Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Algorithm and Complexity (COMP 314)

Lab 2 Report

Submitted to:

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submitted By:

Mani Dumaru

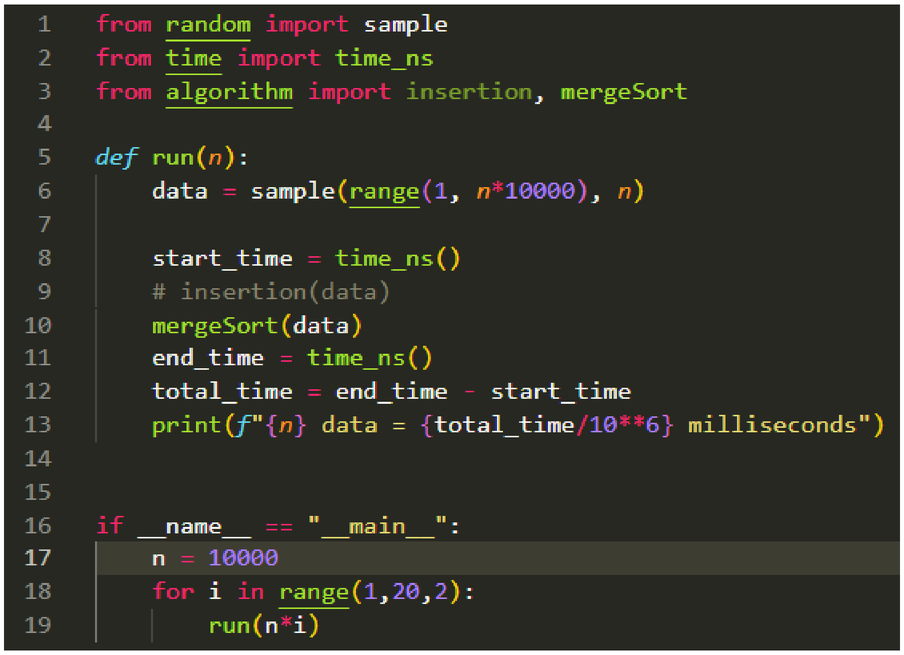
Roll no.: 15

CE-2019 3rd year/2nd semester

Submission Date: 25th April, 2023

Implementation, Testing and Performance of Insertion and Merge sort.

Code for performance testing:



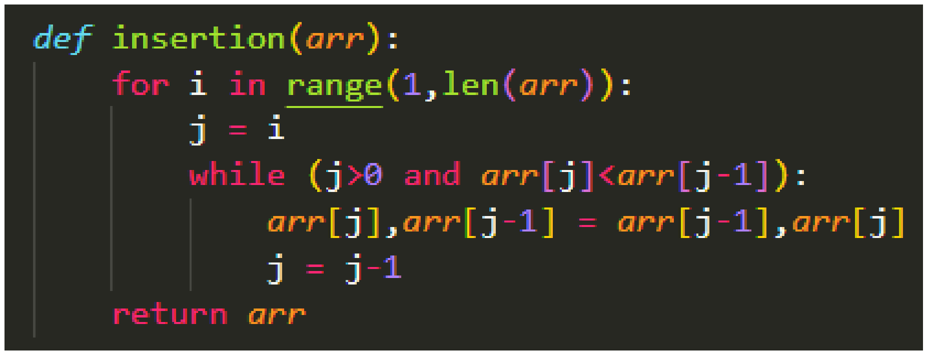
```python
1  from random import sample
2  from time import time_ns
3  from algorithm import insertion, mergeSort
4
5  def run(n):
6      data = sample(range(1, n*10000), n)
7
8      start_time = time_ns()
9      # insertion(data)
10     mergeSort(data)
11     end_time = time_ns()
12     total_time = end_time - start_time
13     print(f"{n} data = {total_time/10**6} milliseconds")
14
15
16 if __name__ == "__main__":
17     n = 10000
18     for i in range(1,20,2):
19         run(n*i)
```

'n' number of random data is generated in line 6 of above code. A timer is started in line 8 and then the randomly generated array is then passed for sorting to respective sorting algorithms (insertion and merge). When completion of sorting, the timer is then stopped. Total amount of time taken to sort 'n' number of data is then calculated as the end_time – start_time and is printed in milliseconds.

