**Kathmandu University**

**Department of Computer Science and Engineering**

**Dhulikhel, Karve**

# COMP 314

Lab 2 Report

Sorting

**Submitted By:**

Mohit Pokharel

Roll No.-42

CE, 3rd Year, 2nd Sem

**Submitted to:**

Rajani Chulyadyo

Department of Computer Science and Engineering

# Lab 1: Implementing, Testing, and Performance measurement for Insertion Sort and Merge Sort

## Implementation

Programming Language Used:- Python

Libraries used: unittest, sample, time

## Insertion Sort

Insertion sort is a simple and efficient sorting algorithm that works by sorting an array one element at a time. It iterates through the array, comparing each element with the elements before it, and moving it to its correct position in the sorted array. Insertion sort has an average and worst-case time complexity of O(n^2).

It's Implementation is as follows:

```
def insertion_sort(arr):
    i = 1
    n = len(arr)
    while i < n:
        key = arr[i]
        m = i - 1
        while m >= 0 and arr[m] > key:
            arr[m+1] = arr[m]
            m -= 1
        arr[m+1] = key
        i += 1
    return arr
```

## Merge Sort

The basic idea behind merge sort is to divide the list into smaller sub-lists, sort those sub-lists recursively, and then merge them back together to produce the final sorted list. The algorithm has a time complexity of O(nlogn). It is also a stable sorting algorithm, which means that it maintains the relative order of equal elements in the input list.

It's implementation is as follows:

```
def merge(arr, p, q, r):
    n1 = q - p + 1
    n2 = r - q
    L1 = [arr[p + i] for i in range(n1)]
    R = [arr[q + i + 1] for i in range(n2)]
    L1.append(float('inf'))
    R.append(float('inf'))
    i, j = 0, 0
     for k in range(p, r+1):
            if L1[i] < R[j]:
            arr[k] = L1[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
def merge_sort(arr, p, r):
    if p < r:
        q = int((p+r)/2)
        merge_sort(arr, p, q)
        merge_sort(arr, q+1, r)
        merge(arr, p, q, r)
```

## Test Cases

Python library unittest was used to test both of the searching algorithms by providing them with a limited data set and inputting both the generated result and the expected result.

The implementation for insertion sort is as follows:

```
import unittest

from insertion import insertion_sort

class TestSum(unittest.TestCase):

    def test_insertion_sort(self):

        input = [3,6,4,2,8,1]

        sorted = insertion_sort(input)

         res = [1,2,3,4,6,8]

         self.assertListEqual(res,sorted)

if __name__ == '__main__':

    unittest.main()
```

Its implementation for Merge sort is as follows:

```
import unittest

from merge import merge_sort


class TestSum(unittest.TestCase):

     def test_insertion_sort(self):

        input  = [3,6,4,2,8,1]

         merge_sort(input,0,5)

         res = [1,2,3,4,6,8]

         self.assertListEqual(input,res)


if __name__ == '__main__':

    unittest.main()
```

## Time Calculation

To measure the time taken by the sorting algorithms, we used the time_ns() function provided by the time library in Python. The time before the execution of the sorting algorithm was noted and the time after the completion was noted, the difference between the two was calculated which is the time taken by the algorithm.

The implementation is as follows:

```
import random

from merge import merge_sort

from insertion import insertion_sort

import time


def run_ins(data):

    start_time= time.time_ns()

    insertion_sort(data)

    end_time=time.time_ns()

    time_taken=end_time- start_time

    return time_taken

def run_mer(data,n):

    start_time1= time.time_ns()

    merge_sort(data,0,n-1)

    end_time1=time.time_ns()

    time_taken1=end_time1- start_time1

    return time_taken1


n=10000

data=[random.randint(1, n) for _ in range(n)]

#c=[run_ins(data) for i in range (1,5)]

c=[run_mer(data,n) for i in range (1,5)]

d=max(c)

print(d)
```

## Result

The graph obtained form Insertion Search is:

