MARMARA UNIVERSITY FACULTY OF ENGINEERING

COMPUTER ENGINEERING

Project:

NAME:	**STUDENT NUMBER:**
Barış Hazar	150118019

Submitted To : Ömer Korçak

Submission Date : 16/05/2021

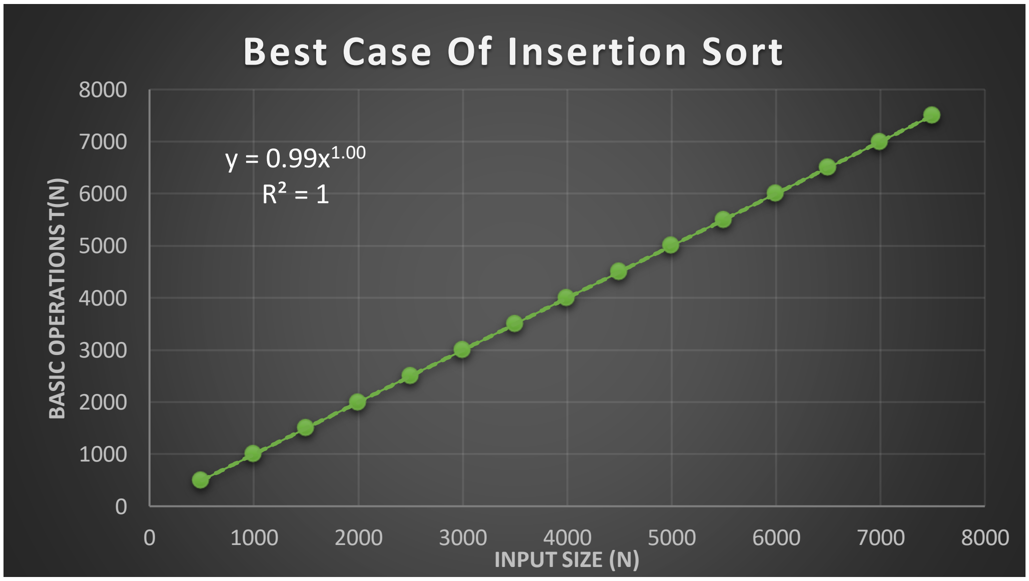
Problem Definition:

In this homework, we compared seven different sorting algorithms by measuring basic operation count for best, worst, and average cases of them. And showed their strengths and weaknesses.

1-) Insertion Sort

Note: The basic operation is selected as number of key comparisons.

- Best Case:

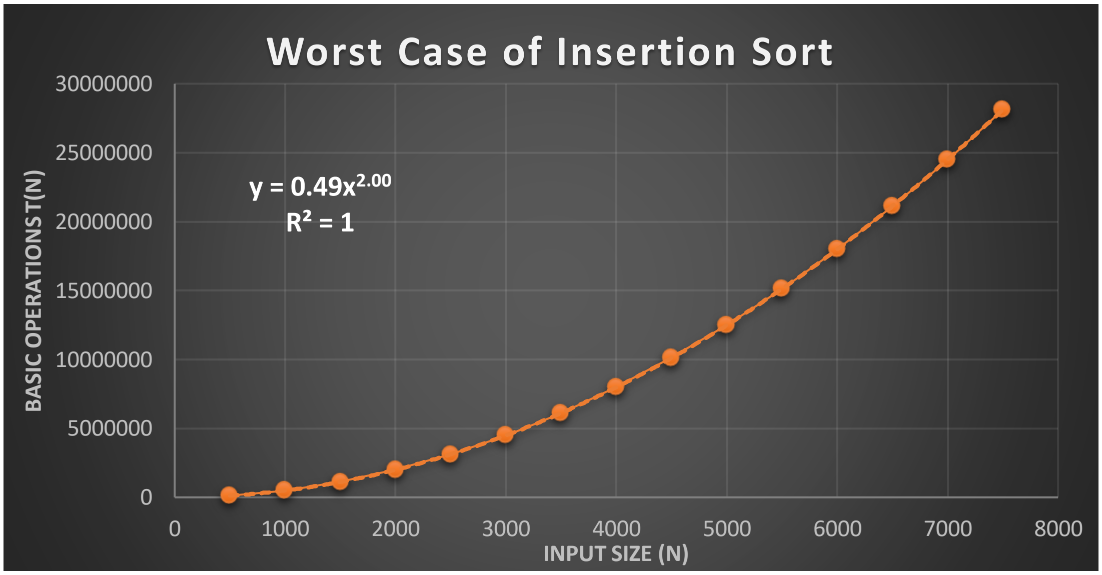


Best case of Insertion Sort occurs when the list is already sorted. It compares all the elements once, without entering the inner while loop.

Since the inner loop is not executed at each iteration of the outer loop, its theoretic time complexity is **Θ(n)**. And as we can see from the trendline equation **$y = 0.99x^{1.00}$**

above, we can clearly say our empirical results are the same as our expectations, that is **Θ(n)**.

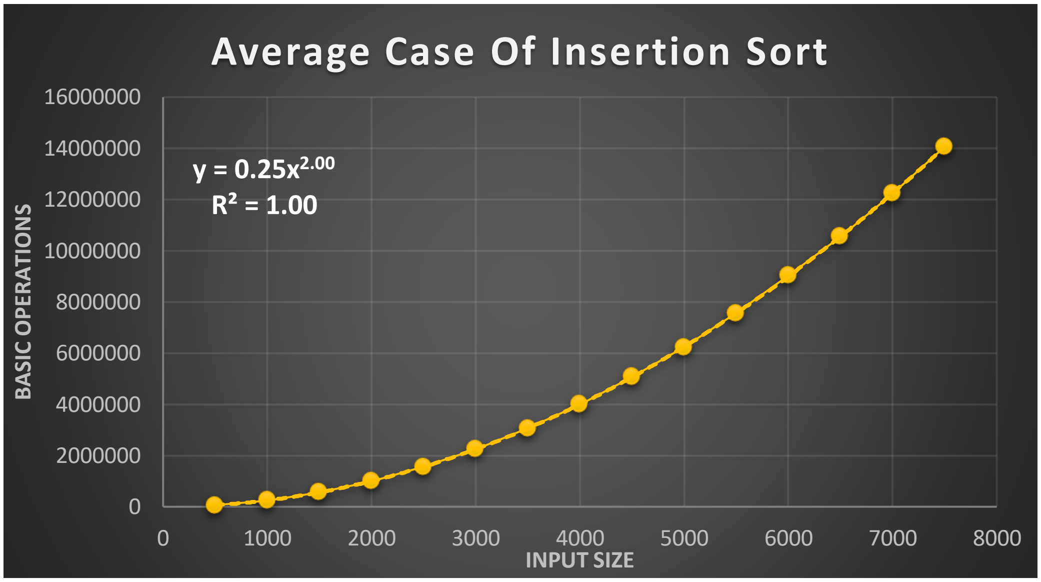
- Worst Case:



Worst case of Insertion Sort occurs when we have a reversely sorted list. Inside the inner loop it goes all the way to the beginning at every iteration.

Since it is a brute force algorithm and there are two nested loops that are both **Θ(n)**, we can clearly say that it is a **Θ(n^2)** algorithm theoretically. And if we look at the chart above, it is seen that the trendline equation for basic operations is **$y = 0.49x^{2.00}$**. From which, one can clearly see that it is a **Θ(n^2)** algorithm that is the same as the theoretical result.

- Average Case:



Average case for Insertion Sort is when we have a random list. The condition for the inner loop is totally random.

