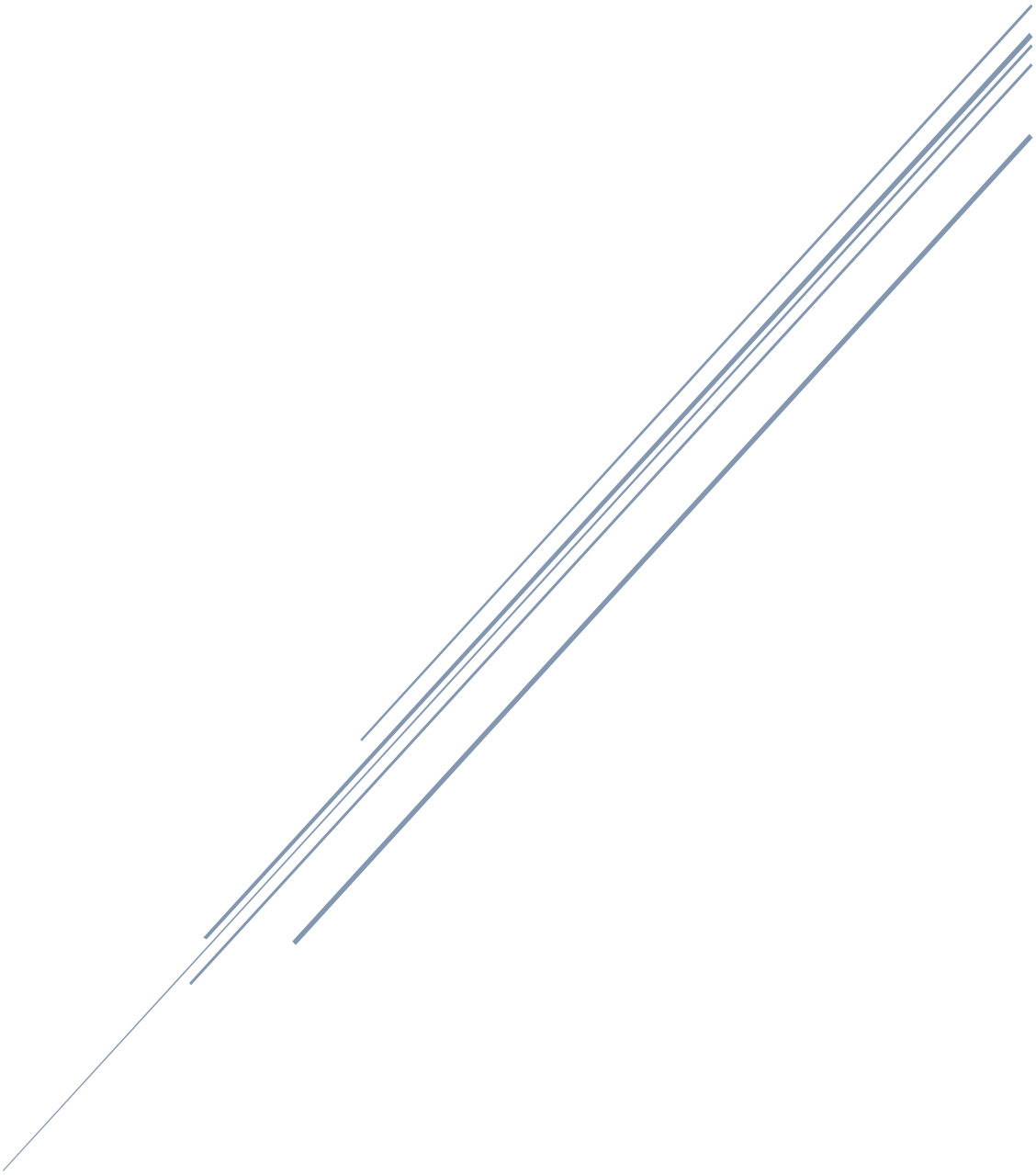
PROGRAMMING ASSIGNMENT 1

21604552 - Emre Hancı



Hacettepe University - Department of Computer Engineering

BBM204 Software Lab.

-Problem definition

In this assignment our advisors expect that to teach us “Which sorting algorithm is more efficient, and which sorting algorithm is more useful than others in some case, and binary search algorithms time complexity.” Binary search is the only search algorithm we examined and those are the sorting algorithms we use for making comparison;

-Insertion Sort:

That algorithm separate an array into two regions: sorted and unsorted, then takes each item from the unsorted region and insert it into its correct order in the sorted region after that finds the next unsorted element and insert it in the correct place. That algorithm is more efficient with using small arrays.