1. Insertion Sorting:

In this algorithm, the array is divided into two parts, sorted and unsorted. The key element of unsorted array is then placed into the correct position of the sorted part of the array.

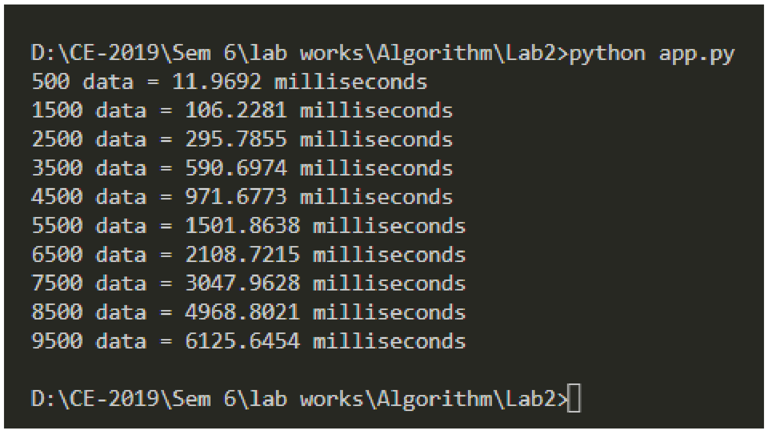
Algorithm:



```
def insertion(arr):
    for i in range(1,len(arr)):
        j = i
        while (j>0 and arr[j]<arr[j-1]):
            arr[j],arr[j-1] = arr[j-1],arr[j]
            j = j-1
    return arr
```

Algorithm for insertion Sorting

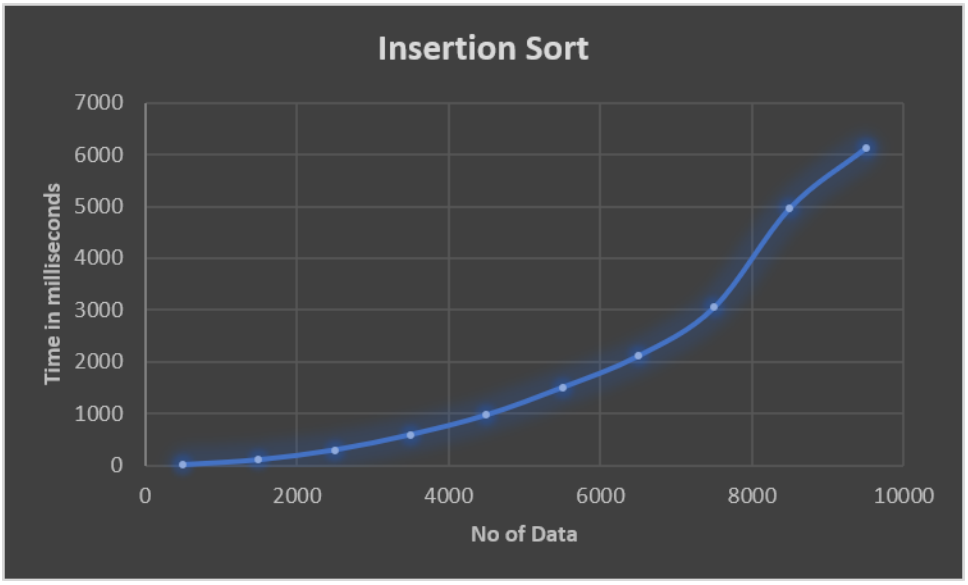
Performance:



```
D:\CE-2019\Sem 6\lab works\Algorithm\Lab2>python app.py
500 data = 11.9692 milliseconds
1500 data = 106.2281 milliseconds
2500 data = 295.7855 milliseconds
3500 data = 590.6974 milliseconds
4500 data = 971.6773 milliseconds
5500 data = 1501.8638 milliseconds
6500 data = 2108.7215 milliseconds
7500 data = 3047.9628 milliseconds
8500 data = 4968.8021 milliseconds
9500 data = 6125.6454 milliseconds

D:\CE-2019\Sem 6\lab works\Algorithm\Lab2>
```

Graph for Insertion sort:

With the data given by performance testing, following graph is plotted.