Theoretically, we assume the inner loop will not always be executed all the way to the beginning as the worst case, or it will never be executed as the best case. But it will be executed directly proportional to the input size. Thus, its theoretical time complexity is **$\Theta(n^2)$** in the average case.

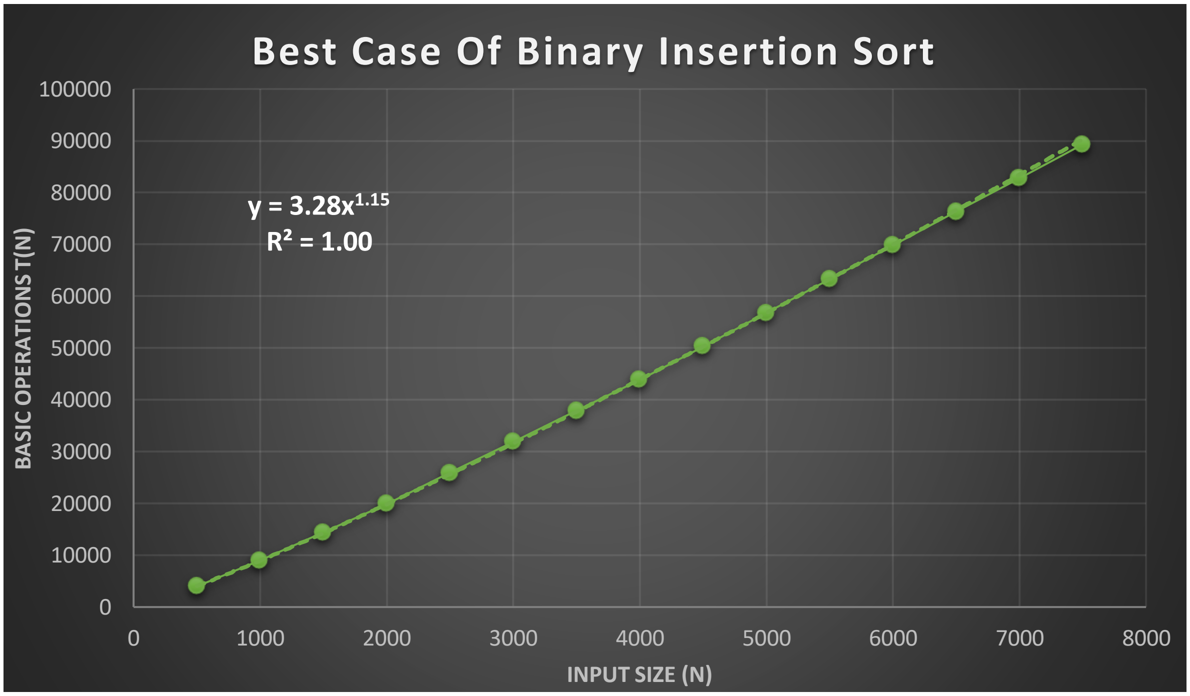
As it is seen from the chart above, the trendline equation for basic operations is **$y = 0.25x^{2.00}$**, which is a **$\Theta(n^2)$** algorithm and meets our theoretical result.

Also, **$0.25x^{2.00}$** is less than **$0.49x^{2.00}$,** we can see average case is the same asymptotically with the worst case. But it is better exactly than the worst case since their coefficients are different.

2-) Binary Insertion Sort

Note: The basic operations are selected as every key comparison and shift operation.

- Best Case:

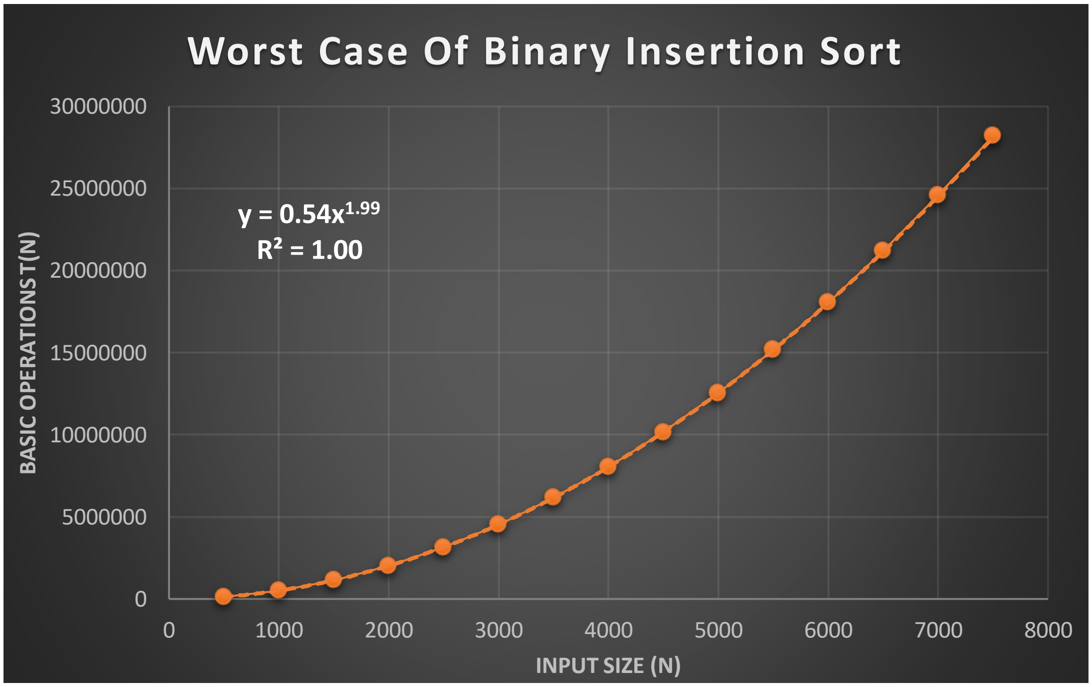


Best case of binary insertion sort occurs when we have an already sorted list. It finds a position for the key by applying binary search at each iteration for the outer loop without entering the inner loop.

Its theoretical time complexity is **Θ(n*log(n))** because we apply binary search which is **Θ(log(n))** at every iteration of the outer for loop, which is **Θ(n).** Thus, **Θ(n) * Θ(n*(log(n))) = Θ(n*log(n)).**

Empirically, our trendline equation is **$y = 3.28x^{1.15}$**. We see that in the equation x is raised to the power of 1.15. Since 1.15 is a number between the interval (1,2), we can directly say it is a **Θ(n*log(n))** algorithm.

- Worst Case:

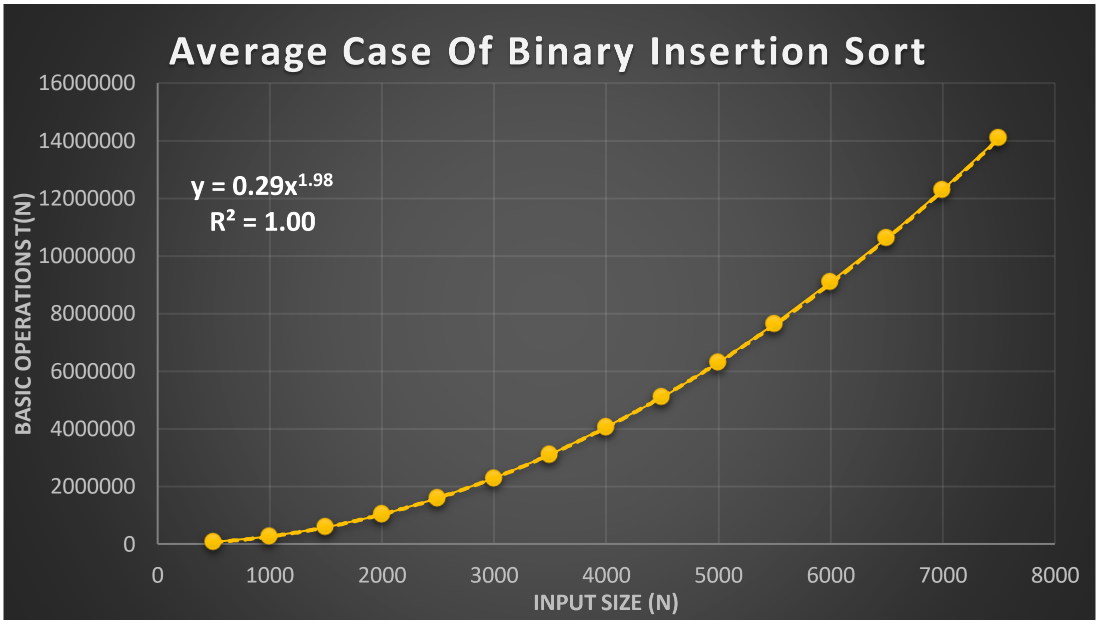


Worst case of Binary Insertion Sort occurs when we use a reversely sorted array. It shifts the elements all the way to the beginning and maximizes the count of basic operations.

Theoretically, it is the same as the worst case of insertion sort since it shifts all the elements to put the next key in sorted subarray at every iteration of the outer loop. Its equation is, **Θ(n) * (Θ(log(n)) + Θ(n))** we can ignore **Θ(log(n))** in the equation and it becomes **Θ(n^2)** theoretically.

Empirically, our trendline equation is **$y = 0.54x^{1.99}$**. We see that in the equation x is raised to the power of 2. Thus, we can say it is a **Θ(n^2)** algorithm both empirically and theoretically.

- Average Case:



Average case of Binary Insertion Sort occurs when the list is random. The number of basic operations will be somewhere between best case and worst case.

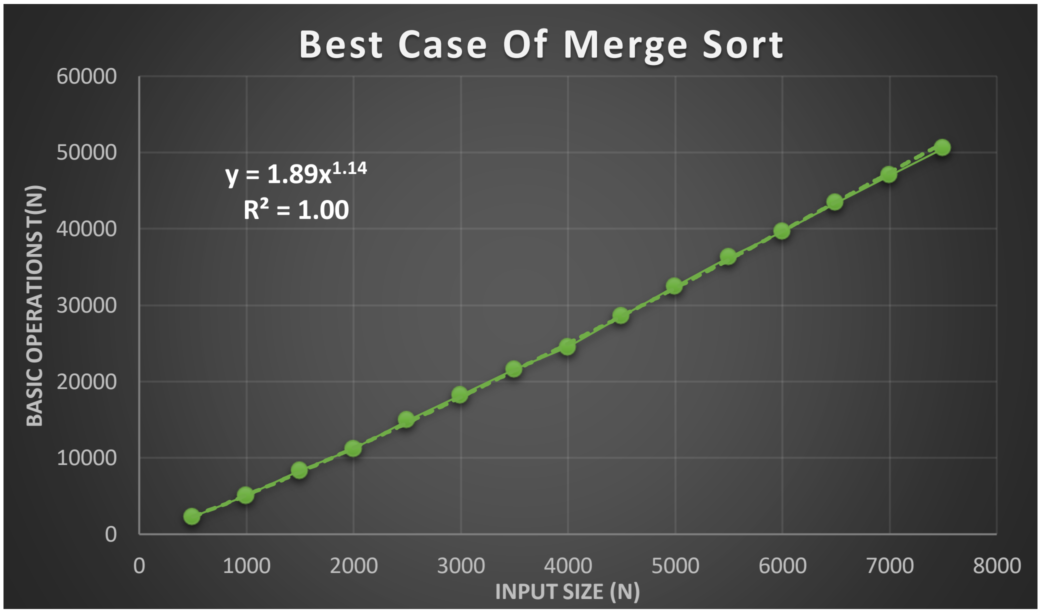
Theoretically, since this is a brute force algorithm, the expression is **Θ(n) * (Θ(log(n)) + Θ(n))** we can ignore **Θ(log(n))** in the equation and it becomes **Θ(n^2)** just like the worst case.

Empirically, our finding is **y = 0.29$x^{1.98}$**, which is **Θ(n^2)** as expected, and meets the theoretical result, and since its coefficient is less than the worst case, it is better than the worst case exactly, but not asymptotically.

3-) Merge Sort

Note: The basic operation is selected as key comparisons, without including other operations to show the difference between best, worst, and average cases.

- Best Case:

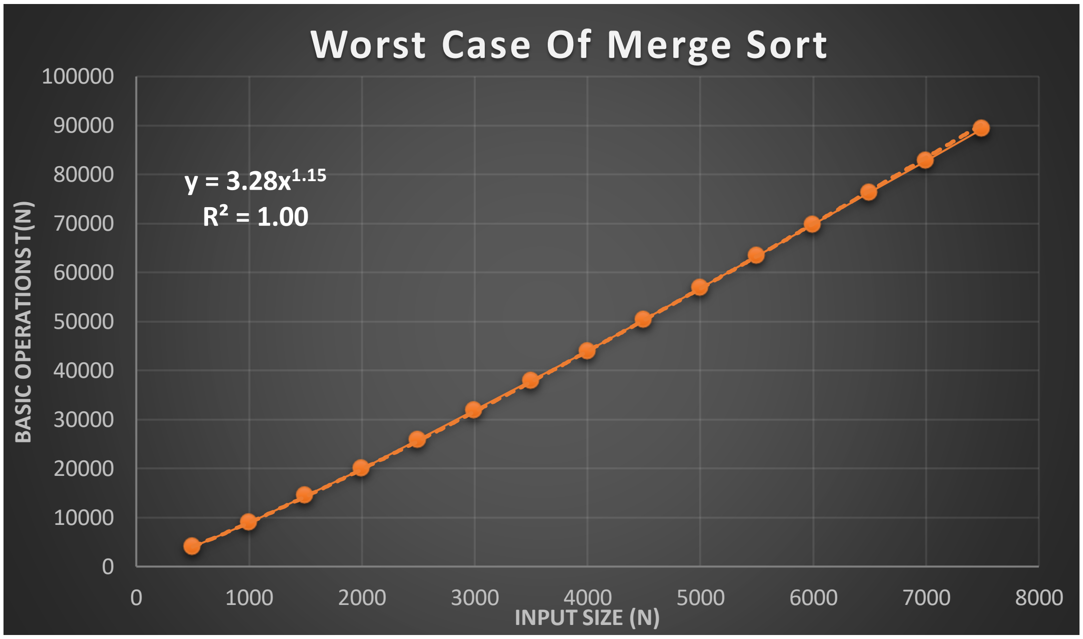


Best case of Merge Sort occurs when the sample is already sorted. It makes the least number of comparisons when merging left and right together, since only left part will be merged by comparing two elements. The right part will be added to the merged list without any comparison.

Theoretically, it is a divide and conquer algorithm with the recurrence relation **T_n = 2T(n/2) + Θ(n).** If we solve this relation using Master's Theorem, we get the time complexity as **Θ(n*log(n)).**

Empirically, our findings have the trendline equation as **$y = 1.89x^{1.14}$**, we can clearly say it is better than **Θ(n),** and worse than **Θ(n^2)**. Thus, this algorithm, in the best case is a **Θ(n*log(n))** both theoretically and empirically.

