# KATHMANDU UNIVERSITY

## Department of Computer Science and Engineering

**Dhulikhel, Kavre**

## Lab Sheet 2

## Algorithm and Complexity

**[Course Code: COMP 314]**

**Submitted by:**

Sajag Silwal (Roll no: 48)

**Submitted to:**

Dr. Rajani Chulyadyo
Department of Computer Science and Engineering

**Submission Date:** 24-03-2022

# Implementation of Merge Sort and Insertion Sort

## Merge Sort

Merge Sort is quite similar to the Quicksort algorithm as it is based on the “Divide and Conquer” algorithm. It fop-down recursion and divides the array into two halves, calls itself for the two halves, and merges the sorted array.

Its simulation is given below:

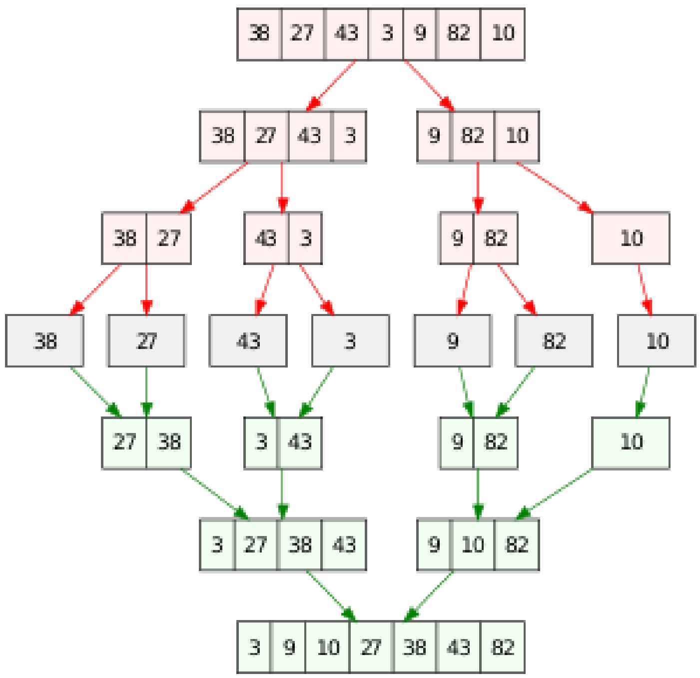


*Figure 1: Simulation of Merge Sort*

## Source Code:

The Source Code for Merge Sort Implementation is given below:

```
def MergeSort(array, left, right):
    if left >= right:
        return

    middle = (left + right)//2
    MergeSort(array, left, middle)
    MergeSort(array, middle + 1, right)
    Merge(array, left, right, middle)
```

```python
def Merge(array, left, right, middle):

    leftCopy = array[left:middle + 1]
    rightCopy = array[middle+1:right+1]
    leftCopyIndex = 0
    rightCopyIndex = 0
    sortedIndex = left

    while leftCopyIndex < len(leftCopy) and rightCopyIndex < len(rightCopy):

        if leftCopy[leftCopyIndex] <= rightCopy[rightCopyIndex]:
            array[sortedIndex] = leftCopy[leftCopyIndex]
            leftCopyIndex = leftCopyIndex + 1

        else:
            array[sortedIndex] = rightCopy[rightCopyIndex]
            rightCopyIndex = rightCopyIndex + 1

        sortedIndex = sortedIndex + 1


    while leftCopyIndex < len(leftCopy):
        array[sortedIndex] = leftCopy[leftCopyIndex]
        leftCopyIndex = leftCopyIndex + 1
        sortedIndex = sortedIndex + 1

    while rightCopyIndex < len(rightCopy):
        array[sortedIndex] = rightCopy[rightCopyIndex]
        rightCopyIndex = rightCopyIndex + 1
        sortedIndex = sortedIndex + 1
```

## Output:

```
PS C:\Users\sajag\Desktop\6th Sem\COMP 314\Lab Works\Lab 2> & C:/Users/sajag/AppData/
exe "c:/Users/sajag/Desktop/6th Sem/COMP 314/Lab Works/Lab 2/AlgoLab2/MergeSort.py"
Input Array :    [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
Sorted Array :   [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
PS C:\Users\sajag\Desktop\6th Sem\COMP 314\Lab Works\Lab 2>
```

## Test Case for Merge Sort

The test case for the merge sort is given below:

```
from MergeSort import MergeSort
import unittest

input1  =  [1,2,3,4,5,6,7,8,9,10]
output1 =  [1,2,3,4,5,6,7,8,9,10]
input2 = [9,1,21,5,6,8,109,4,20,50]
output2  =  [1,4,5,6,8,9,20,21,50,109]
input3  =  [10,9,8,7,6,5,4,3,2,1]
output3  = [1,2,3,4,5,6,7,8,9,10]

r1=len(input1)
r2=len(input2)
r3=len(input3)

class MergeSortCase(unittest.TestCase):
    def test_MergeSort(self):
        MergeSort(input1,0,r1)
        MergeSort(input2,0,r2)
        MergeSort(input3,0,r3)
        self.assertEqual(input1,output1)
        self.assertEqual(input2,output2)
        self.assertEqual(input3,output3)


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

## Output:

The output for the test case is given below:

