INFSCI 2591 Algorithm Design Project: 4

Submitted By: Nidhi Agarwal

Solution #1

a. Pseudo Code:

Function: Comparison Count Sort ()
Input: a list of integers, which are not sorted
Output: a sorted list of integers

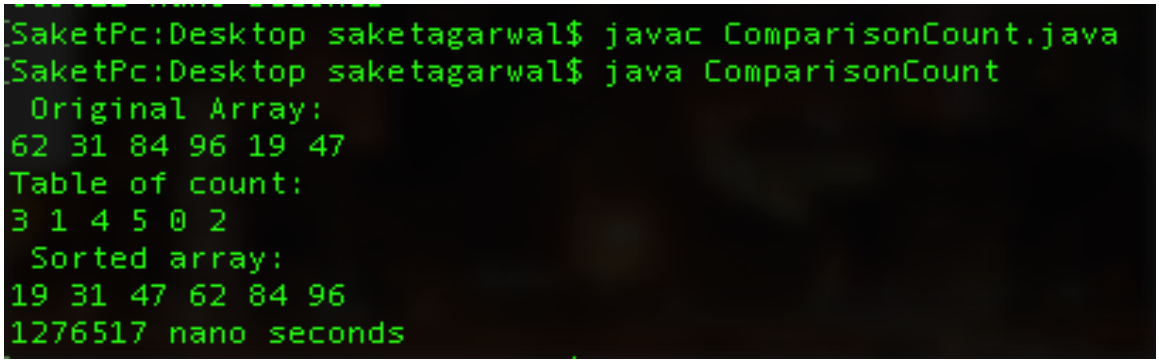
```
Int array [ ]                    // takes the input
Int count [ ]    //stores number of elements smaller than i-th element
Temp_counter [ ]
//maps the elements from array[] according to //number of elements
smaller than i-th element, stored in count[]  to //result[]
result [ ]         //stores sorted array


For (I =0; I < size_of_array; i++)     //loop to read all elements
of the array
Set result[i] = min_value

For ( i =0 ; i < size_of_array ; i++ )
For ( j=0; i < size_of_array ;  i++ )
if (array[i] > array[j])              // compare two neighboring elements
count[i]++
```

The array temp_counter helps in mapping the smallest element with the array count[]. i.e. to find the elements according to their count, and then append the element to result [ ].

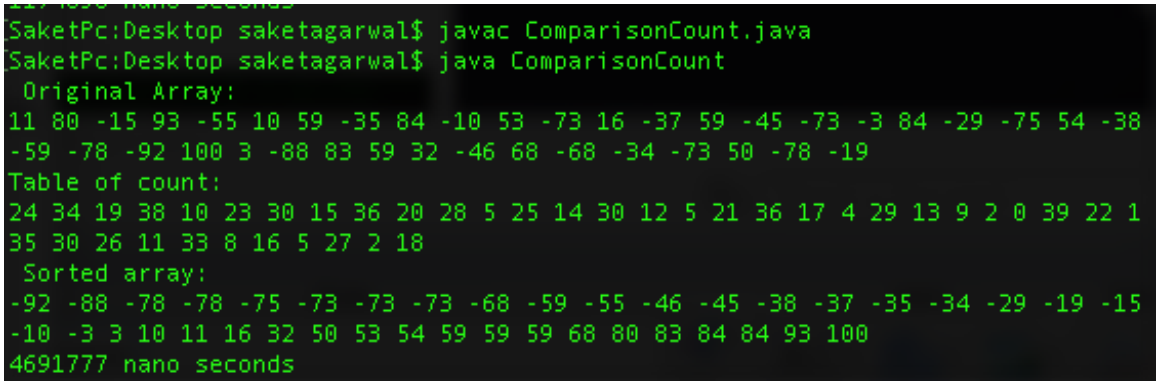
b. Source Code:
Attached with the report

Screen-shots:

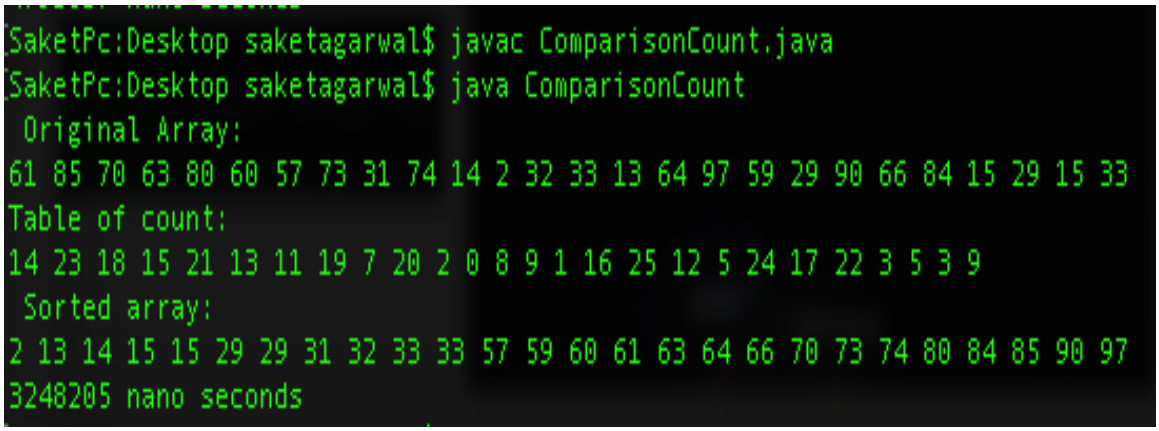
1. The following shows the implementation of test-case 1



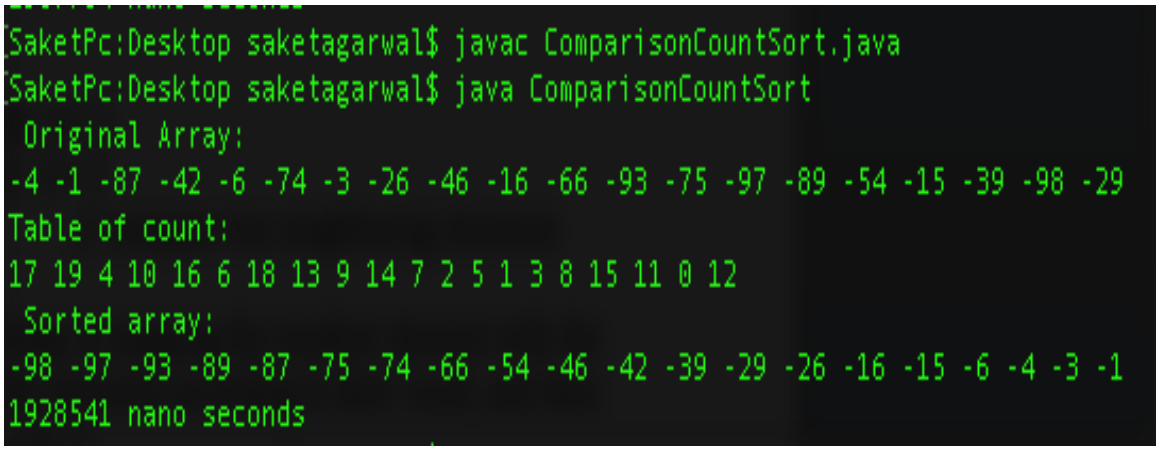
2. The following shows the implementation of test-case 2



3. The following shows the implementation of test-case 3



4. The following shows the implementation of test-case 4



All the 4 test- cases are implemented successfully.

c. Time Complexity:

c.1 Theoretical –

If n is the number of elements, we express time complexity here in terms of n. The first loop runs n times. The second loops runs n times twice, so it takes n^2 iterations.
The fourth loop runs n times.

So, the total time: $n + n^2 + n = n(n+2) \sim n^2$

c.2 Experimental –

The test- cases are each implemented five times. The time taken is shown in table-1. The times are in Nano-seconds. :

	I Test Case	II Test Case	III Test Case	IV Test Case
	6 elements	40 elements	26 elements	20 elements
	1442232	4691777	3248205	2723313
	859012	4243746	4195370	1857530
	1276517	3631361	2681604	1967087
	1180656	3738063	2286534	3561790
	1856098	5156160	2421666	3036555
Average Time(nano sec)	1322903	4292221.4	2966675.8	2629255

Table-1

In figure-1, we plot these times in a chart:

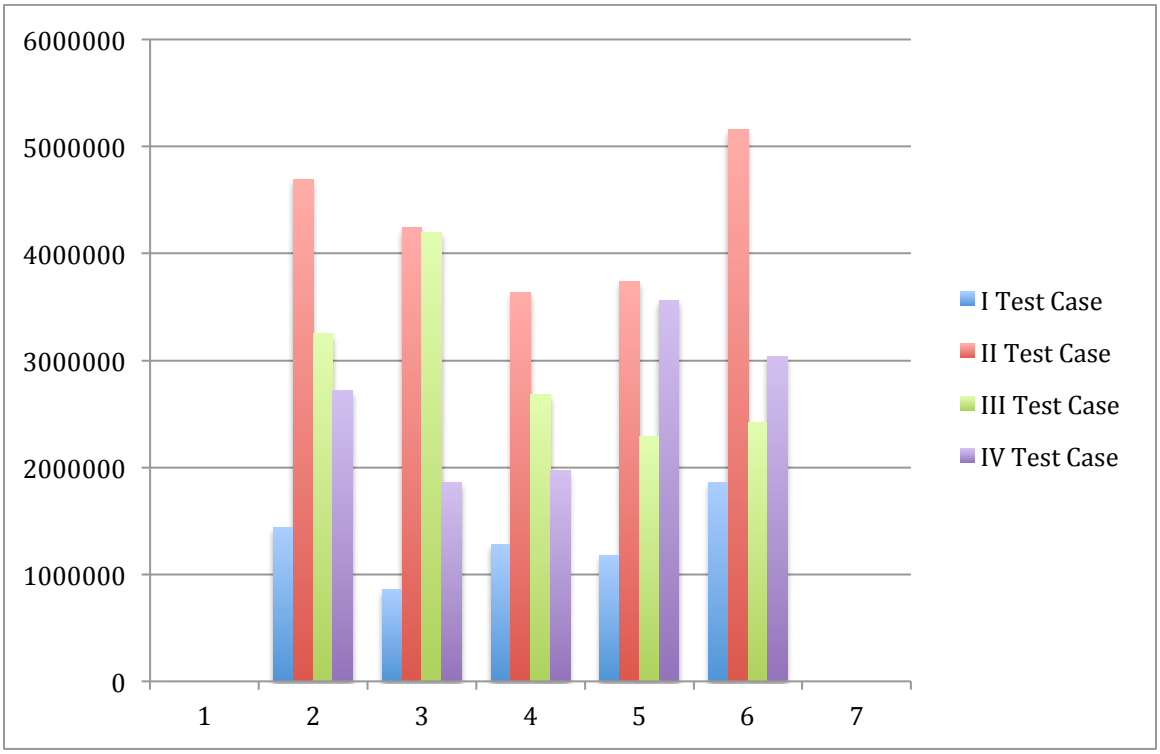


Figure-1

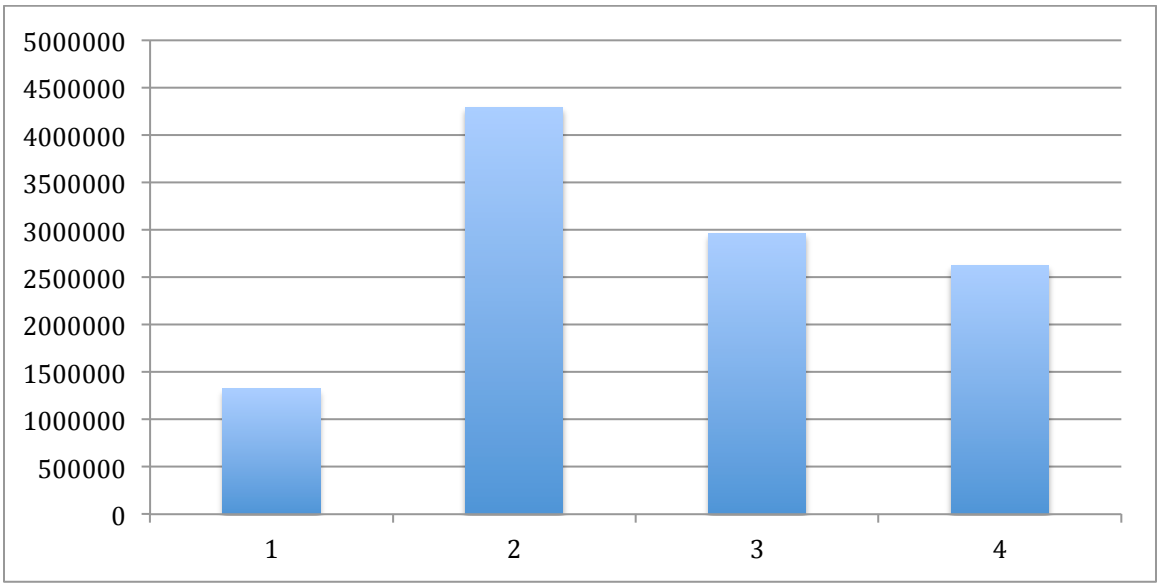


Figure-2

Figure-2 shows the graph of average time taken by the 4 test-cases

By using the graphs, we found out that the time taken by test-cases is directly proportional to the number of elements in that array.

Test-case 1 has only 6 elements, it takes minimum time, whereas test-case 2 has 40 (maximum) elements, it takes maximum time too.

Ideally,test case 4 should take lesser time than test case 3, but in 4^{th} and 5^{th} iteration , it took more time (refer figure -2).
This can be because they have similar number of elements(26 and 20 respectively).

Solution #2

Part-1

Create a mirror of a binary tree.

a) Pseudo-code:

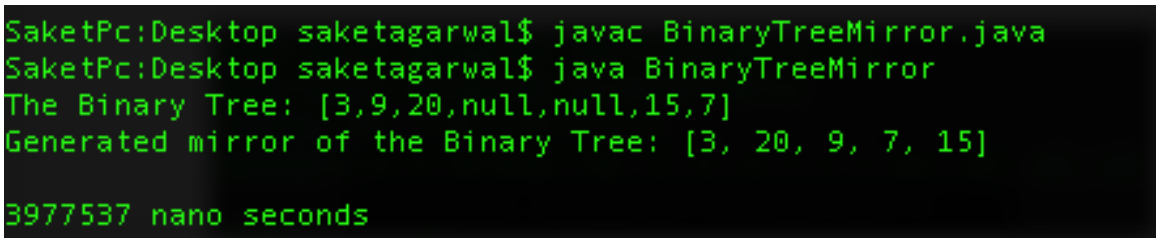
```
Function CreateBinaryTreeMirror()
Input : a binary tree
Output: its mirror tree

String tree[ ]
Hashtree = new hashmap
Int start_index=0 , height = 0 , n , parent
Boll flag=true

String [ ] split_tree
N=2*power[ height ]

 for (int i = 0; i < n; i++)
compare start_index and split_tree.length
if
start_index > split_tree.length
tree.add=”null”
else
        if
                height==0
                tree.add(split_tree[start_index])
        else
                parent= I / 2
                if
                        parent of height-1== null
                        tree.add(“null”)
                else
                        tree.add(split_tree[start_index])


start_index ++
```

```
hashtree.put(height,tree)
height++

for (int k = 0; k < Hashtree.size(); k++)
test_tree = hashtree.get ( k)
num=test_tree.size / 2
j=num-1

string tmp

if ( num ! = 0 )
        for (int i = num; i < testree.size(); i++)
        test_tree.set(j,tmp)
j- -

prune the tree
for (int k = Hashtree.size()-1; k >= 1 ; k--)
for (int i = size-1; i >= 0; i = i-2)
parent= I / 2

check k-1 value of hash_tree
→ find the parent
→ if null : remove ith and ( I - 1)th element from hashtree

check kth value of hash_tree
→ if null : remove ith and ( I - 1)th element from hashtree

output generation :
for (int i = 0; i < Hashtree.size(); i++)
mirrorTree.add(Hashtree.get(i))
```