- Worst Case:

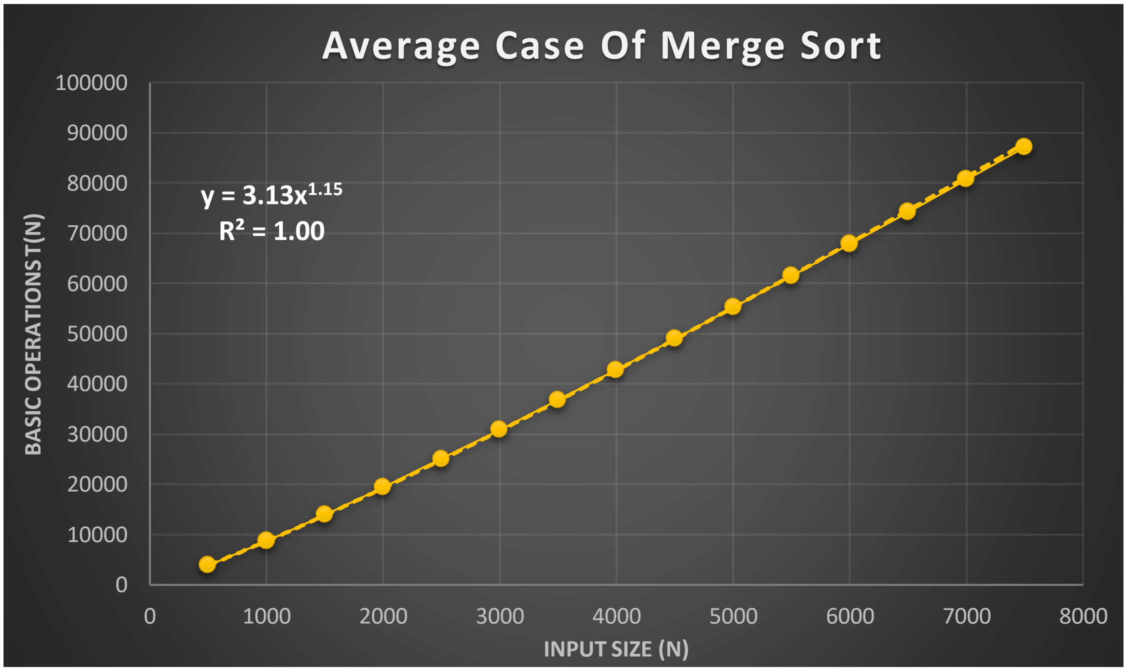


Worst case of Merge Sort occurs when we have a sample that meets the condition, that is we need to maximize the number of comparisons by having alternating elements in left and right subarrays. So, we created a sample list in our program to have maximum number of comparisons for Merge Sort.

Theoretically, it is **Θ(n*log(n))** just like the best case.

Empirically, trendline equation for the char above is, **$y = 3.28x^{1.15}$**, which is **Θ(n*log(n))** as we have explained above. And we can see even though asymptotically, it is same as the best case, it is worse exactly because the coefficient of x is greater than of the best case.

- Average Case:



Average case of Merge Sort occurs when we use a totally random list to sort. The number of comparisons when merging depends on the randomness of the sample.

Theoretically, its time complexity is the same as best and worst case as explained above, that is **Θ(n*logn)**.

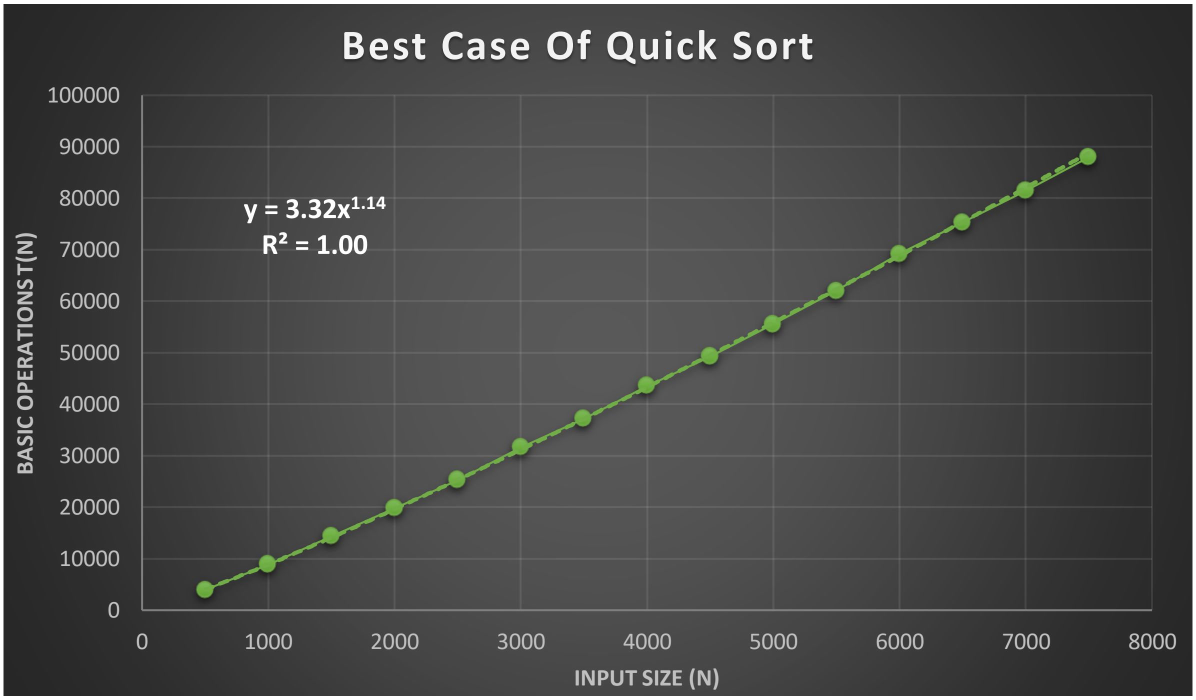
Empirically, the trendline equation for the data above is **$y = 3.13x^{1.15}$**, that is **Θ(n*logn)** algorithm as expected. And its exact complexity is somewhere between best case and worst case exactly, because of the coefficient of x.

4-) Quick Sort

Note 1: Hoare's algorithm is used for partitioning.

Note 2: Basic operation is selected as key comparisons.

- Best Case:

