Link to GitHub repository: https://github.com/iheathers/CE_III_ROLL_NO_52__Lab2

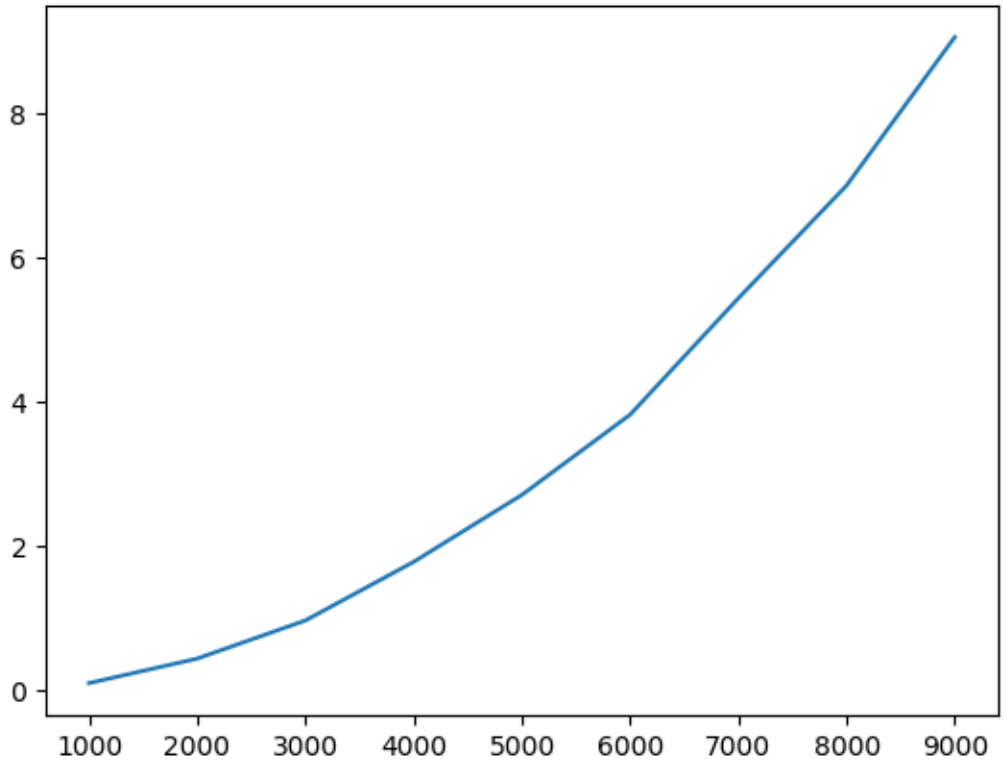

Fig: Time complexity of insertion sort

The insertion sort algorithm iterates through an input array and removes one element per iteration, finds the place the element belongs in the array, and then places it there. This process grows a sorted list from left to right.

Pseudocode of insertion sort algorithm:

```
for i = 1 to length(A)
   x = A[i]
   j = i - 1
   while j >= 0 and A[j] > x
      A[j+1] = A[j]
```

```
    j = j - 1
  end while
  A[j+1] = x
end for
```

Complexity of Insertion Sort:

Insertion sort runs in O(n) time in its best case and runs in $\Omega(n^2)$ in its worst and average cases.

Best Case Analysis:

Insertion sort performs two operations: it scans through the list, comparing each pair of elements, and it swaps elements if they are out of order. Each operation contributes to the running time of the algorithm. If the input array is already in sorted order, insertion sort compares O(n). Therefore, in the best case, insertion sort runs O(n) in time.

Worst and Average Case Analysis:

The worst case for insertion sort will occur when the input list is in decreasing order. To insert the last element, we need at most n-1 comparisons and at most n-1 swaps. To insert the second to last element, we need at most n-2 comparisons and at most n-2 swaps, and so on.

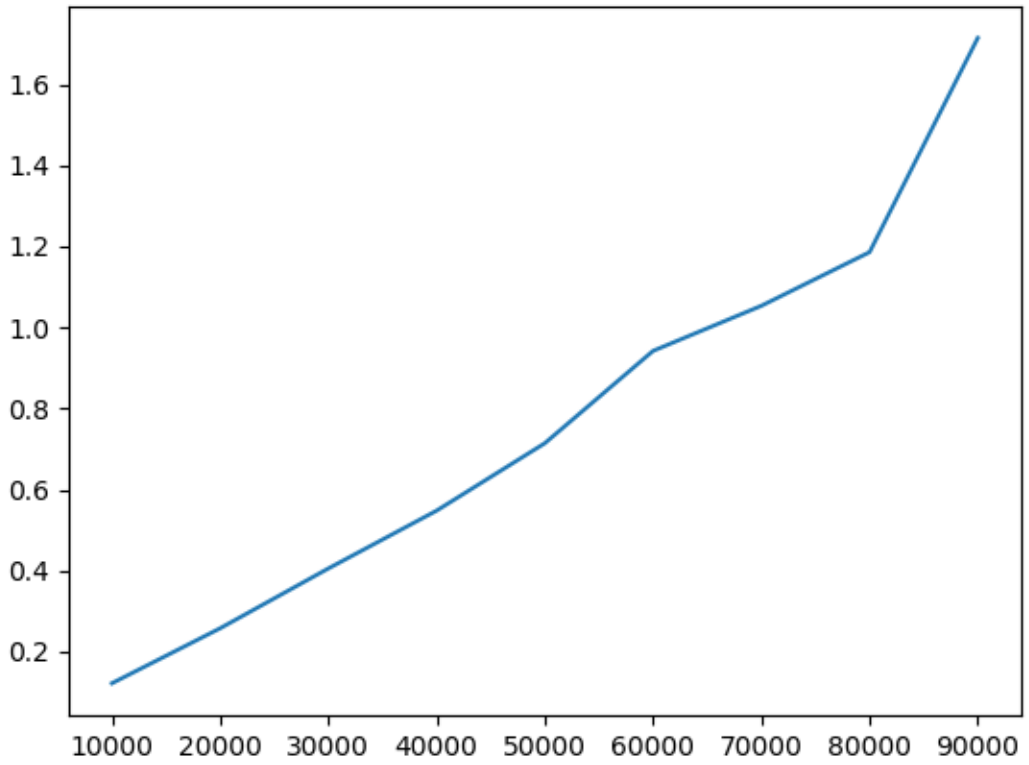

Fig: Time complexity of merge sort

Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. The merge() function is used for merging two halves. The merge(A, p, q, r)is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one.

Pseudocode for Merge Sort:

MERGE-SORT(A, p, r)

1. If p < r
2. q = [(p + q) /2]
3. MERGE-SORT(A, p, q)
4. MERGE-SORT(A, q+1, r)
5. MERGE(A, p, q, r)

MERGE (A, p, q, r)

1. $n_1 = q – p +1$
2. $n_2 = r – q$
3. let L [1.. $n_1 + 1$] and L [1.. $n_2 + 1$] be new arrays
4. for i=1 to n_1
5. L[i] = A [p + i -1]
6. for j=1 to n_2
7. R[j]= A[q + j]
8. L [$n_1 + 1$] = ∞
9. R [$n_2 + 1$] = ∞
10. i = 1
11. j = 1
12. for k = p to r
13. if L[i] $\leq$ R [j]
14. A[k] = L[i]
15. i = i + 1
16. else A[k] = R [j]
17. j = j + 1

Complexity of Merge Sort:

Time complexity of Merge Sort is $\Theta(n*\log n)$ in all 3 cases (worst, average and best) as merge sort always divides the array in two halves and take linear time to merge two halves.