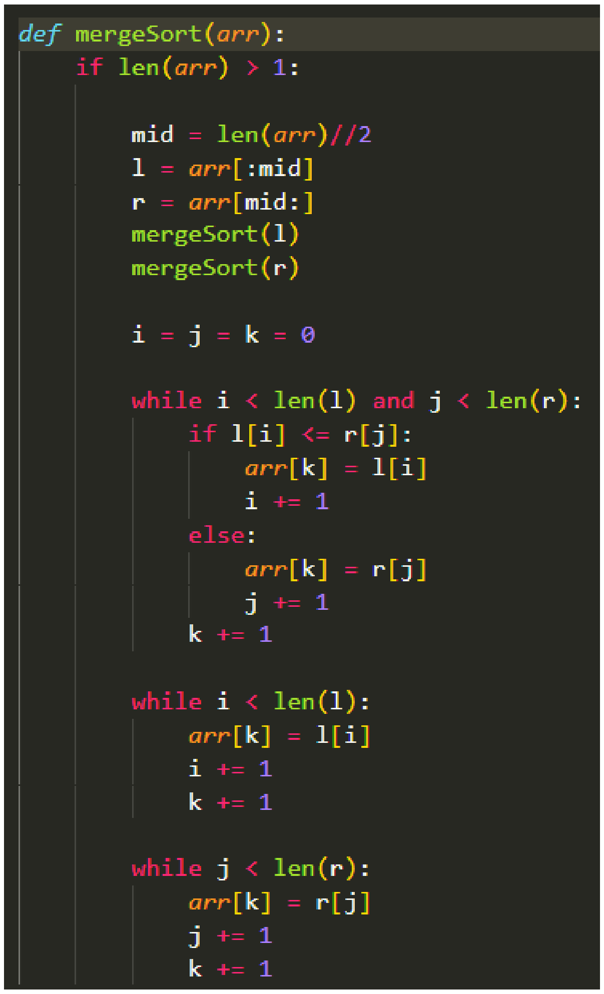


From the Graph, it is clear that the time complexity for insertion sorting algorithm is $O(n^2)$

2. Merge Sorting

This algorithm of sorting data is based on divide and conquer approach. An array is divided into n sub arrays where n is the length of array so that each sub array contains only one element. Then the divided sub arrays are merged back together in sorted order. Such algorithm is called merge sorting.

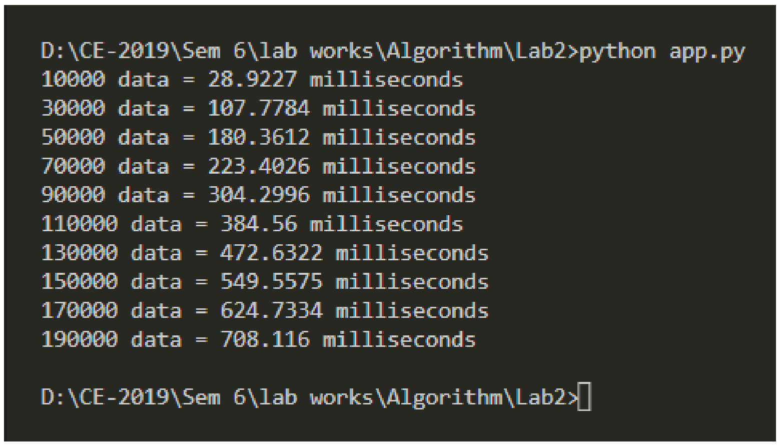
Algorithm:



```python
def mergeSort(arr):
    if len(arr) > 1:

        mid = len(arr)//2
        l = arr[:mid]
        r = arr[mid:]
        mergeSort(l)
        mergeSort(r)

        i = j = k = 0

        while i < len(l) and j < len(r):
            if l[i] <= r[j]:
                arr[k] = l[i]
                i += 1
            else:
                arr[k] = r[j]
                j += 1
            k += 1

        while i < len(l):
            arr[k] = l[i]
            i += 1
            k += 1

        while j < len(r):
            arr[k] = r[j]
            j += 1
            k += 1
```

