**Analysis for Insertion Sort and Merge Sort**

**By Prachi Gupta, G01025257, User: pgupta13**

1. Counter values set in the code for operations in Insertion Sort with input as 10, 1000 and 100000 is given below:

|  |  |
| --- | --- |
| Value of n | Counter value |
| 10 | 336 |
| 1000 | 2275012 |
| 100000 | 1077416010 |

Insertion sort gives good performance for small inputs for example n =10. And slows down as the input increases, like for large inputs the counter notes a value of 2275012 for n = 1000 and 1077416010 for n = 100000.

For inputs like {9, 8, 7, 6, 5, 4, 3, 2, 1, 0} which is n = 10, gives worst case run time. For my code this notes a counter value of 615.

1. Counter values set in the code for operations in Insertion Sort with input as 10, 1000 and 100000 is given below:

|  |  |
| --- | --- |
| Value of n | Counter value |
| 10 | 606 |
| 1000 | 132627 |
| 100000 | 20334642 |

Merge sort notes greater number of operational steps for small inputs like n =10 while as the input increases, the performance increases and fastens the run time and notes a lesser value at counter for large input size when compared with insertion sort.

For inputs like {9, 8, 7, 6, 5, 4, 3, 2, 1, 0} which is n = 10, gives worst case run time. For my code this notes a counter value of 606, which is same as its average case when random integers are passed. This means that Merge sort doesn’t have a worst case performance. It is either best case or average case.

Another example is for n= 30 which is an already sorted array input, the counter notes 2384 while for random input.

As we can see, for small input our insertion sort performs fairly good taking minimum number of steps even with worst case performance. While on the other hand, our merge sort performs slower than our Insertion sort. As we go on increasing our input size opposite case happens. Merge sort gives better performance for Large input while Insertion sort slows down.

The asymptotic analysis of Insertion sort and Merge sort is given below. Note that they are theoretically proven formulas and have been seen in practical cases all practical cases

|  |  |  |  |
| --- | --- | --- | --- |
|  | Best Case | Average case | Worst Case |
| Insertion Sort | Ω(n) | Θ(n2) | O(n2) |
| Merge Sort | Ω(n log(n)) | Θ(n log(n)) | O(n log(n)) |

In the table we can see time complexity analysis for the two sorts for various cases. The table figures out that the merge sort has an average performance in all the three cases while the performance for insertion sort degrades when we increase input.

With this, I conclude by saying that, using Insertion sort for small inputs is a better choice than the other sorting algorithm. While if you have a large set of data, the best option from both is to go with merge sort