-Selection Sort:

That algorithm firstly finds the smallest element in the unsorted section, after finding the smallest element swap that element with the first element in the unsorted section. That algorithm is more efficient with small arrays.

-Radix Sort:

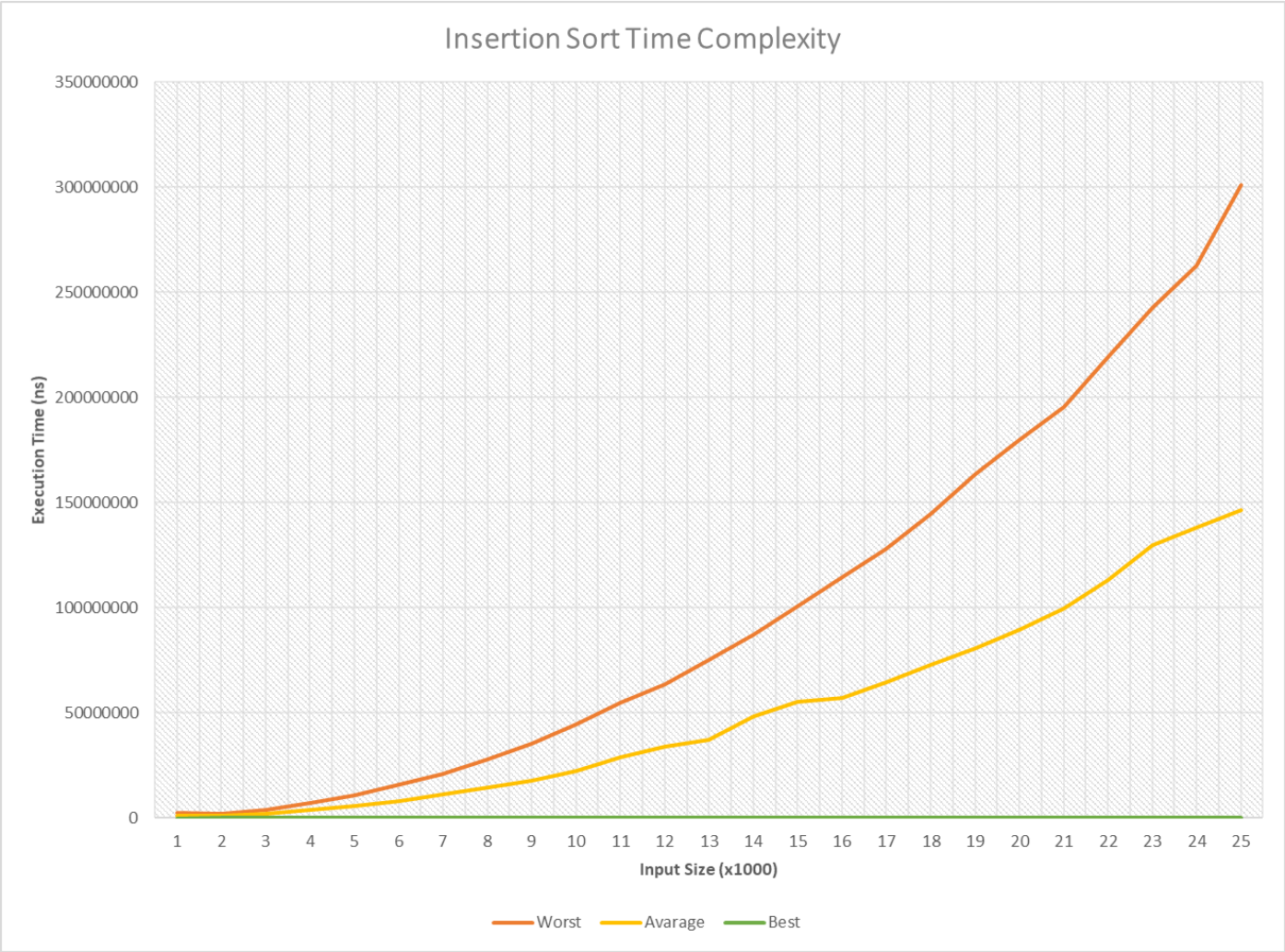
That algorithm performs a sorting according to digits, firstly make an order according to first digit, than second digit until all the digits ordered. Depending on how many digits the numbers are this sort can be more efficient than any O (n log n) sort algorithms, depending on the range of values.

