HACETTEPE UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

BBM204

PROGRAMMING ASSIGNMENT #1

Subject : Analysis of Algorithms

Submission Date : 11.03.2019

Deadline : 25.03.2019

Programming Language : Java

Student ID: 21727432

Student Name: Ali Kayadibi

INTRODUCTION / AIM:

In this experiment we are expected to gain knowledge on sorting algortihms, execution times, input dependance sorting methods,time efficiency,complexity of the algorithms.

PROBLEM DEFINITION

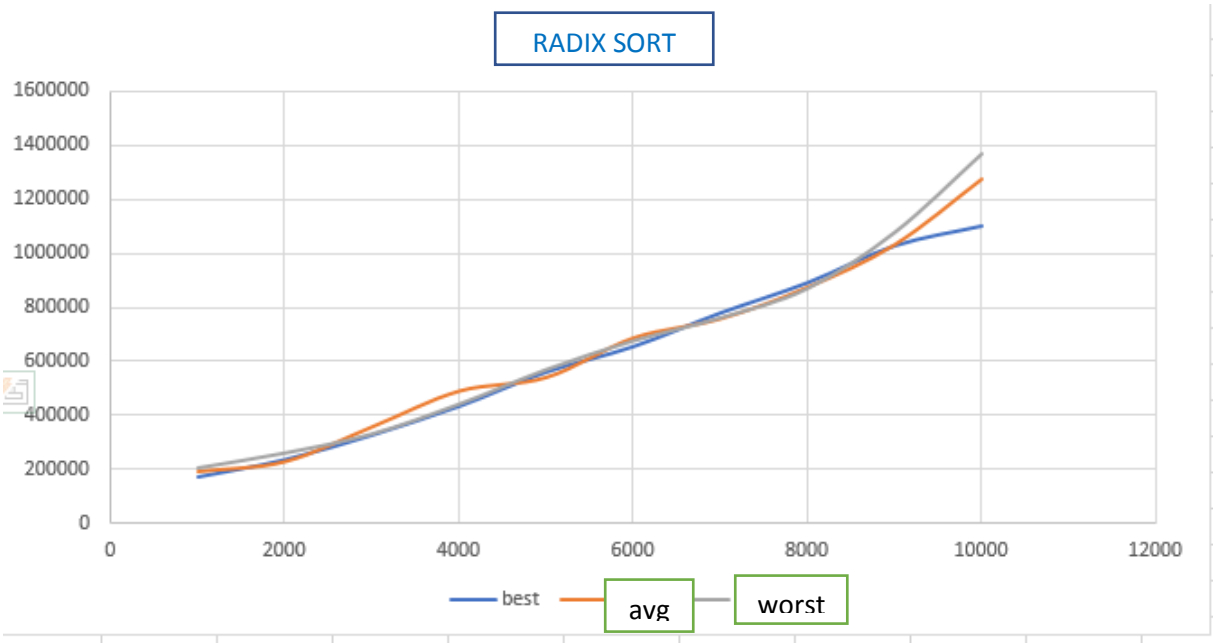
In this assignment, we are expected to implement different sorting operations on different input sized data structures to get their execution time.We will try our arrays with different inputs(best input worst input and average input),different input sizes and we will get the average of their execution time to see if a sorting algorithm is input dependent(best,worst,average) or not.Reason of taking average is there might be time spikes in our execution time so this is why we are taking averages so that we can see the execution results clearly.I have already drown some graphs but execution time always changes during executions.User implementation is important but cpu speed is also important.We will see the method's best,worst,average running times in the graphs.

SORTING METHODS & EXECUTION TIMES

1.Radix Sort:

Basically radix sort sorts all elements of the array starting from the least significant bit to most significant bit.At first, it

finds the maxium value of the array because it is our highest number with the most number of digits.After than for each bit we sort the array starting from the least significant bit which is the last bit.If a number has less digit than our maximum value we assume that it has 0 in front.So we sort(1,10,100…) than all we left is sorted array.In radix sort average complexity is O(digit number x n).In our three case (best,worst and average) our complexity depends on only with the biggest number of digits so they are all same.Our best case is array is already sorted, worst is reverse sorted, average is random values but in those 3 it still depend on the biggest number so we say radix sort has O(n.k) complexity for all 3 cases.