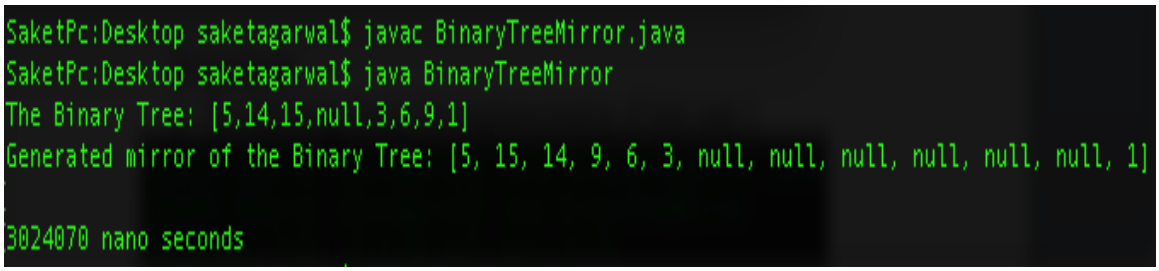
b) Source code attached with the report.

Screenshots:

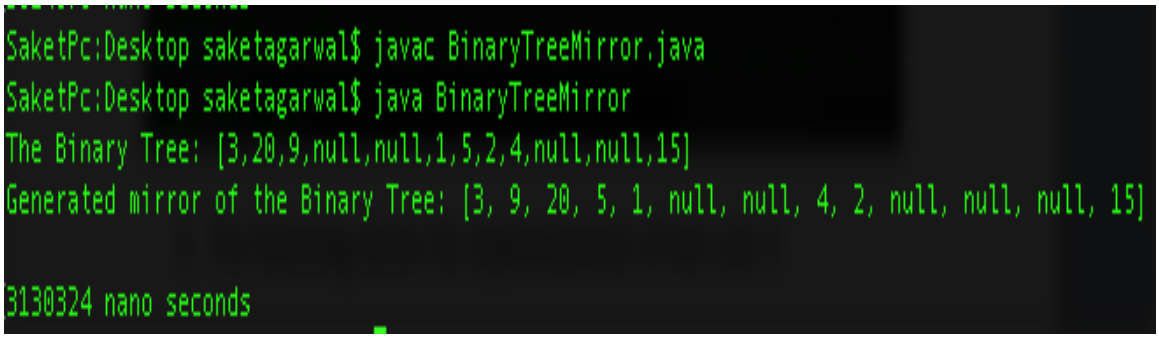
1. The following shows the implementation of test-case 1



2. The following shows the implementation of test-case 2



3. The following shows the implementation of test-case 3



c. Time Complexity:

c.1 Theoretical –
It has a total of 6 for-loops. 4 are single and one – loop is inside another one. The total number of iterations for n elements:
$4n + n^2 = n(n+4) \sim n^2$

c.2 Experimental –

The test- cases are each implemented five times. The time taken is shown in table-2. The times are in nano-seconds. :

	test case 1	test case 2	test case 3
1	3977537	3024070	3130324
2	3712111	2851639	3679201
3	3234584	2820674	3056486
4	2711371	3403582	3124291
5	3117997	3093211	2850072
Average time	3350720	3038635.2	3168074.8

