Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Lab Report 2

COMP 314

(For partial fulfillment of 3rd Year/ 2nd Semester in Computer Engineering)

Submitted to:

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submitted by:

Neha Malla

C.E.

Roll No. 27

1. Implement the following sorting algorithms:

   (a) Insertion sort

   (b) Merge sort

   ➔ Filename: insertionsort.py

```
def insertionsort(array):

    for j in range(1, len(array) - 1):

        key = array[j]

        i = j - 1

        while i >= 0 and array[i] > key:

            array[i+1] = array[i]

            i -= 1

        array[i+1] = key
```

   ➔ Filename: mergesort.py

```
import math


def mergesort(array, p, r):

    if p < r:

        q = math.floor((p + r)/2)

        mergesort(array, p, q)

        mergesort(array, q + 1, r)

        merge(array, p, q, r)


def merge(array, p, q, r):

    n1 = q - p + 1

    n2 = r - q
```

```
    L = []
    R = []


    for i in range(n1):
        L.append(array[p+i])


    for j in range(n2):
        R.append(array[q+j+1])


    L.append(math.inf)
    R.append(math.inf)


    i, j = 0, 0


    for k in range(p, r + 1):
        if L[i] <= R[j]:
            array[k] = L[i]
            i += 1
        else:
            array[k] = R[j]
            j += 1
```

2. Write some test cases to test your program.

➔ Filename: sortTest.py

```
import unittest
from mergesort import mergesort
from insertionsort import insertionsort


class TestSort(unittest.TestCase):
    def test_mergesort(self):
        data = [1, 4, 3, 9, 6, 12, 7, 4, 8]
        dataS = data
        mergesort(data, 0, len(data) - 1)
        dataS.sort()
        self.assertEqual(data, dataS)


    def test_insertionsort(self):
        data = [1, 4, 3, 9, 6, 12, 7, 4, 8]
        dataS = data
        insertionsort(data)
        dataS.sort()
        self.assertEqual(data, dataS)


if __name__ == "__main__":
    unittest.main()
```

3. Generate some random inputs for your program and apply both insertion sort and merge sort algorithms to sort the generated sequence of data. Record the execution times of both algorithms for inputs of different size. Plot an input-size vs execution-time graph.

➔ Filename: sortMain.py

```
from mergesort import mergesort

from insertionsort import insertionsort

from time import time

import random

import matplotlib.pyplot as plt



mergeTime = []

insertTime = []

i = 10000

dataSize = []


while i <= 100000:

    dataSize.append(i)

    dataM = random.sample(range(i), i)

    dataI = dataM


    mergeStart = time()

    mergesort(dataM, 0, len(dataM) - 1)

    mergeEnd = time()

    mergeTime.append(mergeEnd - mergeStart)


    insertStart = time()
```

```
    insertionsort(dataI)
    insertEnd = time()
    insertTime.append(insertEnd - insertStart)

    i += 10000



print("Time taken:")
print("For merge sort: ", mergeTime)
print("For insertion sort: ", insertTime)



plt.plot(dataSize,mergeTime,"g")
plt.show()

plt.plot(dataSize,insertTime,"r")
plt.show()
```

4. Explain your observations.

➔ **Output:**

Time taken:

For insertion sort: [0.03889632225036621, 0.16356301307678223, 0.37902402877807617, 0.8386895656585693, 0.9156358242034912, 1.304849624633789, 1.884899616241455, 2.5383729934692383, 2.895573854446411, 3.6321587562561035]

For merge sort: [0.02493739128112793, 0.0484774112701416, 0.08228635787963867, 0.13022089004516602, 0.14455914497375488, 0.14866971969604492, 0.20339035987854004, 0.274334192276001, 0.2323904037475586, 0.26180553436279297, 0.2897305488586426, 0.3181612491607666, 0.3510725498199463, 0.37999558448791504, 0.4738898277282715, 0.4849660396575928, 0.6081764698028564, 0.533740758895874, 0.6203415393829346, 0.8856303691864014]