-Merge Sort:

That algorithm runs with divide and conquer strategy. An array divided into sub-arrays until all the sub arrays having two elements than sort those two elements after sorting turn back and sort all those sub-arrays. That algorithm has the best time complexity of sorting algorithms we discuss in this assignment.

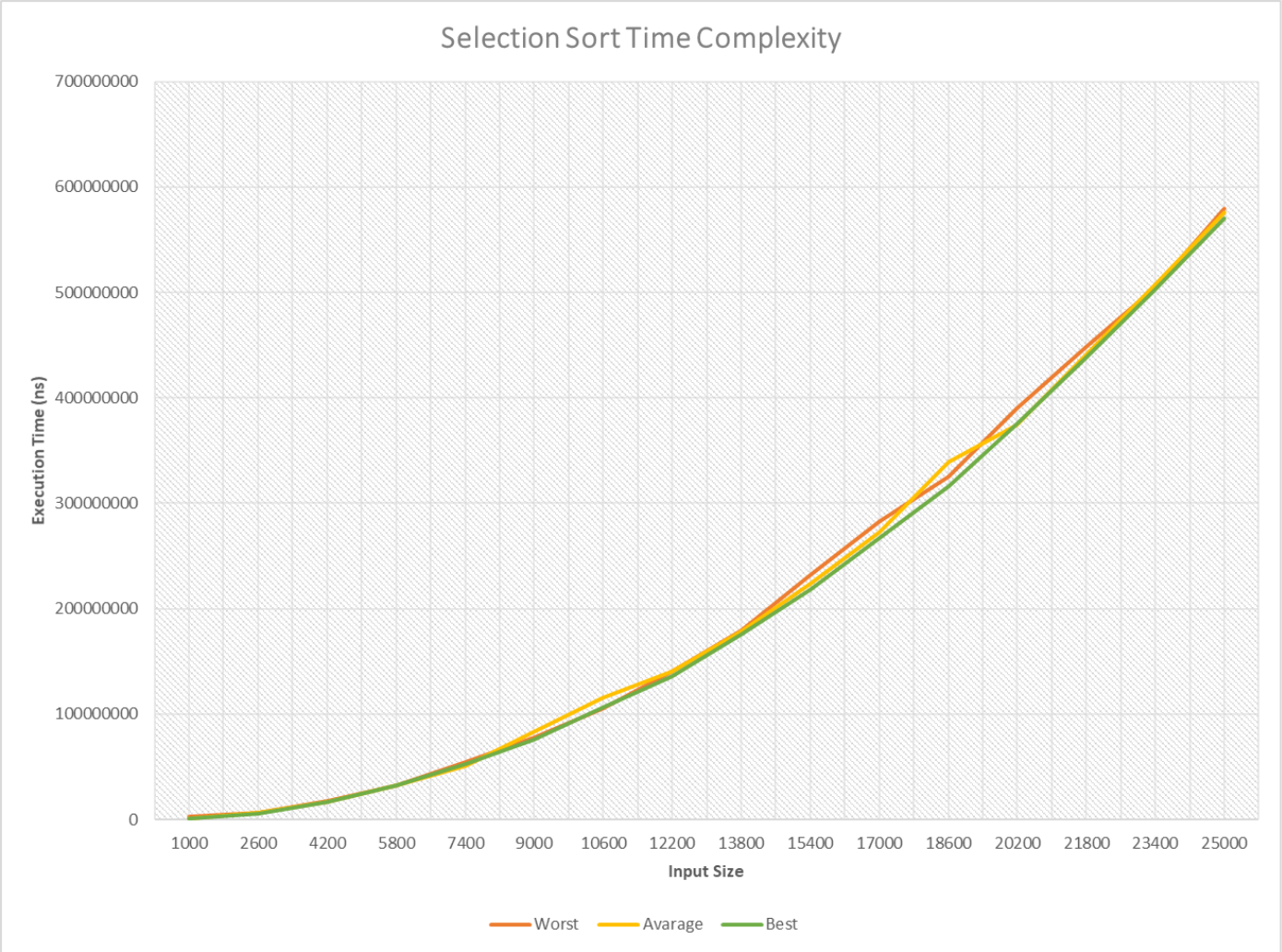
-Findings & Discussion

+Insertion Sort Algorithm