Algorithm for Merge Sort

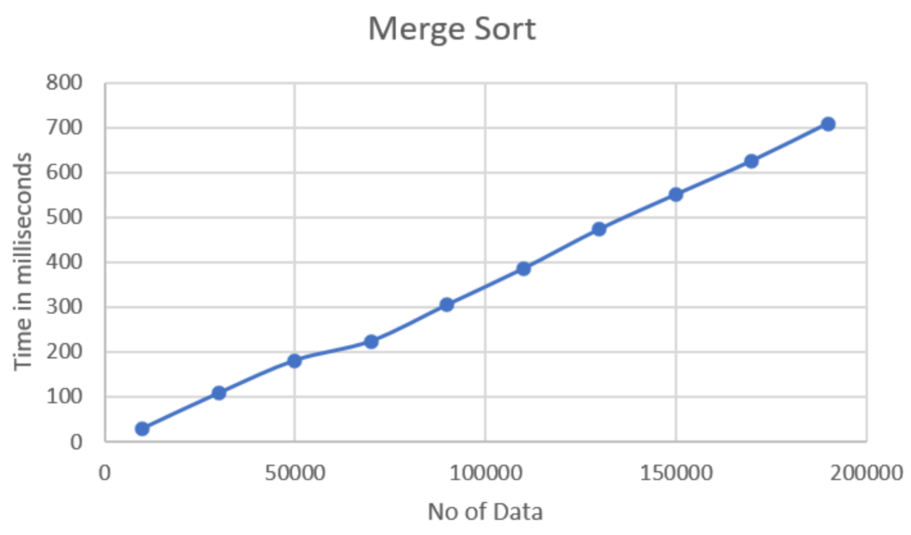
Performance:



```
D:\CE-2019\Sem 6\lab works\Algorithm\Lab2>python app.py
10000 data = 28.9227 milliseconds
30000 data = 107.7784 milliseconds
50000 data = 180.3612 milliseconds
70000 data = 223.4026 milliseconds
90000 data = 304.2996 milliseconds
110000 data = 384.56 milliseconds
130000 data = 472.6322 milliseconds
150000 data = 549.5575 milliseconds
170000 data = 624.7334 milliseconds
190000 data = 708.116 milliseconds

D:\CE-2019\Sem 6\lab works\Algorithm\Lab2>
```

Graph for Merge Sort:

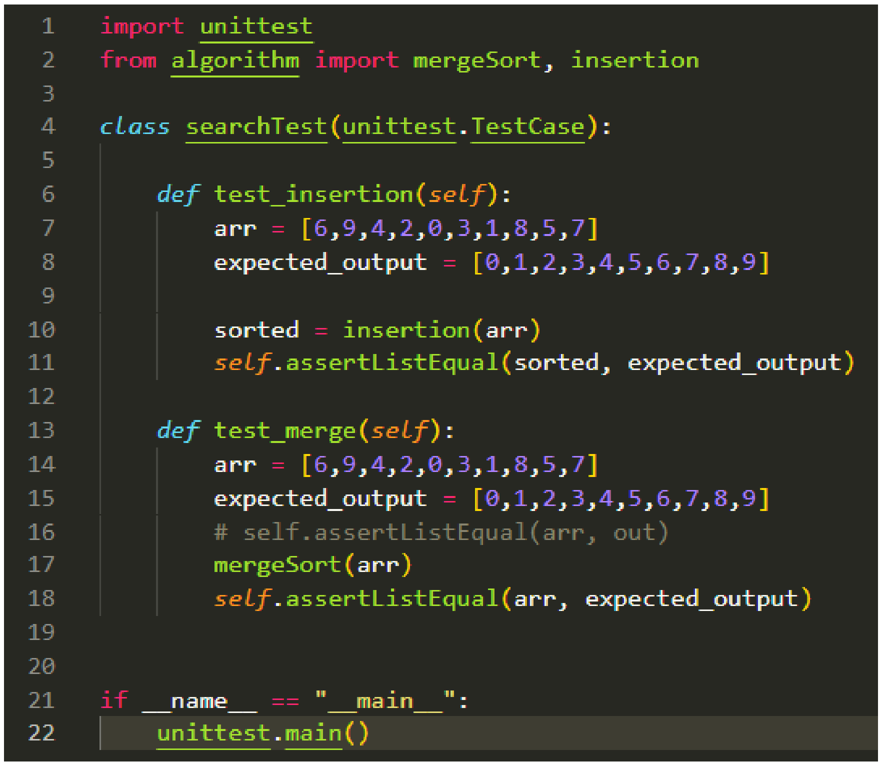
With the data given by performance test on merge sort, following graph is plotted.