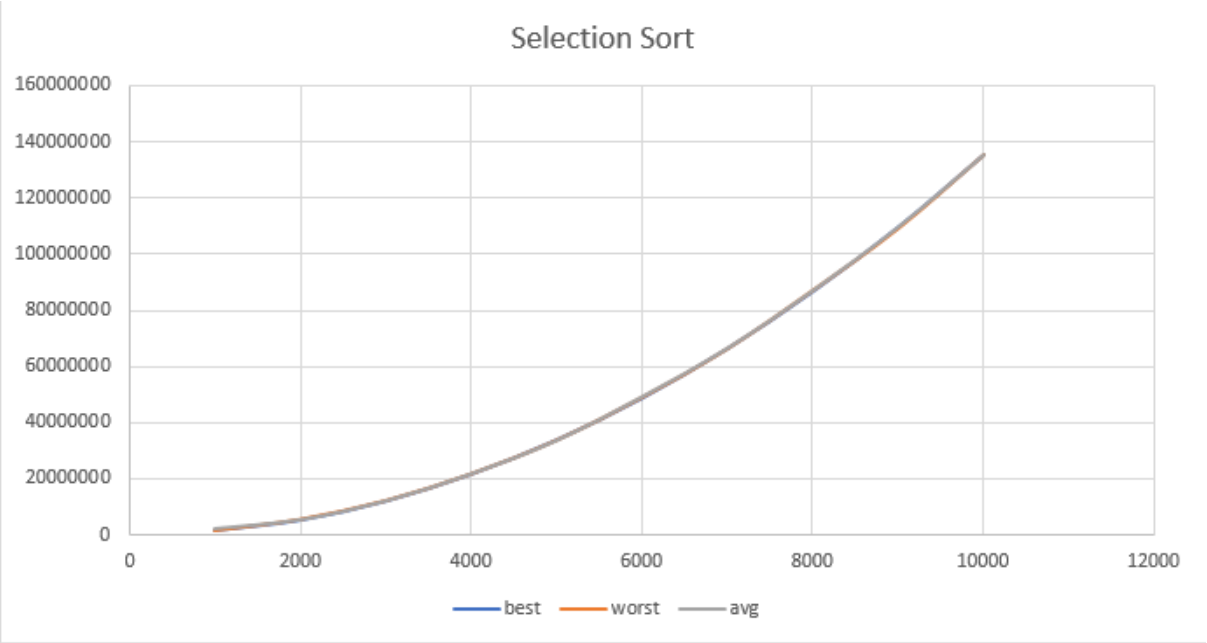


In our three case(best worst average) we had one biggest number with biggest digit so their complexities are nearly same.As you see from the picture, as the input grows all three cases are nearly same(cpu speed is the reason of the difference)

2.Selection Sort:

Selection sort goes through all elements of the array,finds the minimum value and swaps it with the our index which starts from 0 and goes to n.If minimum is not in his place we just trade it with our index and search the next minimum value until we reach end of the array.As you can see we always go through the elements of the array to find the minimum value.And also we go through with our index so for every index we find the minimum value in the array.So our complexity is O(n^2) for all cases.Our best case is array is already sorted but we must go through all elements,go through again to find the minimum.In our worst case array is reverse sorted but again we go through everthing in the array and find minimum for that every index so worst case and best case are both equal.Our average case is random values but again we go through all to find min and index.As you can see they are always equal(there might be small differences

because of the swap operation)