Table-2

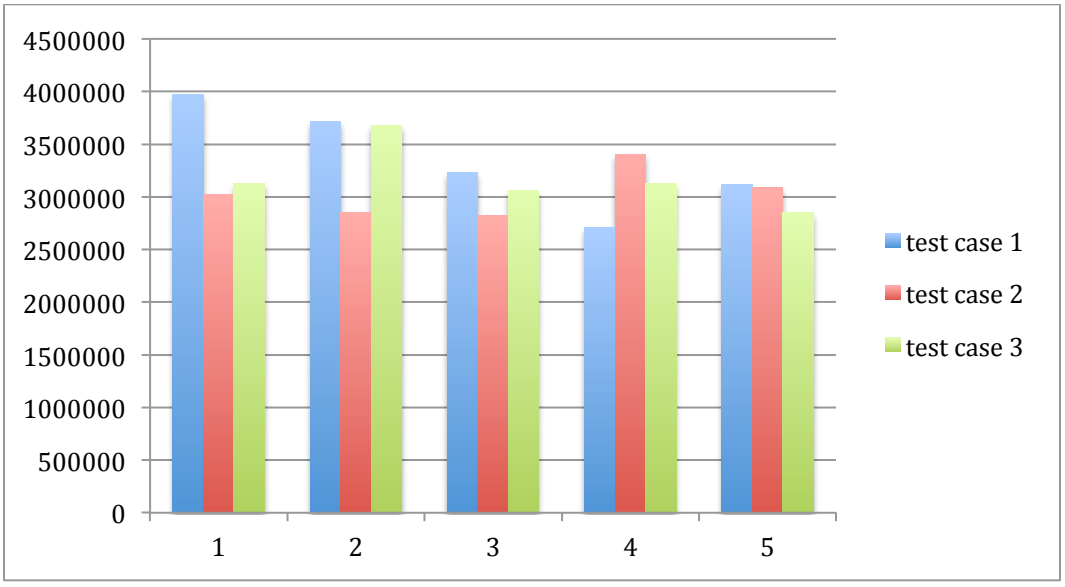


Figure-3

Figure-3 shows the graph plotted using values from table-2

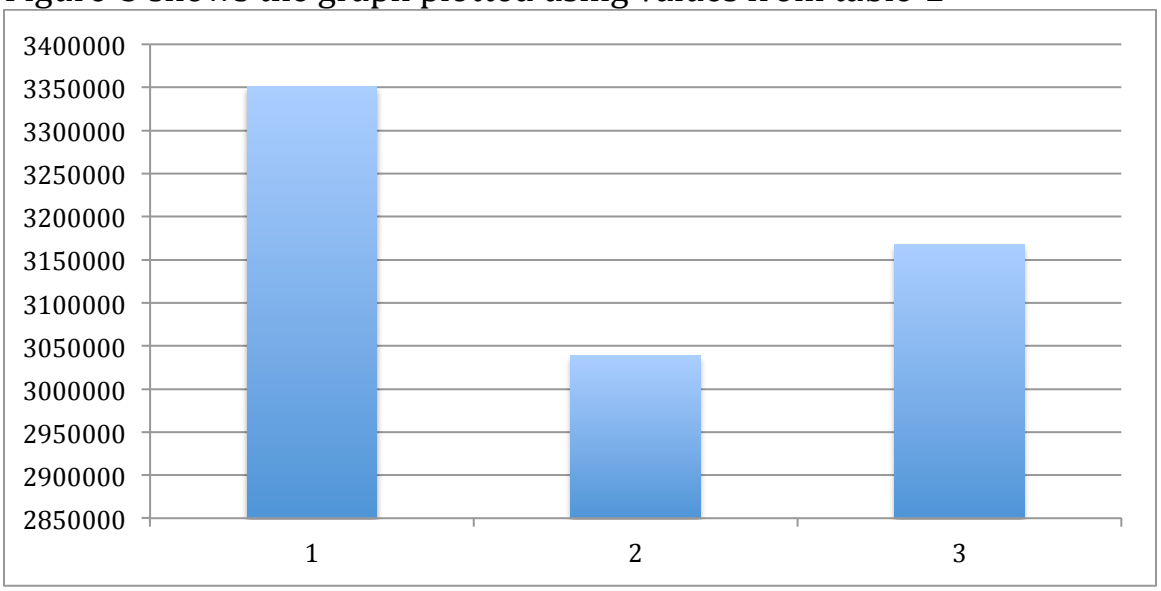


Figure-4

The first test case has the minimum number of elements; still it mostly takes the maximum time.

Test case 1 → minimum elements→Maximum time

Ideally this test case should have taken the minimum time. The reasons can be: the time taken is random and doesn't really change up to a number of elements.
My computer system might have some other application running and taking more system resources.
Due to which the implementation took such long time (compared to the other 2 test- cases).

Solution #2

Part-2

Compare if two binary trees are mirror of each other

a. Pseudo code

```
Function CompareBinary_treeMirror ()
Input : 2 binary trees
Output: comparison, If the two trees are mirror of each other or not

String tree[ ]
String mirrortree [ ]
Hashtree = new hashmap
Int start_index=0 , height = 0 , n , parent
Boll flag=true

String [ ] split_tree
N=2*power[ height ]

   for (int i = 0; i < n; i++)
  compare start_index and split_tree.length
if
start_index > split_tree.length
tree.add=”null”
else
        if
                height==0
                tree.add(split_tree[start_index])
        else
                parent= I / 2
                if
                        parent of height-1== null
                        tree.add(“null”)
                else
                        tree.add(split_tree[start_index])
start_index ++
hashtree.put(height,tree)
height++
```