```
PS C:\Users\sajag\Desktop\6th Sem\COMP 314\Lab Works\Lab 2> & C:/Users/sajag/AppData/Loca
jag/Desktop/6th Sem/COMP 314/Lab Works/Lab 2/AlgoLab2/test.py"
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
PS C:\Users\sajag\Desktop\6th Sem\COMP 314\Lab Works\Lab 2>
```

## Insertion Sort

Insertion sort is a simple sorting algorithm that is similar to the way playing card is sorted in hands. In this method, an array is divided into two parts: Sorted and Unsorted. It is sorted gradually from left to right. The simulation of insertion sort is given below:

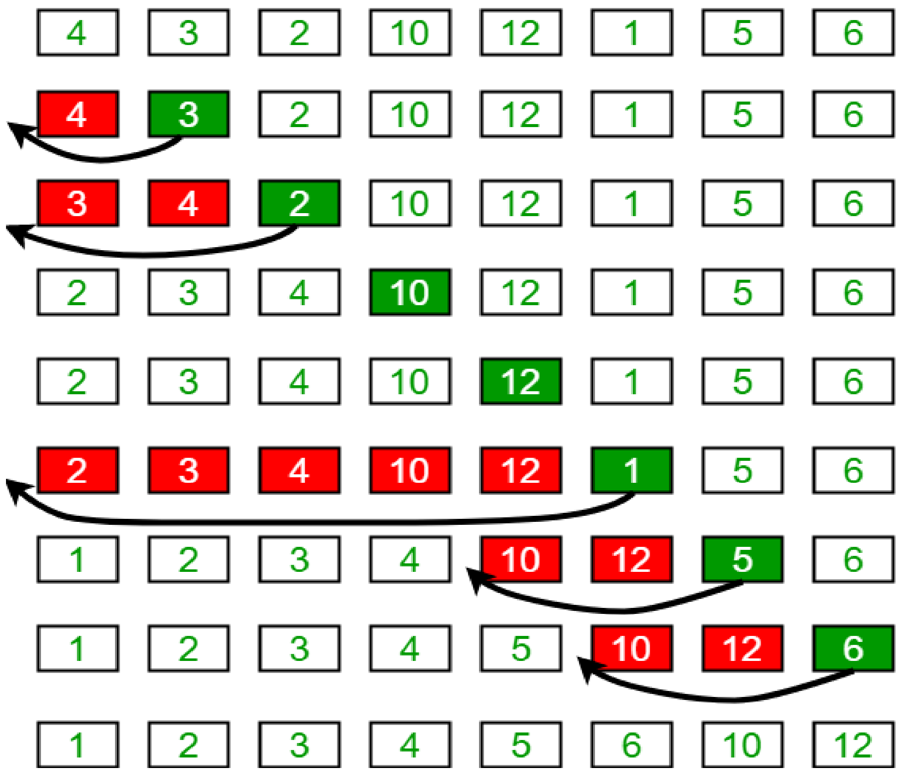


*Figure 2: Simulation of Insertion Sort*

## Source Code:

The source code for the Insertion Sort is :

```
def InsertionSort(A):
    n  = len(A)
    for j  in range (1,n):
        key =A[j]
        i = j-1
        while i>=0  and A[i]>key:
            A[i+1] = A[i]
            i  =  i-1
        A[i+1] =  key

array= [10,9,1,8,3,2,4,5,90,5]
array= [10,9,1,8,3,2,4,5,90,5]
print(f"Input Array : \t {array}")
InsertionSort(array)
```

```
print(f"Sorted Array : \t {array}")
```

## Output:

The output of the source code above is:

```
PS C:\Users\sajag\Desktop\6th Sem\COMP 314\Lab Works\Lab 2> & C:/Users/sajag/AppData/Local/
jag/Desktop/6th Sem/COMP 314/Lab Works/Lab 2/AlgoLab2/InsertionSort.py"
Input Array :    [10, 9, 1, 8, 3, 2, 4, 5, 90, 5]
Sorted Array :   [1, 2, 3, 4, 5, 5, 8, 9, 10, 90]
PS C:\Users\sajag\Desktop\6th Sem\COMP 314\Lab Works\Lab 2>
```

# Test Case for Merge Sort

The test case for insertion sort is given below:

```
from InsertionSort import InsertionSort
import unittest

input1  =  [1,2,3,4,5,6,7,8,9,10]
output1 =  [1,2,3,4,5,6,7,8,9,10]
input2 = [9,1,21,5,6,8,109,4,20,50]
output2  =  [1,4,5,6,8,9,20,21,50,109]
input3  =  [10,9,8,7,6,5,4,3,2,1]
output3  = [1,2,3,4,5,6,7,8,9,10]


class   InsertionTestCase(unittest.TestCase):
    def test_insertionSort(self):