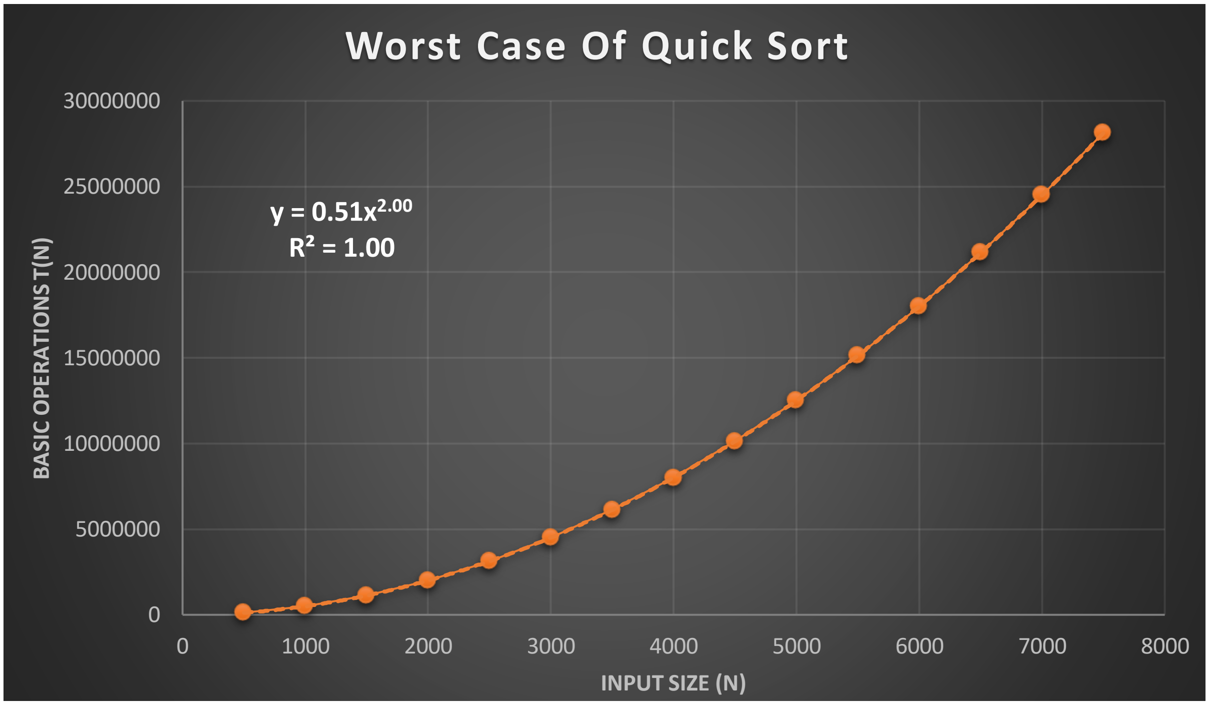


Best case of Quick Sort occurs when the pivot is always put in the middle of the list at each iteration. Simplest occurrence of this condition is when we use a sample where all the elements are the same. Because Hoare's algorithm will always place the pivot to the middle of the list.

Theoretically, this is a divide and conquer algorithm. And its recurrence relation for the best case is, **Tn = 2T(n/2) + O(n),** because we divide the array into two equal subarrays and apply partitioning which is **O(n)** complexity**.** If we solve this relation using Master's Theorem, we get the time complexity as, **Θ(n*logn).**

Empirically, trendline equation for the data we obtained is, **y = 3.32$x^{1.14}$** which has clearly **Θ(n*logn)** time complexity as we explained many times. And it meets the theoretical time complexity perfectly.

- Worst Case:

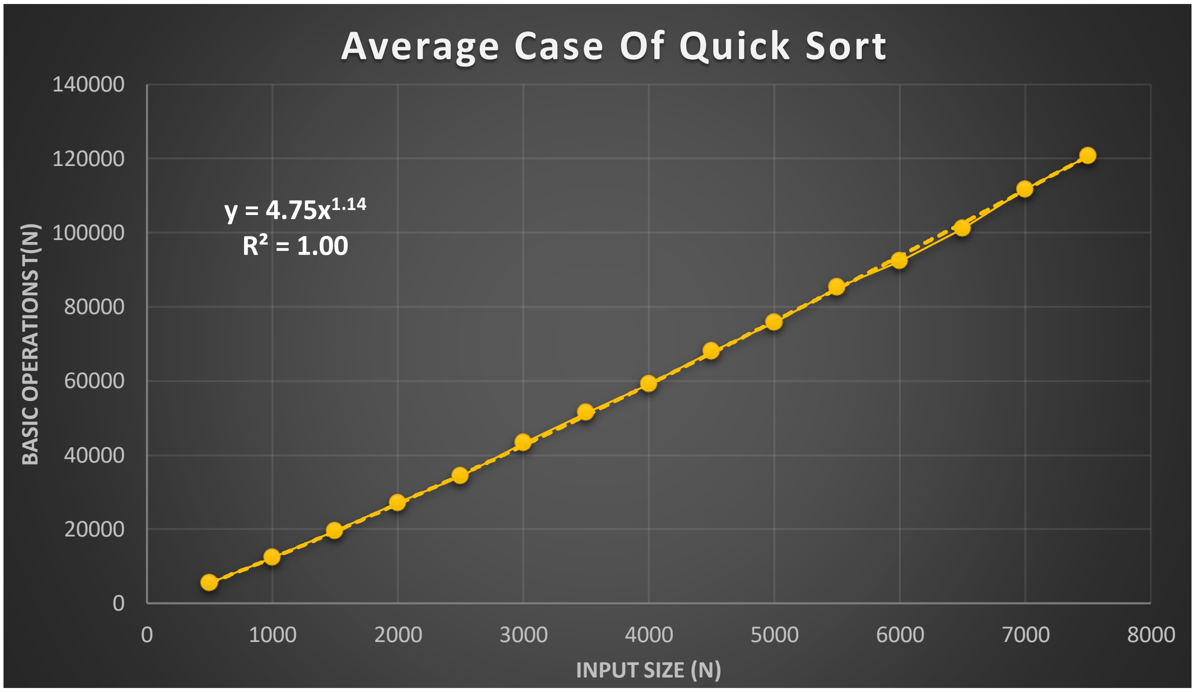


Worst case of Quick Sort occurs when the left and right part of the lists are skewed at each iteration. That happens when the pivot is selected as either first or last element at each iteration which causes the recursion tree to be skewed in one direction. And this condition happens if the list is already sorted.

Theoretically, for example assume we have the list {1, 2, 3, 4, 5}. After first partitioning, the pivot will be ‘1’ and the list will become {**1**, 2, 3, 4, 5}. As we can see, there is only right part of pivot, which has **T(n-1)** elements unlike the best case. The recurrence relation for the worst case then becomes **Tn = T(n-1) + O(n).** If we solve this relation using Master’s Theorem, we get the time complexity **$\Theta(n^2)$.**

Empirically, the trendline equation for our data is **$y = 0.51x^{2.00}$**, as can be seen from the chart above. And it is clearly seen that this is a **$\Theta(n^2)$** algorithm just like our theoretical expectation.

- Average Case:



Average case of Quick Sort occurs when we use a totally random list to sort. The number of comparisons will depend on the randomness of the list.

Theoretically, if we make the probability calculations, we get the time complexity as , **Θ(n*logn).** But it must be worse than the best case because we will not always split the array in half when we use a random array.