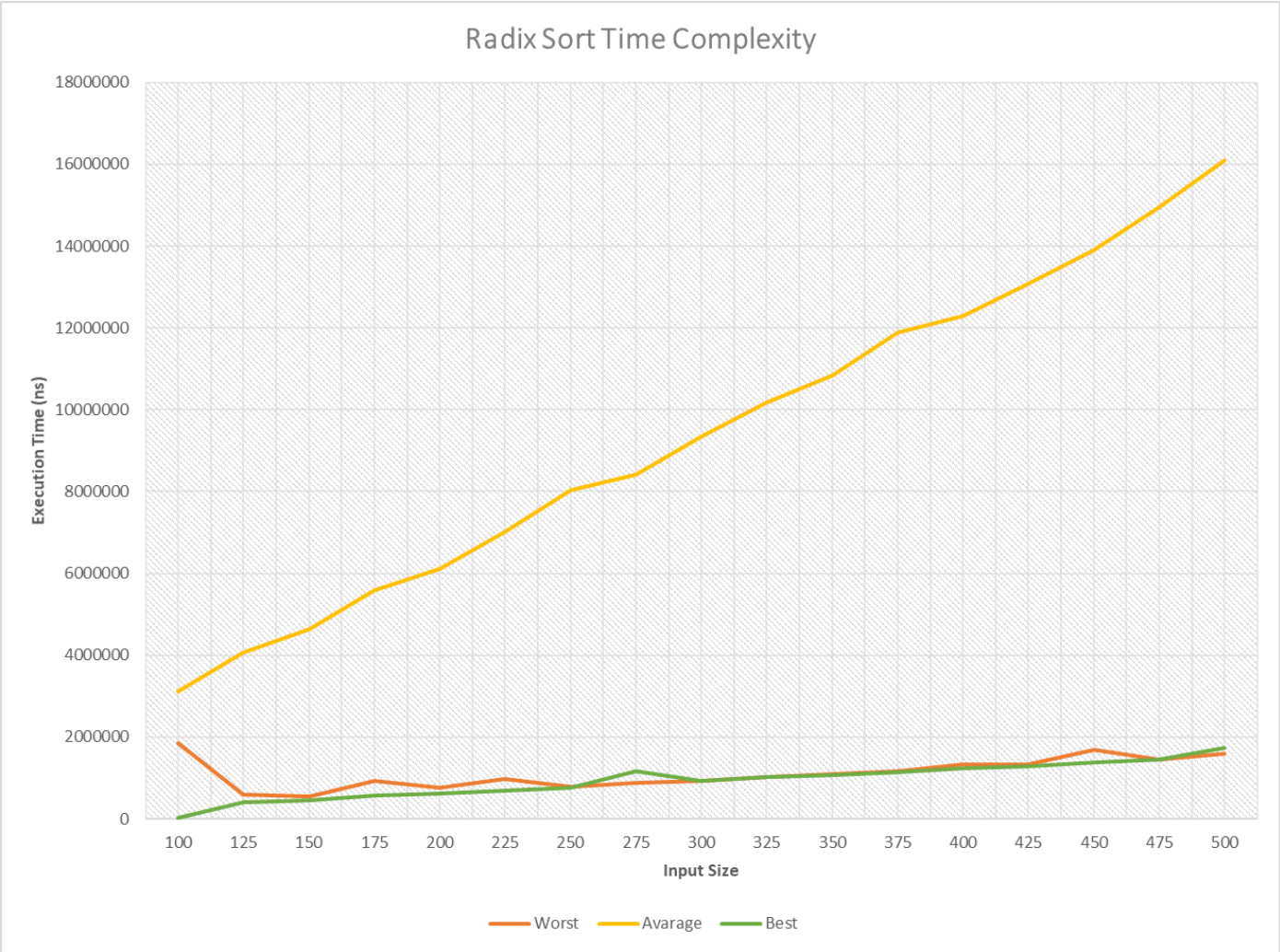
For insertion sorts best case I generate numbers in ascending, for average case I generate random numbers shuffle, the worst case of this algorithm is descending order, therefore in worst case I generate numbers in descending order. My interests in this plot is input size is not important for this algorithm, the main problem in this algorithm is shuffle degree of inputs.