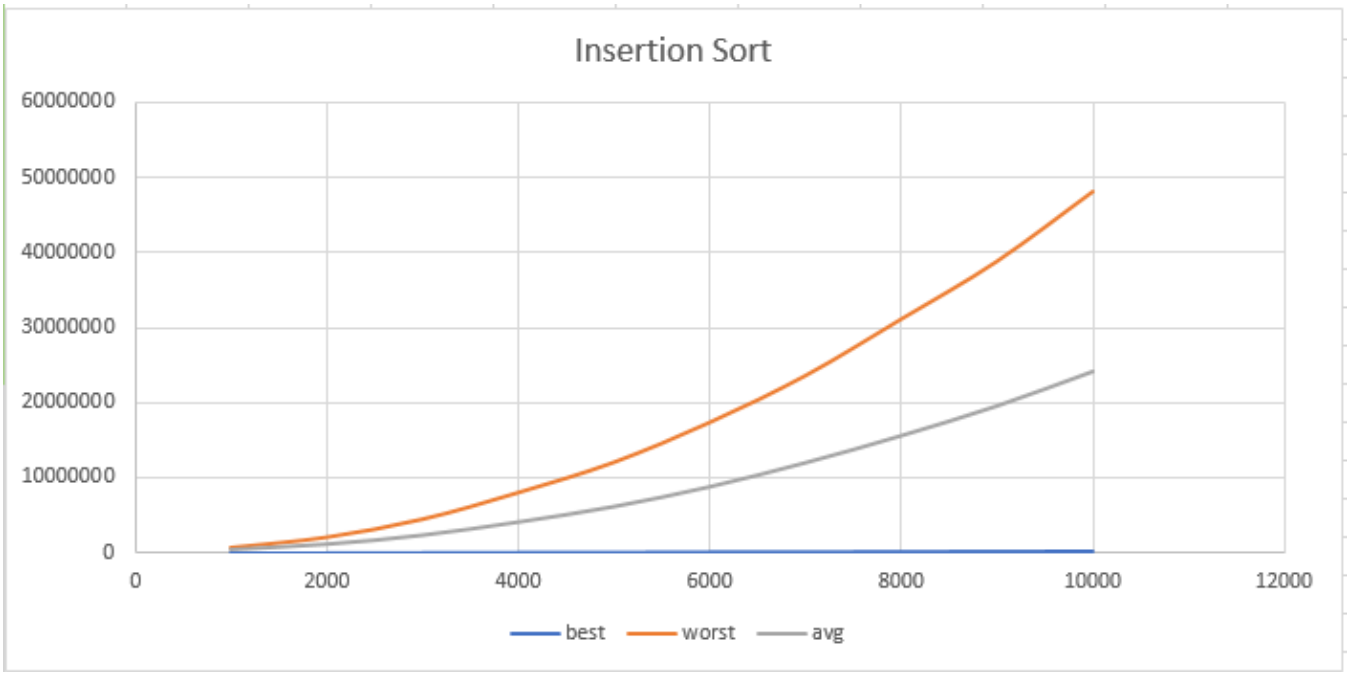


They are nearly same this is why we say selection sort is not input dependent.It has O(n^2) for all cases.

3.Insertion Sort:

Insertion sort goes through all elements of the array and in every step is compares the value with the value on the left.If our value is smaller than its left so we trade the values.It goes through until it finds a value which is bigger than itself or if we reach the 0 element.So it is comparing and finding the index of that value.Our best case is array is already sorted so we look at each element every element is bigger than its left so we dont do any swap operation we just go through all elements which is O(n).In our worst case array is reverse sorted so every value is lesser than its left value so we swap the value until the first element.So this is O(n^2) because for every index we go through from k to 0 index.In

our average case we might or might not have lesser value so in average complexity is O(n^2).



This is the graph of insertion sort.For the average case we swap many elements from k to l.(their correct index) but for the worst case we swap all the way to 0 so this picture really helps us understand the insertion sort.

4.Merge Sort:

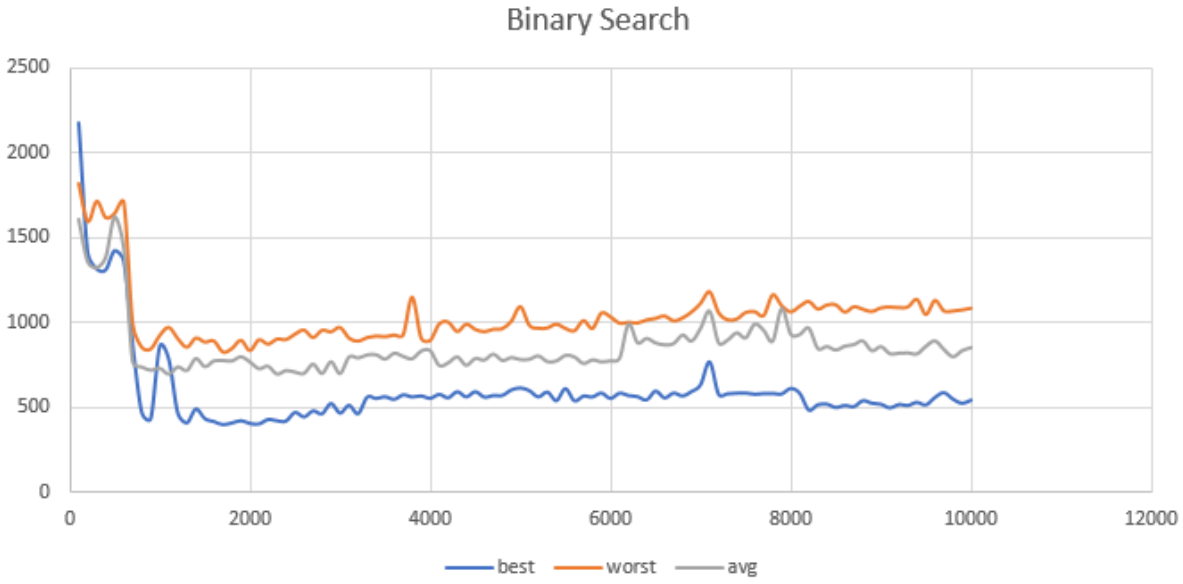
Merge sort is an divide and conquer algorithm which divides the original array recursively into smaller arrays with element of 1 or 2.And with the merge function merges these little arrays into bigger arrays.At the end we get our sorted array.In our best case elements are already sorted but merge sort divides these elements into smaller arrays and merges them so it is not an input dependent sort.If whe had reverse sorted array which is our worst case we still divide them into little arrays and merge them into bigger arrays so our inputs

does not determine the complexity of our code.It is always O(nlogn).Our average case is random numbers but we still divide the array into smaller arrays so complexity is the same.

5.Binary Search

To use binary search we must use sorted arrays even if it is worst case or best case.This time our best case is element is in the middle of the array, worst case is there is no such element in array and average is element is somewhere in the array.So in our best case it is O(1), in our worst case we always limit our array and search for a value so it is O(logn) and our average is O(logn) but since there is a value exist it is

better than worst case.



Since log n is very small and our cpu's are very fast it is looking like it is O(1) for worst and average case and log n is really small number.If we had much more bigger arrays we would see better.