        InsertionSort(input1)
        InsertionSort(input2)
        InsertionSort(input3)
        self.assertEqual(input1,output1)
        self.assertEqual(input2,output2)
        self.assertEqual(input3,output3)


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

## Time complexity (Merge Sort and Insertion Sort)

The time complexity for merge sort and insertion sort where the sample size is 100 is given below:

### Source Code:

```
import time
import random

from matplotlib import pyplot as plt

from MergeSort import MergeSort
from InsertionSort import InsertionSort

def generate_random_list(size):
    return [random.choice(range(size)) for i in range(size)]


def calculate_time(func):
    """Decorator function for calculating time"""

    def inner(*args, **kwargs):

        tic = time.time_ns()
        func(*args, **kwargs)
        toc = time.time_ns()

        return toc - tic

    return inner


@calculate_time
def check_time_insertion_sort(arr):
    return InsertionSort(arr)


@calculate_time
def check_time_merge_sort(arr):
    r = len(arr)
    return MergeSort(arr,0,r)


if __name__ == "__main__":
    samples = [generate_random_list(i) for i in range(0, 1000, 10)]

    sample_sizes = []
    insertion_sort_times = []
```

```python
merge_sort_times = []

for sample in samples:
    sample_sizes.append(len(sample))
    insertion_sort_times.append(check_time_insertion_sort(sample))
    merge_sort_times.append(check_time_merge_sort(sample))

# Plotting
plt.figure(figsize=(10, 6))
plt.xlabel("Sample Size (n)")
plt.ylabel("Time Elasped (ns)")


plt.title("Time Complexity: Insertion sort vs Merge Sort")
plt.plot(sample_sizes, insertion_sort_times, ",-", label="Insertion Sort")
plt.plot(sample_sizes, merge_sort_times, ",-", label="Merge Sort")

plt.legend()
plt.show()
print("done")
```

Output:

The graph for the time complexity is given as:

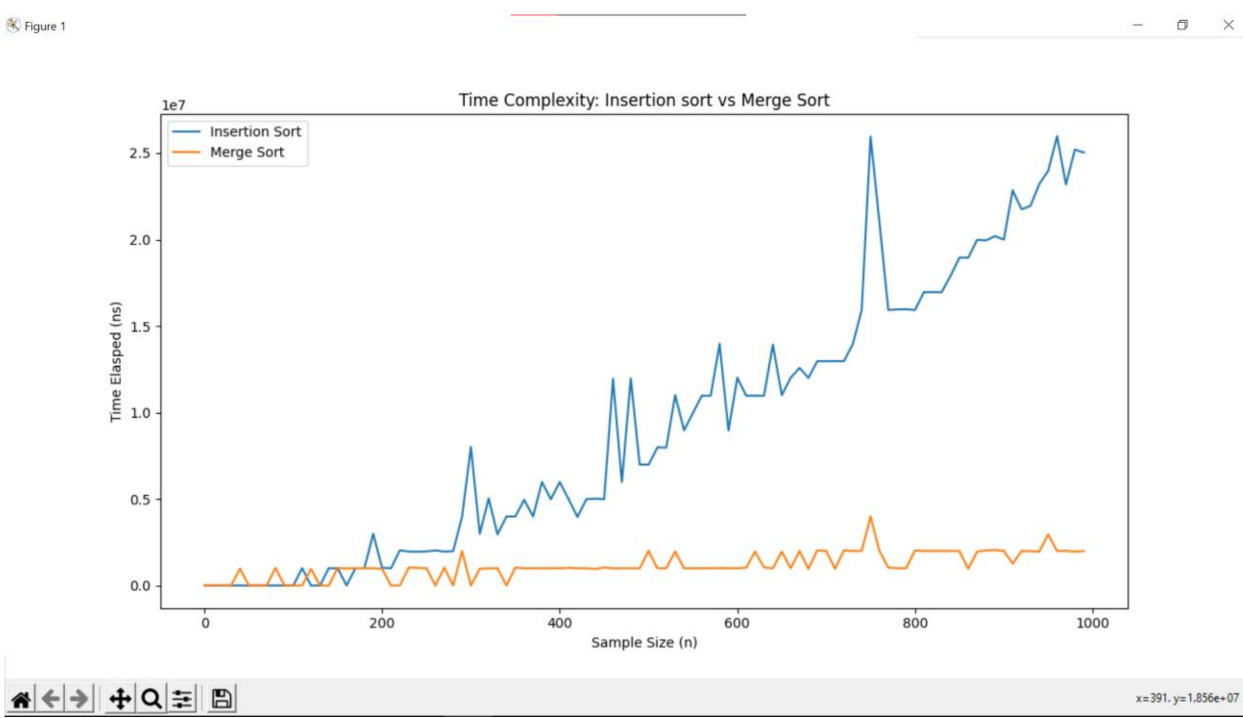


*Figure 3: Time Complexity for Merge Sort and Insertion Sort*