From this graph, we can see that the time of complexity of Merge sort seems to be O(nlogn)

Testing the algorithm:

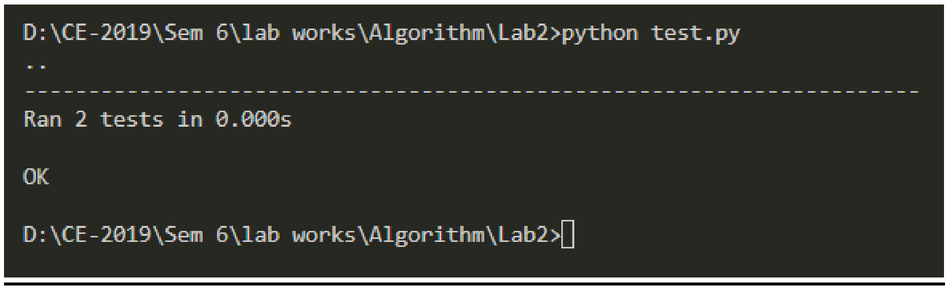
Code for testing insertion and merge sorting algorithm.



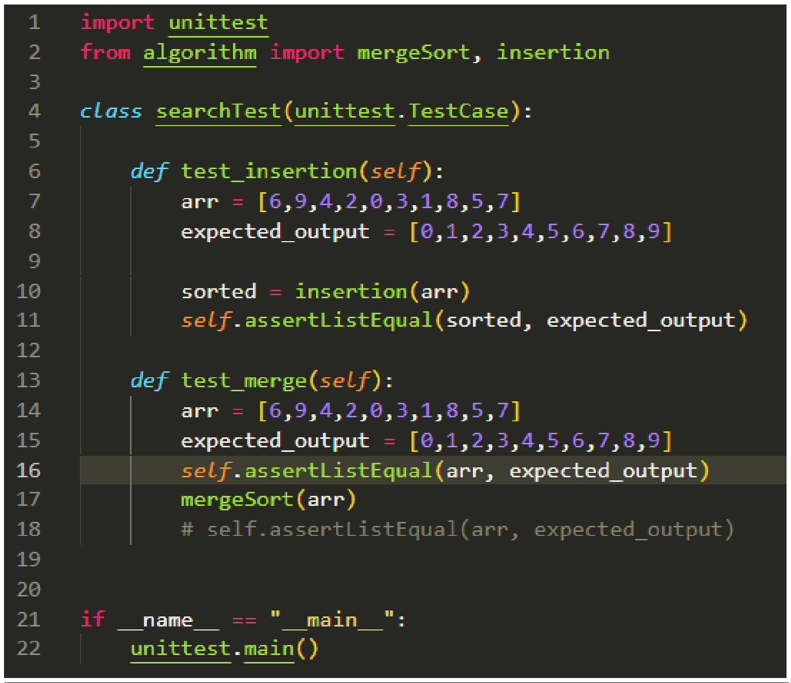
```python
import unittest
from algorithm import mergeSort, insertion

class searchTest(unittest.TestCase):

    def test_insertion(self):
        arr = [6,9,4,2,0,3,1,8,5,7]
        expected_output = [0,1,2,3,4,5,6,7,8,9]

        sorted = insertion(arr)
        self.assertListEqual(sorted, expected_output)

    def test_merge(self):
        arr = [6,9,4,2,0,3,1,8,5,7]
        expected_output = [0,1,2,3,4,5,6,7,8,9]
        # self.assertListEqual(arr, out)
        mergeSort(arr)
        self.assertListEqual(arr, expected_output)


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

A random array along with another array with its sorted result is created. The random array is then passed to the sorting algorithm. After it has been sorted by the algorithm, it is checked if the result provided by the algorithm matches the expected output. If they match, the test is successful otherwise, the test is failed.

Successful Test Output:



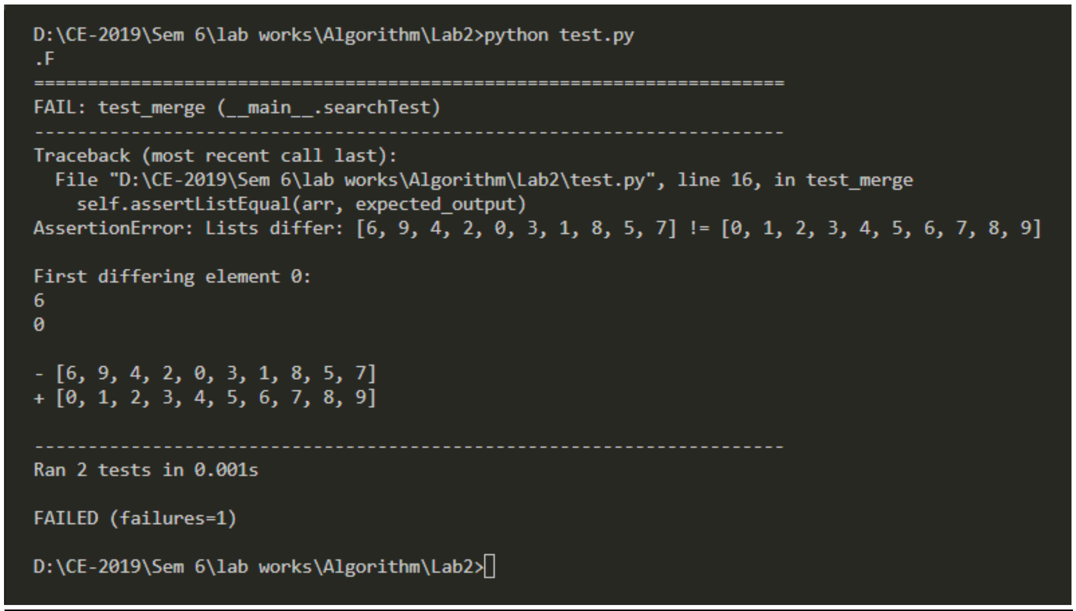
```
D:\CE-2019\Sem 6\lab works\Algorithm\Lab2>python test.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK

D:\CE-2019\Sem 6\lab works\Algorithm\Lab2>
```



```python
1  import unittest
2  from algorithm import mergeSort, insertion
3
4  class searchTest(unittest.TestCase):
5
6      def test_insertion(self):
7          arr = [6,9,4,2,0,3,1,8,5,7]
8          expected_output = [0,1,2,3,4,5,6,7,8,9]
9
10         sorted = insertion(arr)
11         self.assertListEqual(sorted, expected_output)
12
13     def test_merge(self):
14         arr = [6,9,4,2,0,3,1,8,5,7]
15         expected_output = [0,1,2,3,4,5,6,7,8,9]
16         self.assertListEqual(arr, expected_output)
17         mergeSort(arr)
18         # self.assertListEqual(arr, expected_output)
19
20
21 if __name__ == "__main__":
22     unittest.main()
```

In this test, in line number 16, a test is done before the array is passed to the sorting algorithm. This test should fail since, the array and expected result does not match.

Failed Test Output:



```
D:\CE-2019\Sem 6\lab works\Algorithm\Lab2>python test.py
.F
======================================================================
FAIL: test_merge (__main__.searchTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "D:\CE-2019\Sem 6\lab works\Algorithm\Lab2\test.py", line 16, in test_merge
    self.assertListEqual(arr, expected_output)
AssertionError: Lists differ: [6, 9, 4, 2, 0, 3, 1, 8, 5, 7] != [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

First differing element 0:
6
0

- [6, 9, 4, 2, 0, 3, 1, 8, 5, 7]
+ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (failures=1)

D:\CE-2019\Sem 6\lab works\Algorithm\Lab2>
```

Conclusion:

Hence, Insertion and Merge sort algorithms were implemented. We checked if the algorithms were correct. Also, their performance was tested based on how much time they took to sort ‘n’ amount of data. With the results we got, merge sort algorithm is clearly a better algorithm for sorting as it took way lesser time to sort huge number of data than insertion sorting did. Time complexity for insertion sort was found to be O(n^2) while that of merge of sort was found to be O(nlogn).