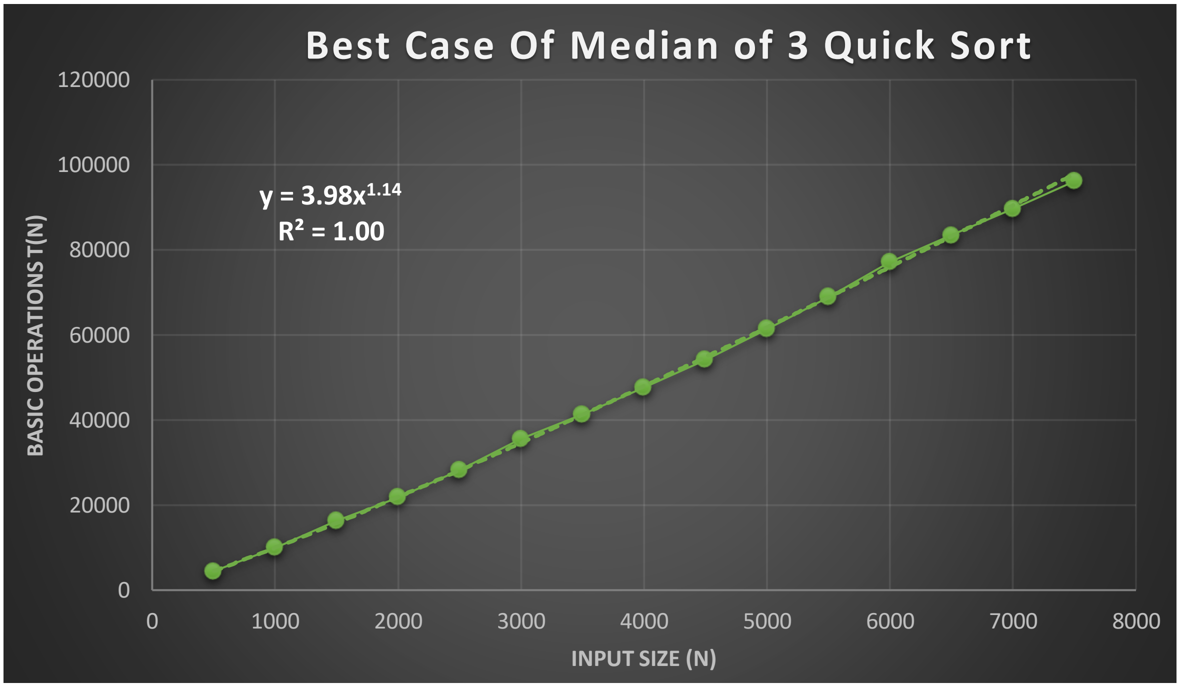
Empirically, the trendline equation for the data is, **$y = 4.75x^{1.14}$**, which has the complexity , **Θ(n*logn)** just like our theoretical expectations. And we can see this is worse than the best case exactly because coefficient **4.75** is greater than **3.32**. But of course, they have the same asymptotical complexity because x is raised to the same power which is **1.14.**

5-) Median of Three Quick Sort

Note 1: Hoare's algorithm is used for partitioning.

Note 2: Basic operation is selected as key comparisons.

- Best Case:

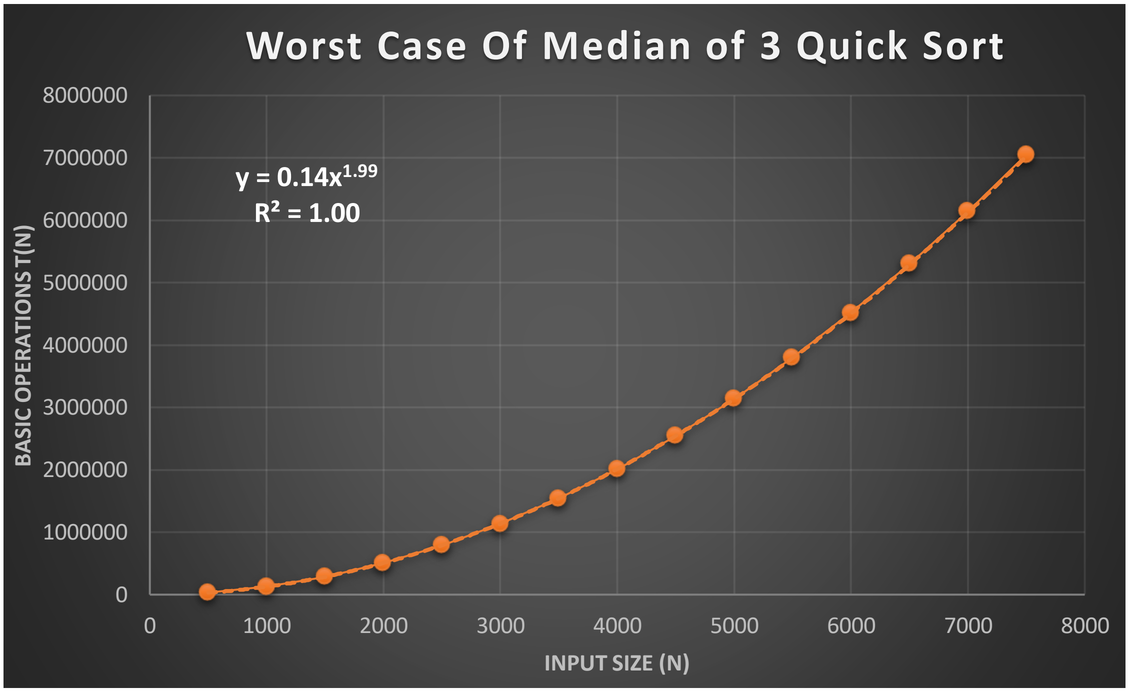


Best case of Median of Three Quick Sort occurs when we use a sample list where all the elements are the same. Partitioning will split the arrays in half at each recursive call just like the standard Quick Sort.

Theoretically, its recurrence relation is the same as standard Quick Sort in the best case that is **Tn = 2T(n/2) + O(n).** And theoretical time complexity then becomes, **Θ(n*logn).**

Empirically, our trendline equation for the chart above is, **y = 3.98x$^{1.14}$**, which is **Θ(n*logn),** just like the theoretical complexity**.** And if we compare this equation with the standard Quick Sort's best-case equation, we see that this is worse than standard quick sort exactly. That is because we make extra key comparisons to obtain the median of three.

- Worst Case:



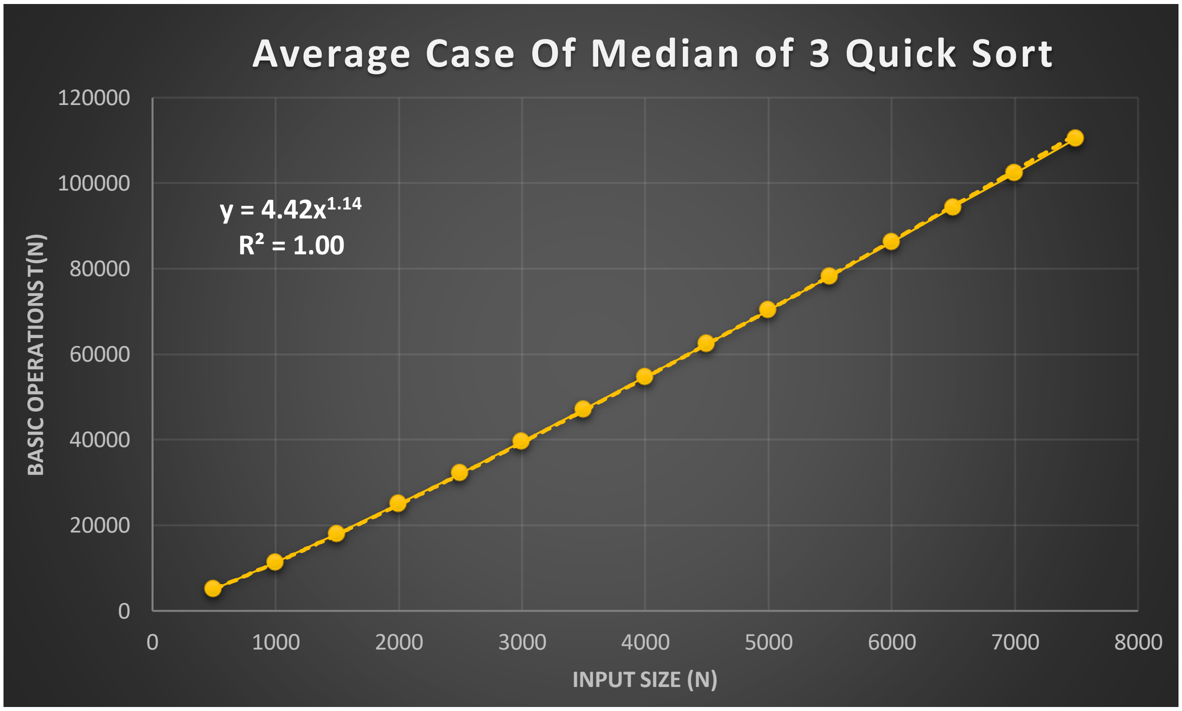
Worst case of Median of 3 Quick Sort occurs when we use a unique reversely sorted sample. At each iteration, the partitioning will cause rotated arrays to form. Thus, the recursion tree will be skewed to right.

