Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavrepalanchowk

Lab report on **Lab 2**

IMPLEMENTATION, TESTING AND PERFORMANCE MEASUREMENT OF INSERTION AND MERGE SORT ALGORITHMS

Sub Code: COMP 314

Submitted by:

Saskar Khadka, CE

Roll No: 27

Submitted to:

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submission Date: 21 April, 2023

LAB 2:

Implementation, testing and performance measurement of of insertion and merge sort algorithms

Merge Sort:

Merge sort is a sorting algorithm that uses the Divide and Conquer strategy to sort a given array. It works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays together to form the final sorted array. It is a stable sorting algorithm, meaning it maintains the relative order of items with equal sort keys. The average case complexity of merge sort is O(nlog(n))

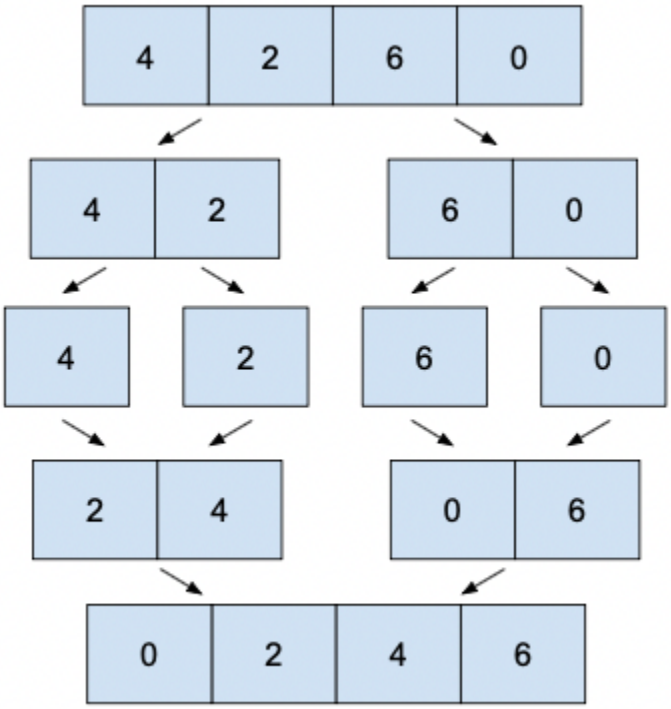


Fig 1: Merge Sort in action

Insertion Sort:

Insertion sort is a sorting algorithm that works similar to the way card players sort cards in their hands. Here the array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sort part. Just like merge sort, insertion sort is also stable. The average case complexity of insertion sort is $O(n^2)$

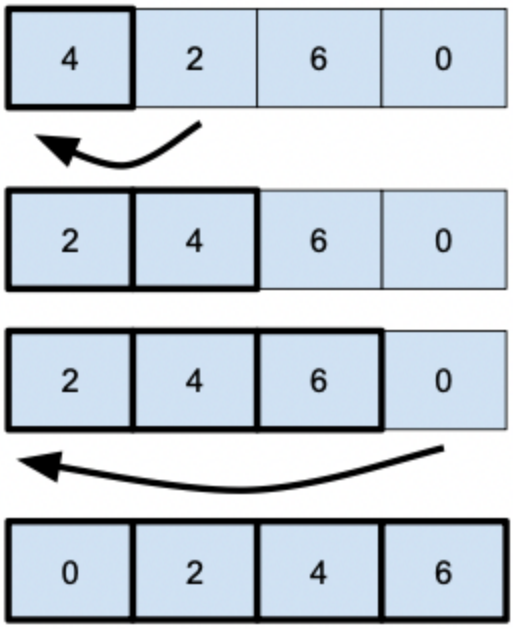


Fig 2: Insertion Sort in action

1. Implementation of Insertion Sort and Merge Sort.

[Source Code]