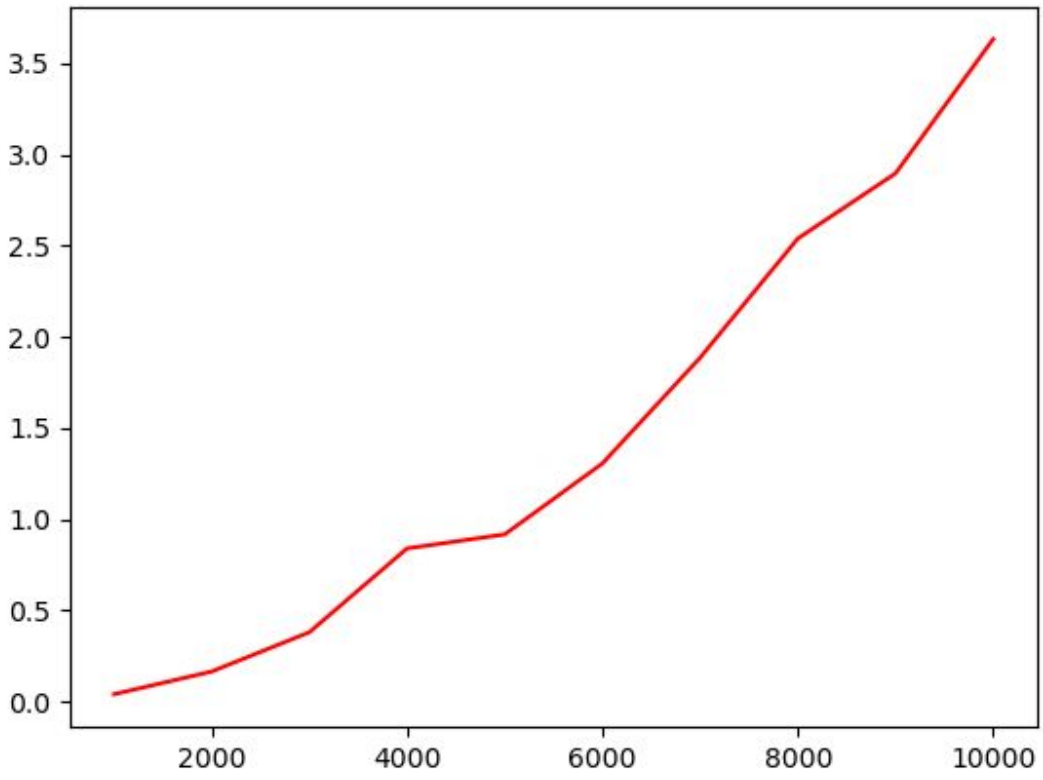

Figure: Insertion Sort Time Complexity

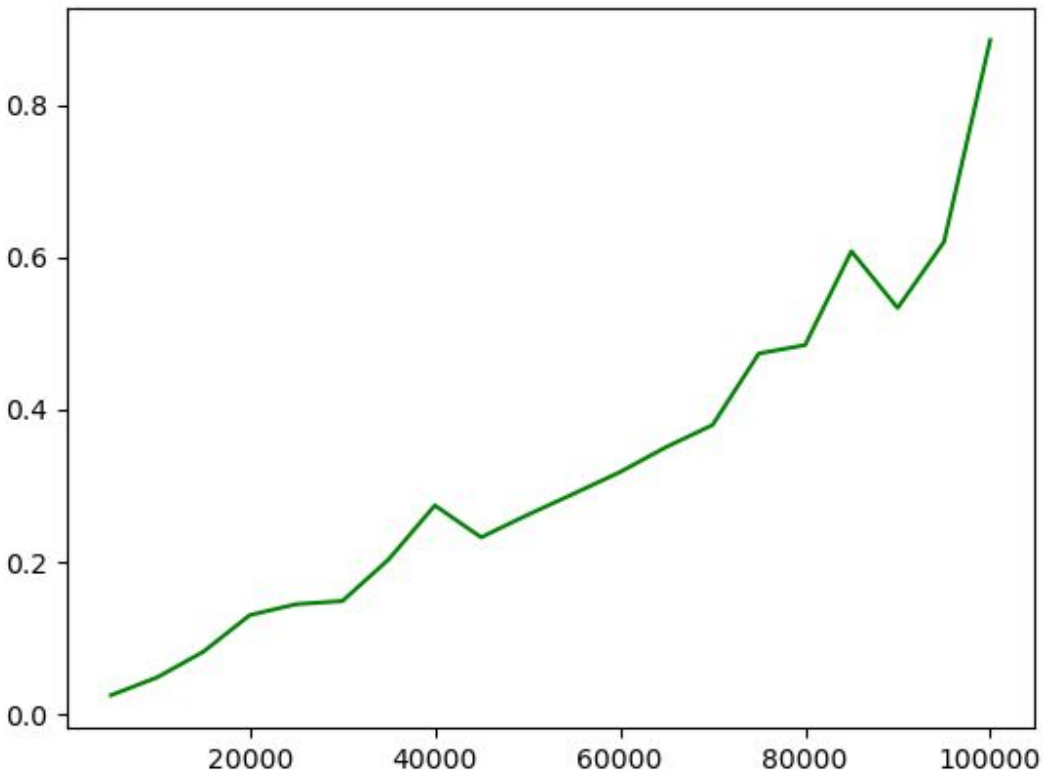

Figure: Merge Sort Time Complexity

Insertion Sort Time Complexity:

The time complexity is O(n^2).The graph above is a curve similar to n^2 curve even for a small number of input data. The max data is 10000 and the step is 1000.

Merge Sort Time Complexity:

The time complexity is O(n log n). The curve above tends to be along the nlog n curve but isn't quite like it due to small data size. The max data size is 100000 and the step is 5000.