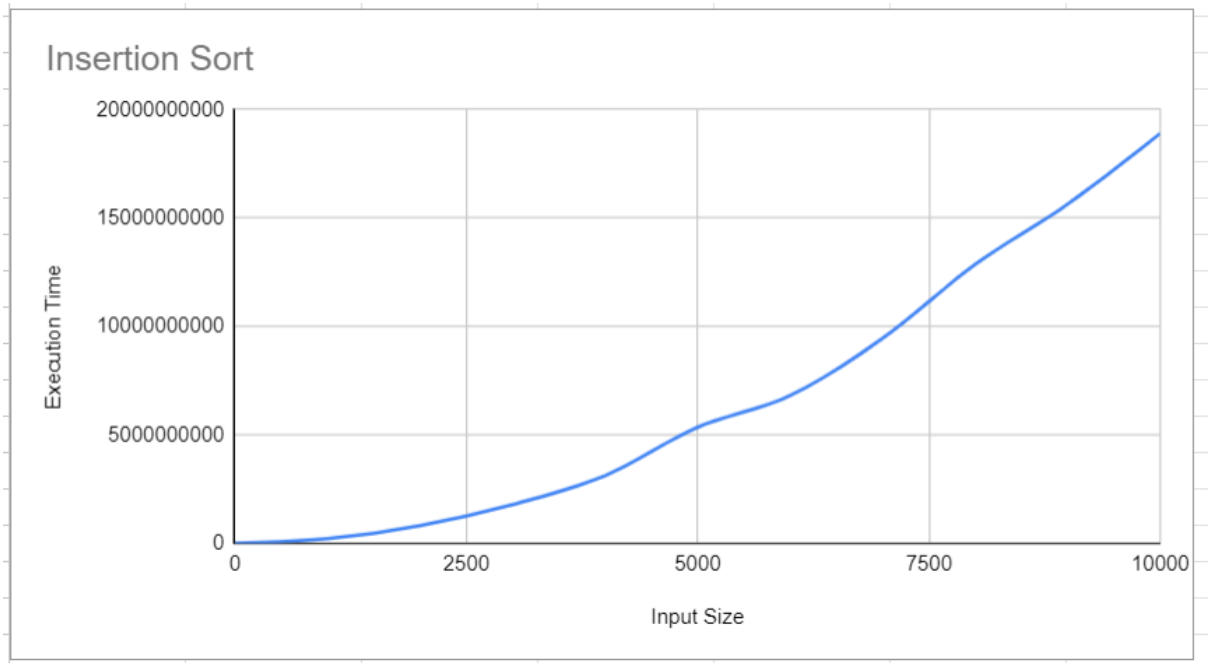


*Fig: Insertion Sort*

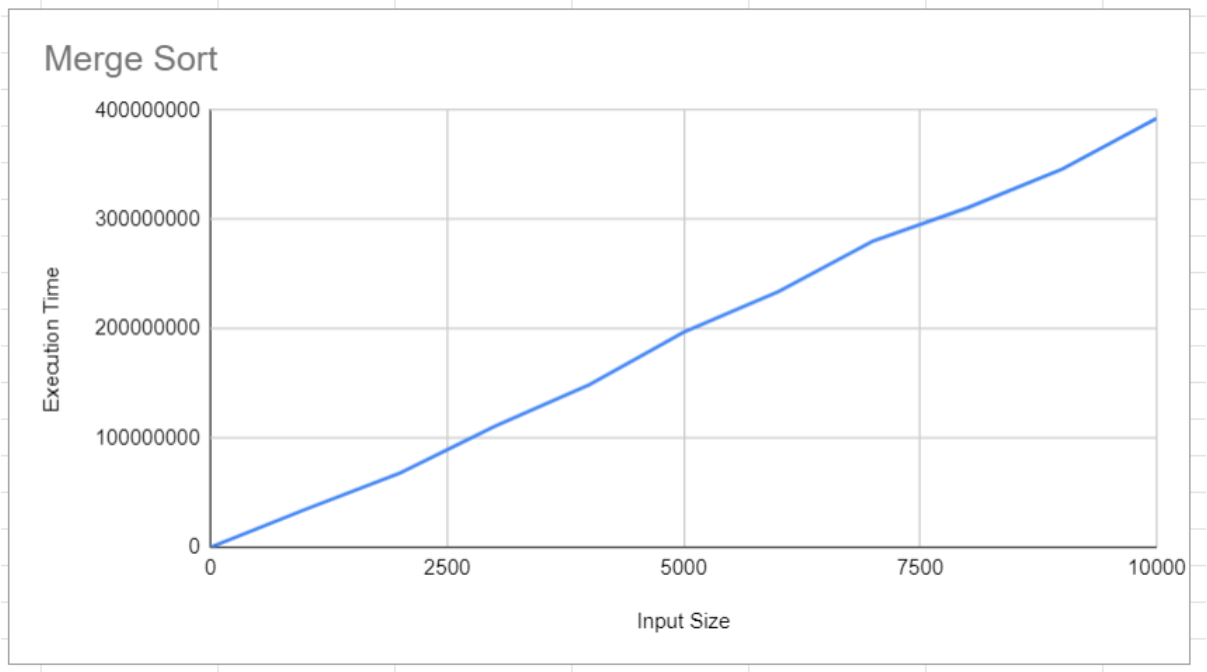


*Fig: Merge Sort*

## Conclusions

By the end of this Project, Insertion and Merge sorting algorithms were implemented, tested and their time consumption was calculated. The result came as per out assumption as the graph of insertion sort shoes quadratic curve while thar of merge sort shows almost linear property with a slight curve.