1.1 Output:

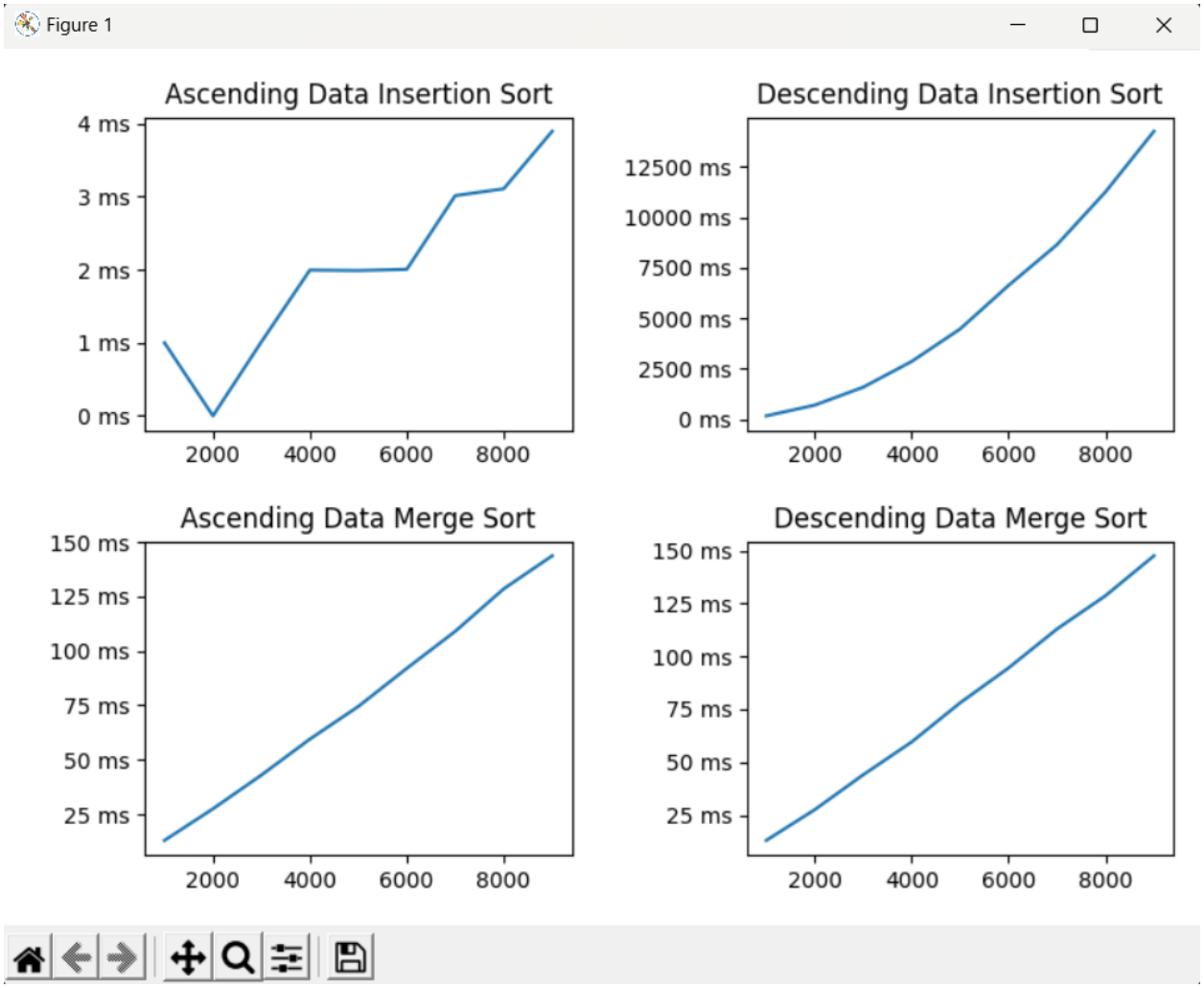
```
PS C:\Users\Saskarkhadka\OneDrive\Desktop\Algorithms-Labs\Lab 2\src> py main.py
Array before insertion sort: [4, 6, 1, 7, 2, 9, 3]
Array after insertion sort: [1, 2, 3, 4, 6, 7, 9]
Array before merge sort: [4, 6, 1, 7, 2, 9, 3]
Array after merge sort: [1, 2, 3, 4, 6, 7, 9]
```

1.2 Test Output:

```
PS C:\Users\Saskarkhadka\OneDrive\Desktop\Algorithms-Labs\Lab 2\src> py test.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

1.2 Performance Measurement Output:



Observations:

Through performance testing, it was found that when the data was already sorted insertion sort performed extremely well just as it should since the best case complexity of insertion sort is linear O(n). However, when the data was arranged in descending order then the graph showed that the time taken to sort the array increased almost quadratically with the input size i.e a complexity of $O(n^2)$.

While the performance of insertion sort varied vastly for ascending and descending data, Merge Sort performed almost the same for both of the cases. The graph for Merge Sort seems to be mimicking the nlog(n) order in both of the cases.