+Selection Sort Algorithm



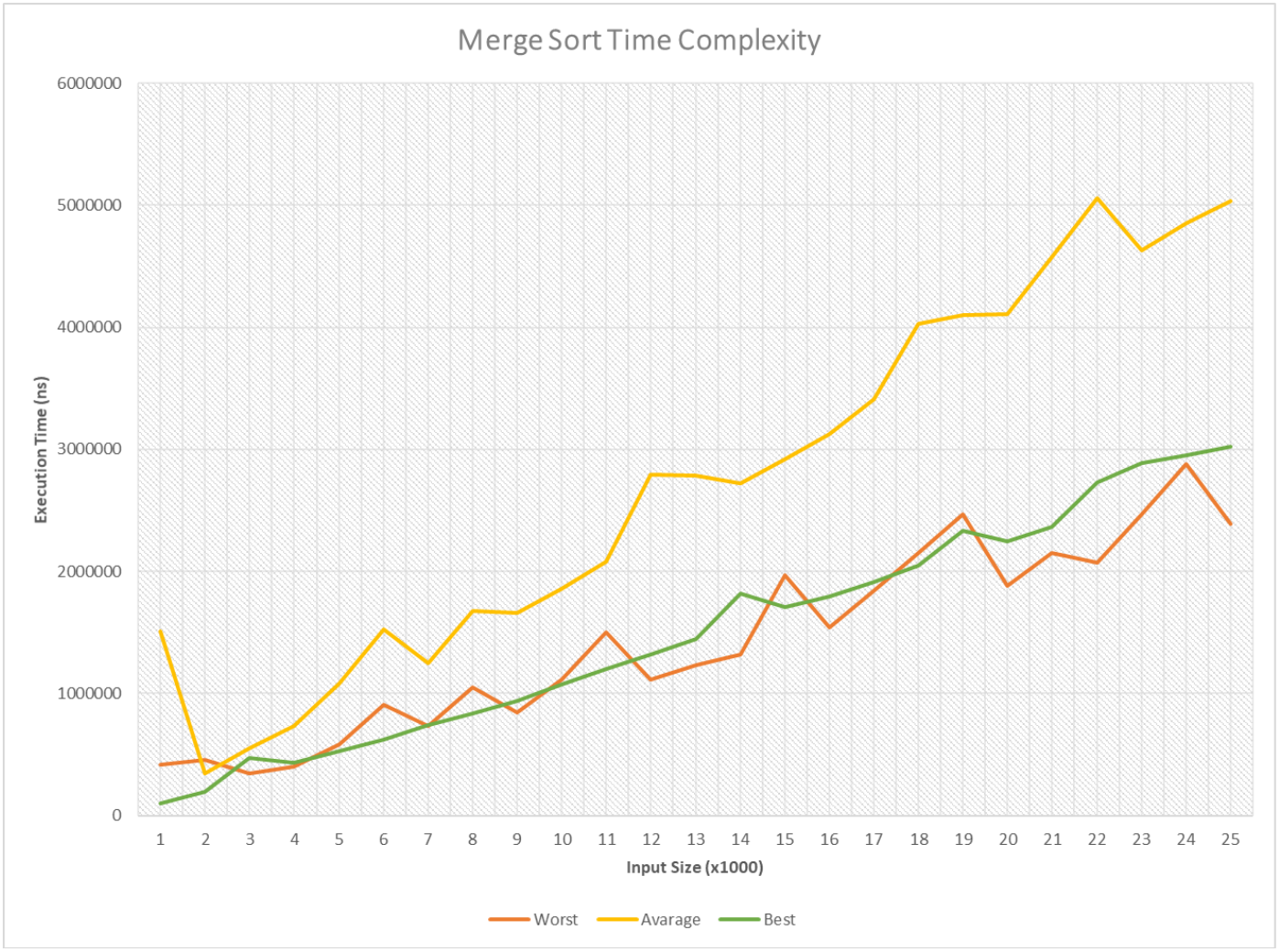
For selection sorts best case I generate numbers in ascending, for average case I generate random numbers shuffle, the worst case of this algorithm is descending order, therefore in worst case I generate numbers in descending order. My interests in this plot is shuffling degree is not important for this algorithm, the main problem of this algorithm is input size, when input size is increasing time complexity is getting worst.