Theoretically, let us assume we have the list {10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0}. The pivot is selected as ‘5’ and after the first partitioning the list will be : {4, 0, 1, 2, 3, 5, 10, 6, 7, 8, 9}. If we examine left and right subarrays, we find out they are rotated arrays. Let us now examine left part of the pivot. The median of ‘4’ ,’1’ and ‘3’ will be ‘1’ which is the second element of its sorted subarray. Similarly for the right part, ‘7’ will be selected as pivot. And it is also the second element of its subarray when sorted.

By this logic we can say the recurrence relation becomes, **Tn = T(n-2) + O(n)** because the arrays will always be split from the second element. If we solve this relation using Master Theorem, we get the time complexity **$\Theta(n^2)$.**

Empirically, our trendline equation is, **$y = 0.14x^{1.99}$**, which has the complexity **$\Theta(n^2)$** as expected. Also, we can see that Median-of- Three Quick Sort’s worst case is better than standard Quick Sort’s worst case because we eliminate two elements at each recursive call instead of one.

- Average Case:



Average case of Median-of-Three Quick Sort occurs when we use a totally random list to sort. The number of comparisons will depend on the randomness of the list.

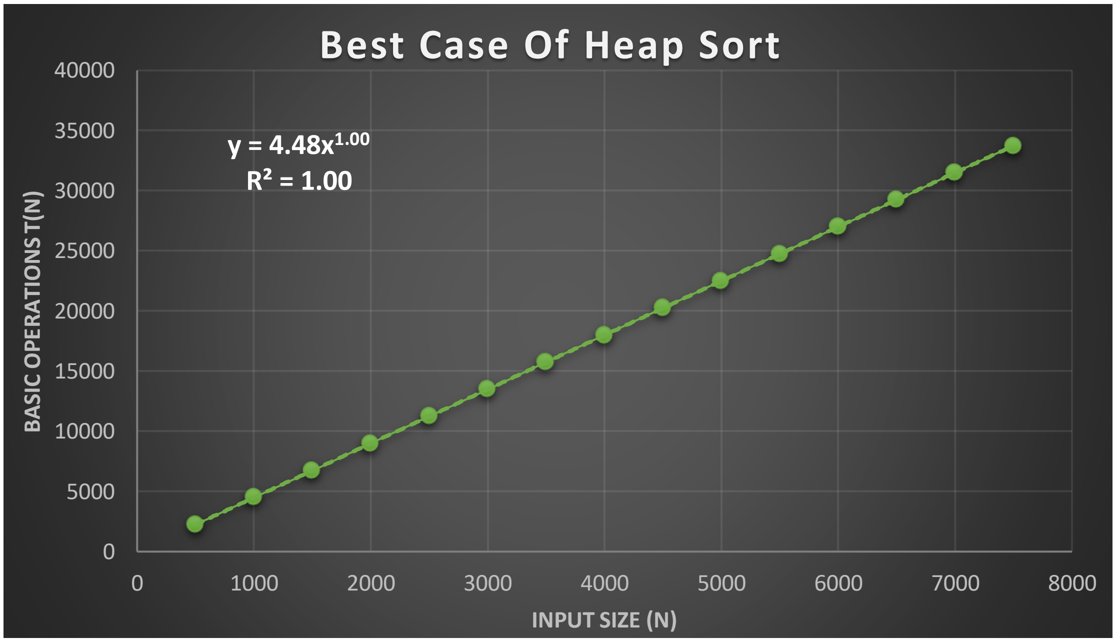
Theoretically, if we make the probability calculations, we get the time complexity as , **Θ(n*logn).** But it must be worse than the best case because we will not always split the array in half when we use a random array.

Empirically, the trendline equation for the data is, **$y = 4.42x^{1.14}$**, which has the complexity , **Θ(n*logn)** which meets the theoretical result. And we can see this is worse than the best case exactly because coefficient **4.42** is greater than **3.98**. But of course, they have the same asymptotical complexity because x is raised to the same power which is **1.14.** Also, if we compare the average case of Median-of-Three with the standard Quick Sort, we see this algorithm makes less comparisons than the standard quick sort in the average case at every input size.

6-) Heap Sort

Note: The basic operations are selected as number of comparisons and number of swap operations (delete max).

- Best Case:

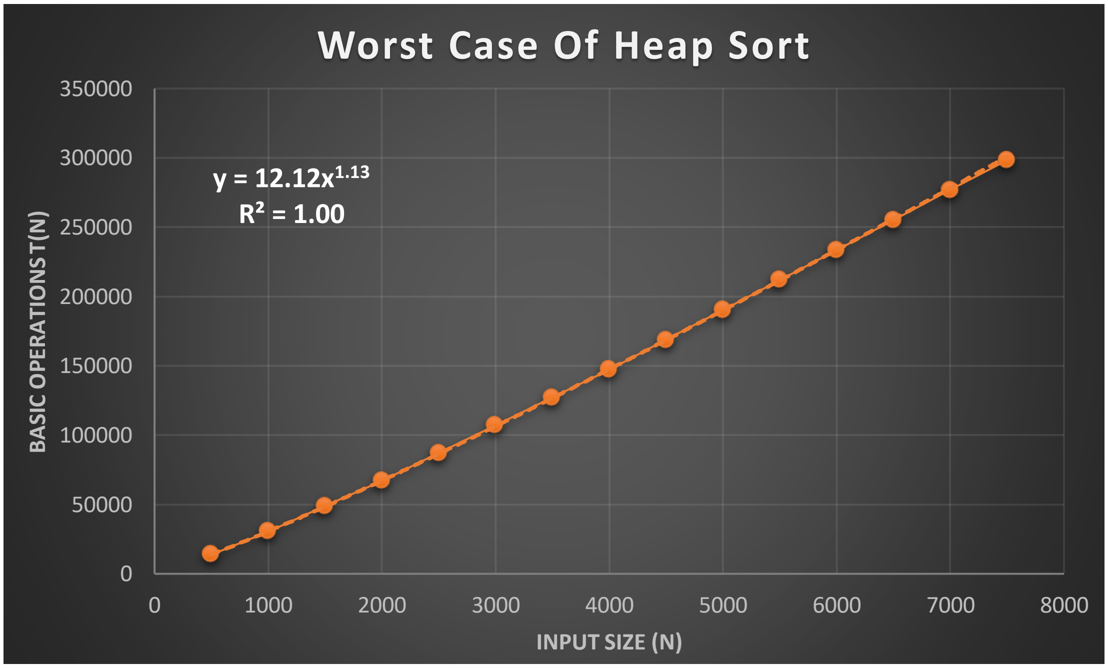


Best case of Heap Sort occurs when we use a sample which all the elements are same. Because that will cause heapify operations to be **Θ(1)** instead of **O(logn).**

Theoretically, it will call heapify operation nearly **n/2** times, and in the best case as we said above, the heapify operation has complexity **Θ(1),** since it will never call itself again. And **n/2 * Θ(1)** becomes **Θ(n/2).** After heapifying the list, we apply swap operation **n – 1** times, for deleting max node, which is **Θ(1).** If we add **Θ(n/2)** and **Θ(n-1),** the overall time complexity becomes **Θ(n).**

Empirically, our trendline equation is, **$y = 4.48x^{1.00}$** as can be seen from the above chart, which is in one-to-one correspondence with our theoretical result.

