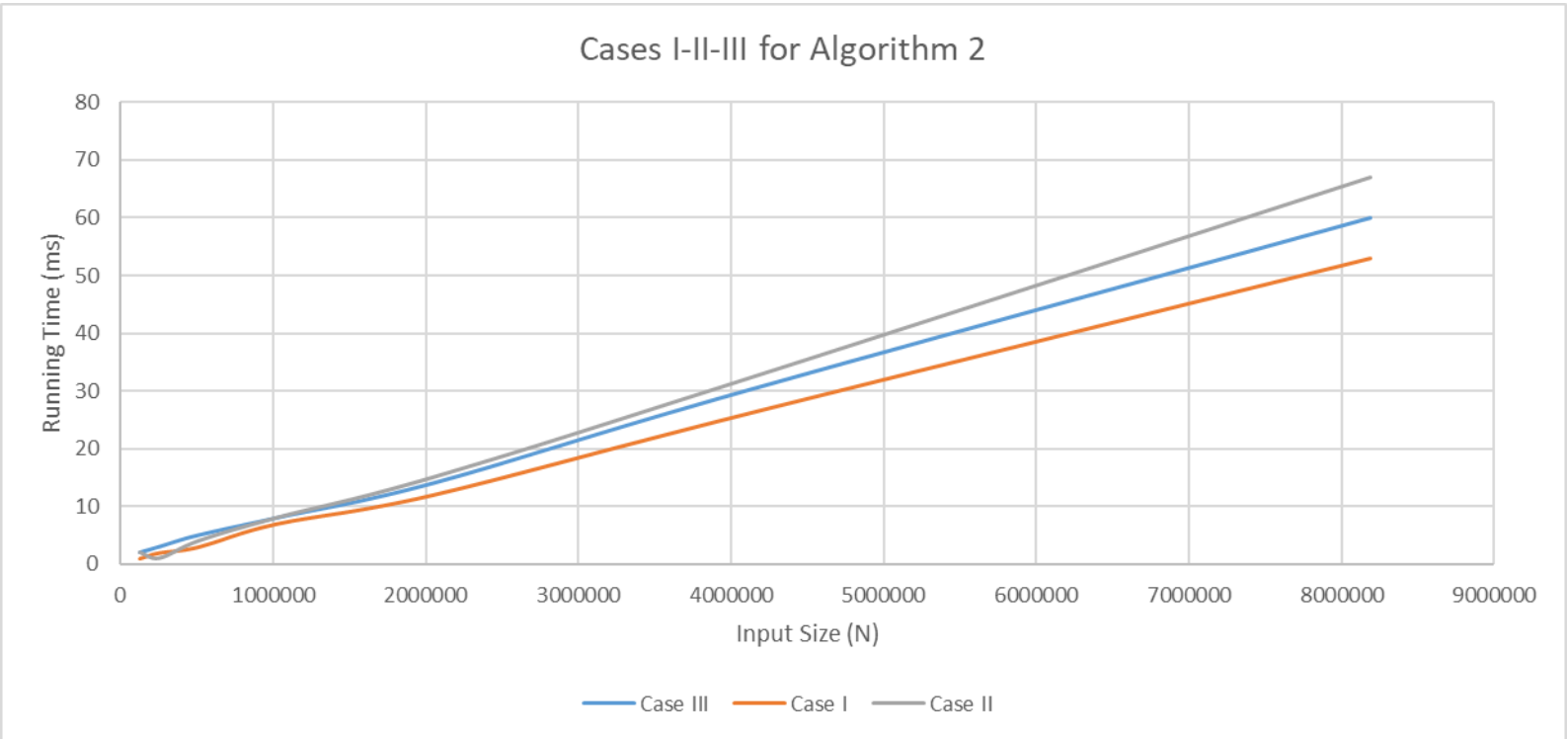
Efe Beydoğan, 21901548, CS201 Section: 1

Plots of running time versus the input size for cases I-II-III:





Best, average and worst cases for each algorithm and the corresponding worst case time complexity:

For Algorithm 1:

Best Case: Case I

Average Case: Case III

Worst Case: Case II

Corresponding worst case time complexity: $O(N^2)$

For Algorithm 2:

Best Case: Case I

Average Case: Case III

Worst Case: Case II

Corresponding worst case time complexity: O(N)

Specifications of the computer I used to obtain these execution times:

CPU: Intel Core i7 4790K

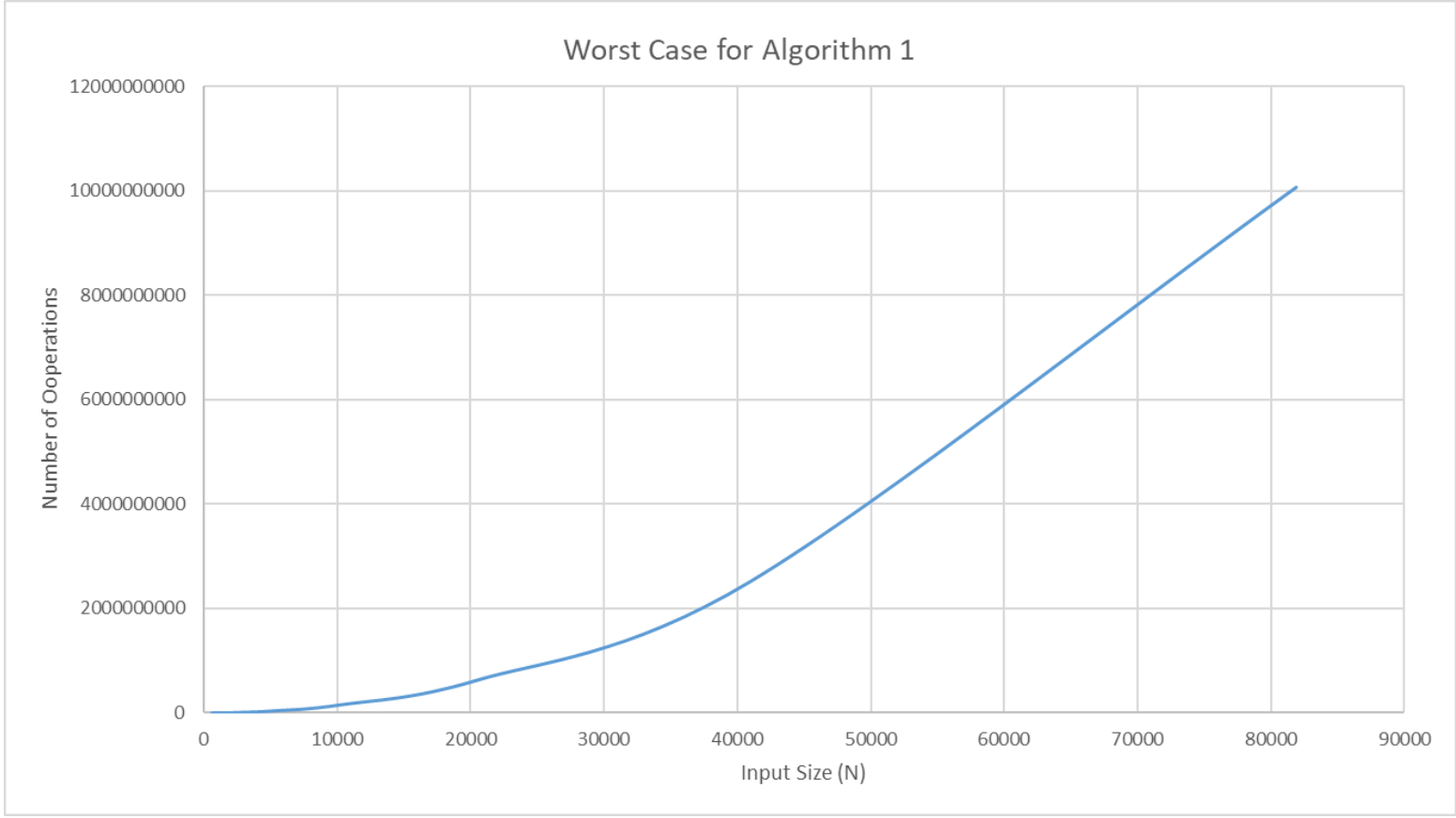
Graphics Card: ASUS GTX970

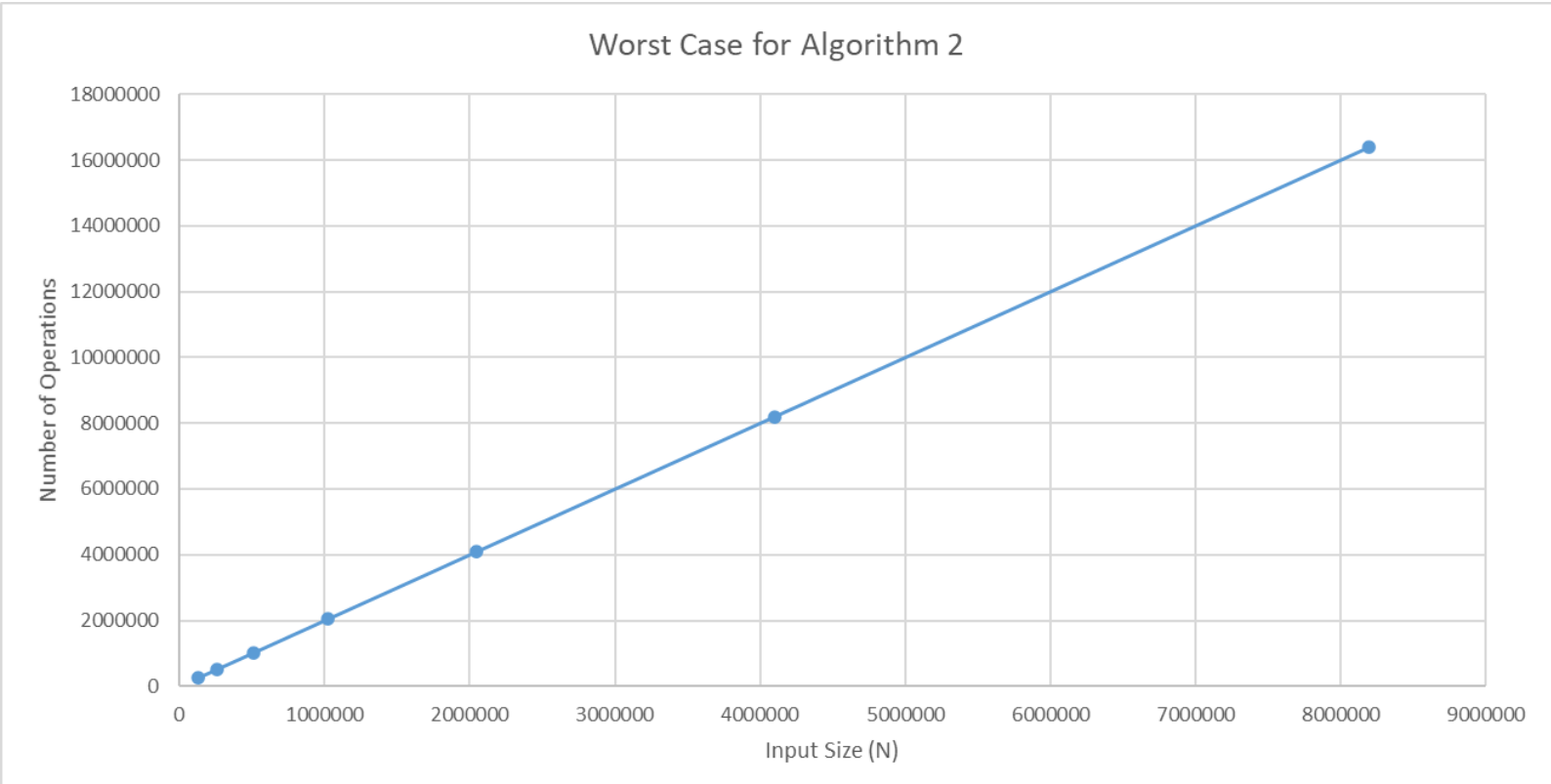
Motherboard: ASUS Maximus VII Hero

RAM: GSKILL 16GB RipjawsX

OS: Windows 10

Plot of the expected worst case growth rates obtained from the theoretical analysis by using the same N values that were used in the simulations:





Comparison of the expected growth rates obtained in step 5 and the worst case results obtained in step 3:

In Step 5, I plotted the expected worst case growth rates by analysing the algorithms to find the approximate number of operations it would take the functions to sort the arrays with different sizes of N. I observed from those graphs that the worst case for Algorithm 1 has an $O(N^2)$ complexity while the worst case for Algorithm 2 has an O(N) complexity. The graphs I plotted in Step 3 using the execution times for these algorithms are consistent with the theoretical worst case growth rates in Step 5. The graph for Algorithm 1 in Step 3 also shows that the worst case has an $O(N^2)$ time complexity and the worst case for Algorithm 2 has O(N) time complexity. Algorithm 1 takes a longer time to sort the arrays in its worst case because after appending all the items in arr1 in the same order to arr3, since in the worst case every element in arr1 is bigger than the elements in arr2, the algorithm has to shift every element in arr3 to the right for every element in arr2, hence the complexity of $O(N^2)$. Algorithm 2 however, adds elements to arr3 after comparing pairs of elements in arr1 and arr2, and thus doesn't have to do any shifting, it only iterates through the two arrays and appends elements to the end of arr3, so has linear time complexity.