+Radix Sort Algorithm



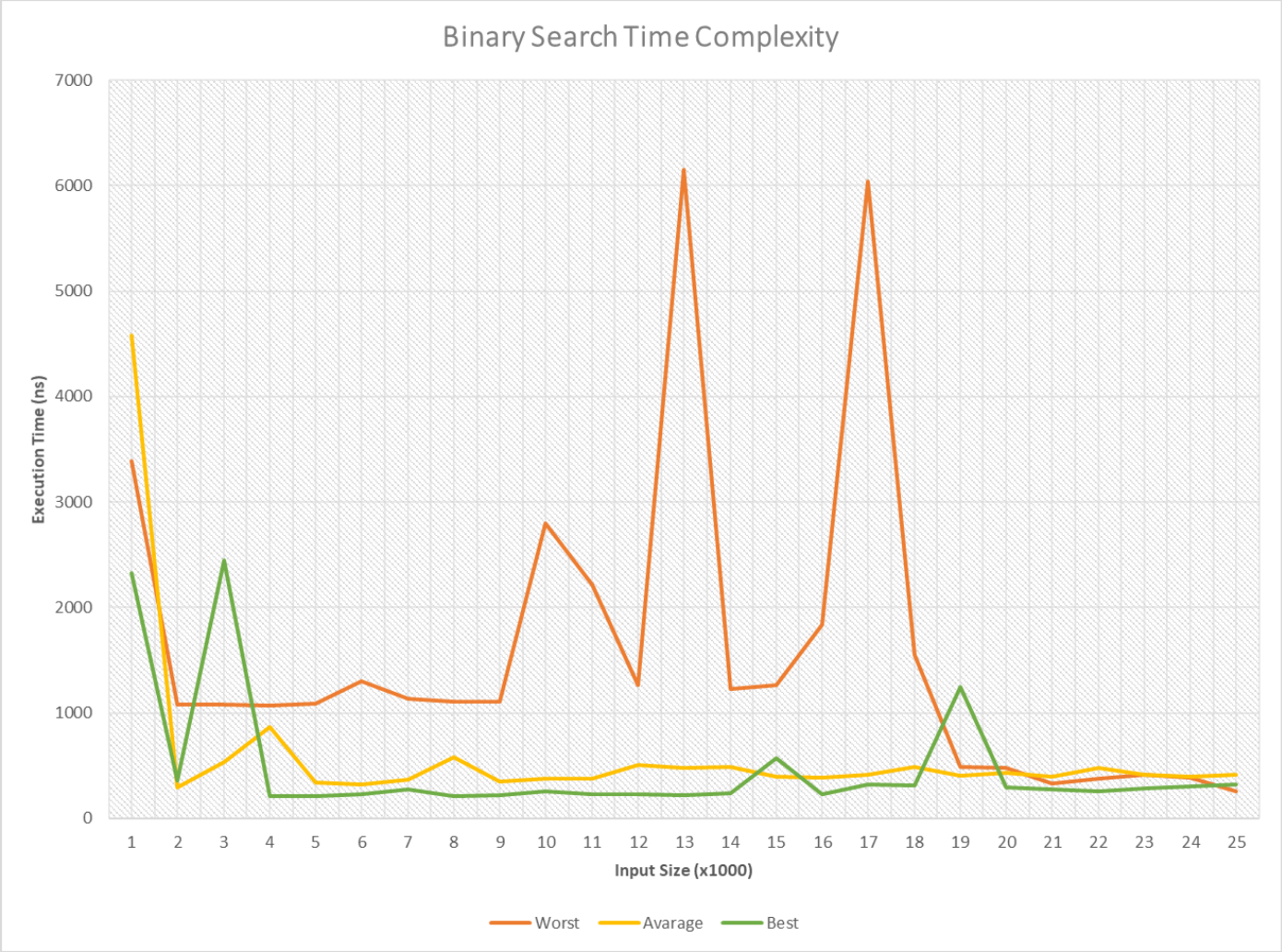
For radix sorts actually I am confused for best, worst and average case, firstly I thought that the best case must be one digit number array but if that is the best case than what is the worst case? Infinity digit numbers array? After discussed with myself, my final idea is best case I generate numbers in ascending, for average case I generate random numbers shuffle, the worst case of this algorithm is descending order, therefore in worst case I generate numbers in descending order. My interests in this plot is if numbers are in any order radix sort perform more efficient than shuffle array.

+Merge Sort Algorithm



For merge sorts best case I generate numbers in ascending, for average case I generate random numbers shuffle, the worst case of this algorithm is descending order, therefore in worst case I generate numbers in descending order. My interests in this plot is merge sort provides same time complexity for every kind of case also that merge sort is the most efficient sort algorithm in all the sort algorithm we examination in this assignment.

+Binary Search Algorithm



For binary search all case I generate numbers in ascending order, but for worst case I demand the number which is not in the array, for average case I demand random number, for the best case I demand the number which is full middle of the array. My interests in this plot is time complexity of this algorithm is getting worse, repeatedly calling function, the best case is getting the number O (1). That plots is not looks like O (n log n) because of making this plot less information for such a fast algorithm, if I had much more heap, and much more strong computer then I could show that this algorithms time-complexity is O (n log n).