- Worst Case:

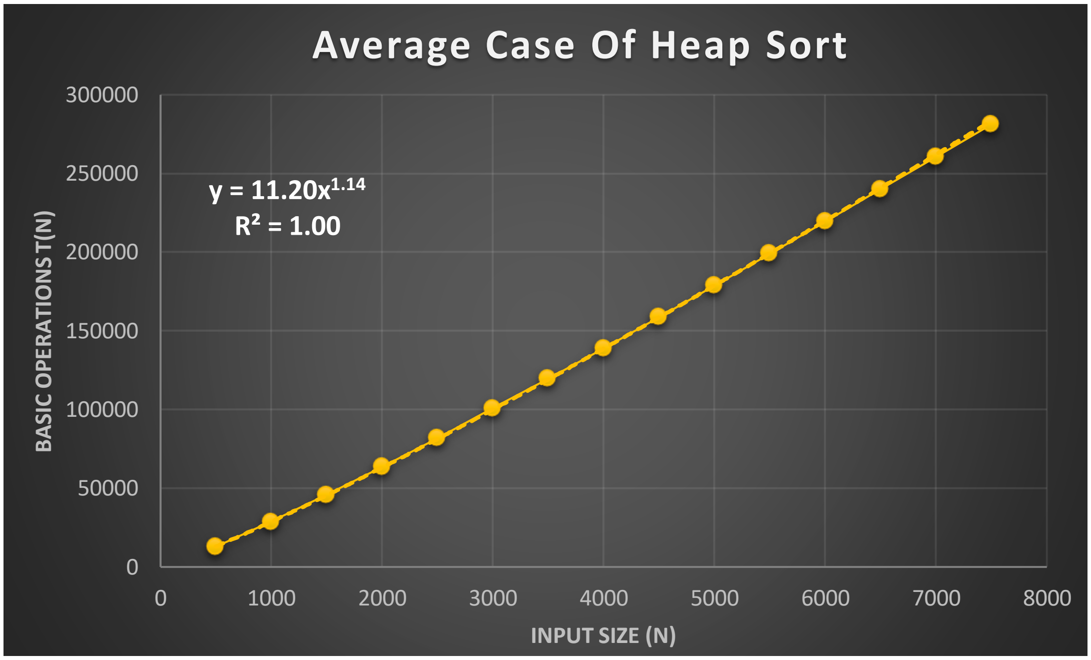


Obtaining a worst-case sample for the Heap Sort is a very complicated task. So, we used a sorted sample to represent the worst case for simplicity. Because transforming a sorted list to a max-heap is harder than transforming an average sample to a max-heap.

Theoretically, heapify operation takes **O(log(n))** time, and we call it **n/2** times, which makes **O(n*logn).** In addition to that, we apply delete max operation **n-1** times. If we solve the equation **O(n*logn) + Θ(n),** we obtain the result **O(n*logn).**

Empirically, our trendline equation for the basic operation count is, **$y = 12.12x^{1.13}$** as can be seen from the chart above. Thus, it is **O(n*logn)** just as we expected.

- Average Case:



Average Case of the Heap Sort occurs when we use a random sample. Count of Heapify operations will depend on the randomness of the sample.

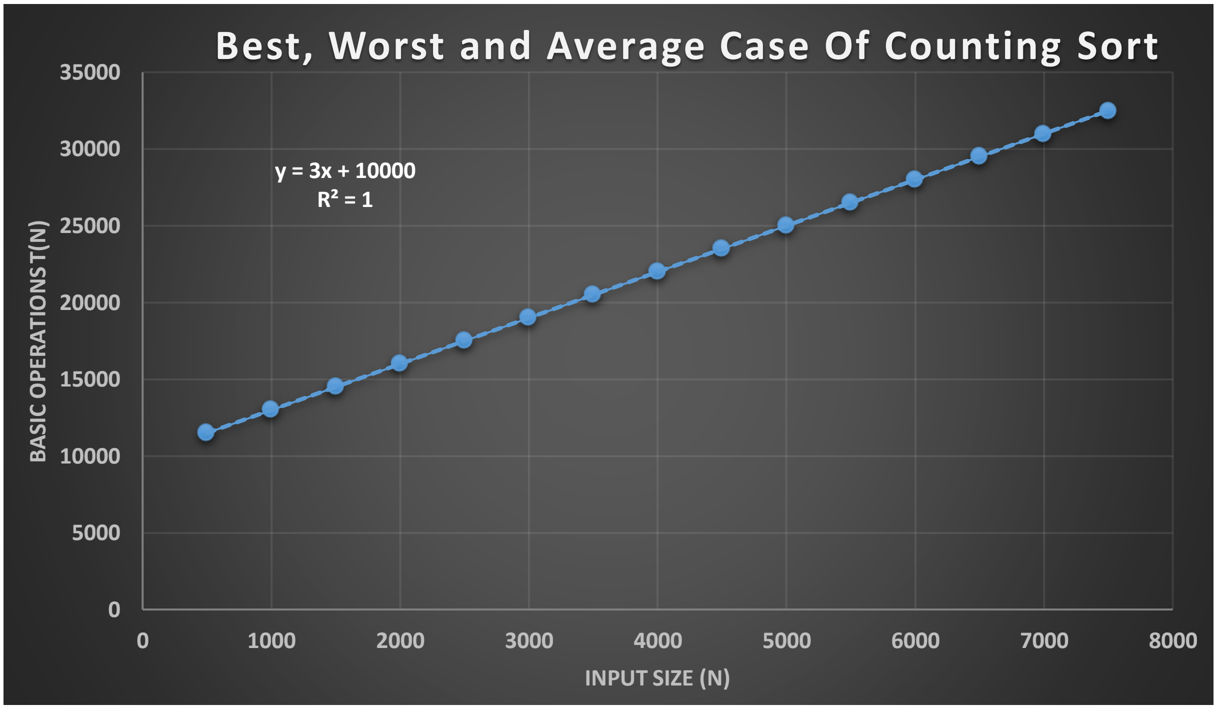
In average case, if we make proper probability analysis, we see heapify operation will also take **O(logn)** time**.** So, its asymptotic time complexity in the average case is the same as the worst case that is **O(n*logn).**

Empirically, our trendline equation for the basic operation is **y = 11.20x$^{1.14}$** which is **Θ(n*logn)** time complexity as we expected. Also, if we compare the coefficients of average case and worst case, we can see that **11.20** is less than **12.12.** Thus, we can say average case is better than the worst case exactly, despite being the same asymptotically(x is raised to the same number).

7-) Counting Sort

Note: The basic operation is selected as every assignment operation for the four for loops.

- Best, Worst and Average Case:



Best, worst, and average case time complexity of Counting Sort are the same (there is no certain sample to obtain different cases) for our algorithm.

Theoretically, the time complexity depends on the input size(n) and range of values(k). Because in our algorithm, three of four for loops depend on the input size, and one of the loops depend on the range(k). And our basic operations(assignment) have the time complexity of **Θ(1).** Thus, **[3 * (Θ(n) * Θ(1))] + [1 * (Θ(k))]** makes **Θ(3n + k) = Θ(n + k),** where k is the range of values.

Empirically, our trendline equation is **y = 3x + 10,000.** Where x is the input size and **10,000** is our range of values. Thus, we can clearly say our empirical result is **Θ(n + k)** and therefore, meets the theoretical time complexity of the Counting Sort algorithm.