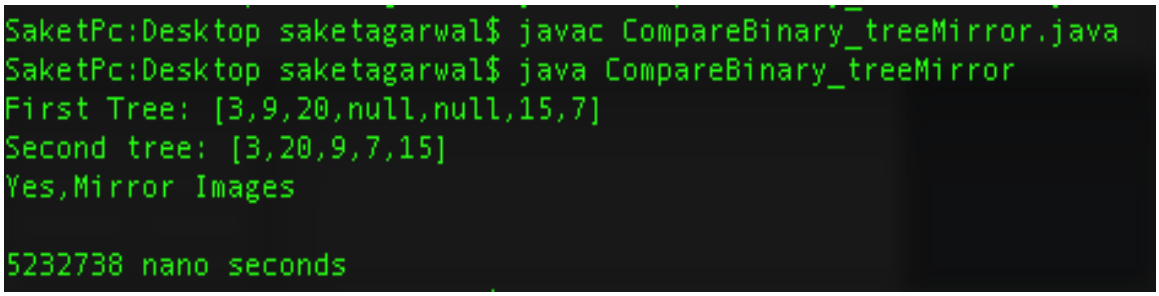
```
for (int k = 0; k < Hashtree.size(); k++)
test_tree = hashtree.get ( k)
num=test_tree.size / 2
j=num-1

string tmp

if ( num ! = 0 )
        for (int i = num; i < testree.size(); i++)
        test_tree.set(j,tmp)
j- -

prune the tree
for (int k = Hashtree.size()-1; k >= 1 ; k--)
for (int i = size-1; i >= 0; i = i-2)
parent= I / 2

check k-1 value of hash_tree
→ find the parent
→ if null : remove ith and ( I - 1)th element from hashtree

check kth value of hash_tree
→ if null : remove ith and ( I - 1)th element from hashtree

output generation :
for (int i = 0; i < Hashtree.size(); i++)
mirrorTree.add(Hashtree.get(i))
String result = mirrorTree.toString();
if (result.equalsIgnoreCase(mirrortree))
        return true;
      else
        return false;
```

We compare the mirror of first tree generated , with the given second tree, if they are same(equal), the function returns true, otherwise false.

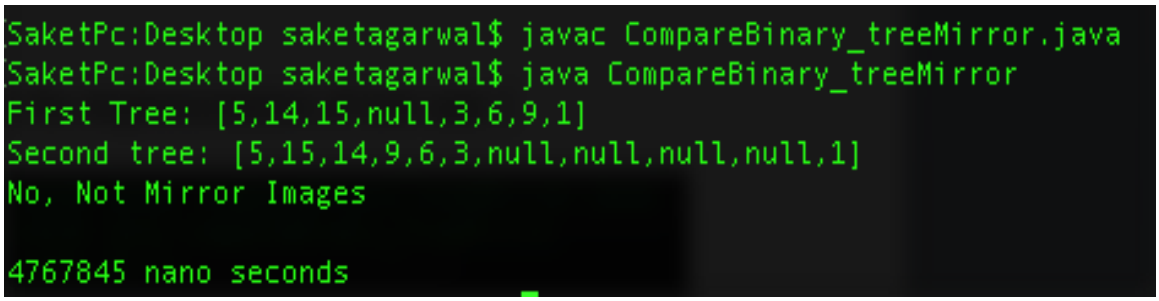
b. Source code attached with report

Screen-shots:

1. The following shows the implementation of test-case 1



2. The following shows the implementation of test-case 2



3. The following shows the implementation of test-case 3

c. Time Complexity:

c.1 Theoretical –
It has 5 loops and a loop inside a loop.
Therefore → $5n + n^2 = n(n+5) \sim n^2$

c.2 Experimental –

The test- cases are each implemented five times. The time taken is shown in table-3. The times are in Nano-seconds. :

	Test case 1	Test case 2	Test case 3
	5064407	4767845	5435419
	3717623	3441122	20817289
	3260185	3810254	3458042
	3326434	5145455	4186300
	3402648	3411004	5041183
Average Time	3754259.4	4115136	7787646.6

Table-3

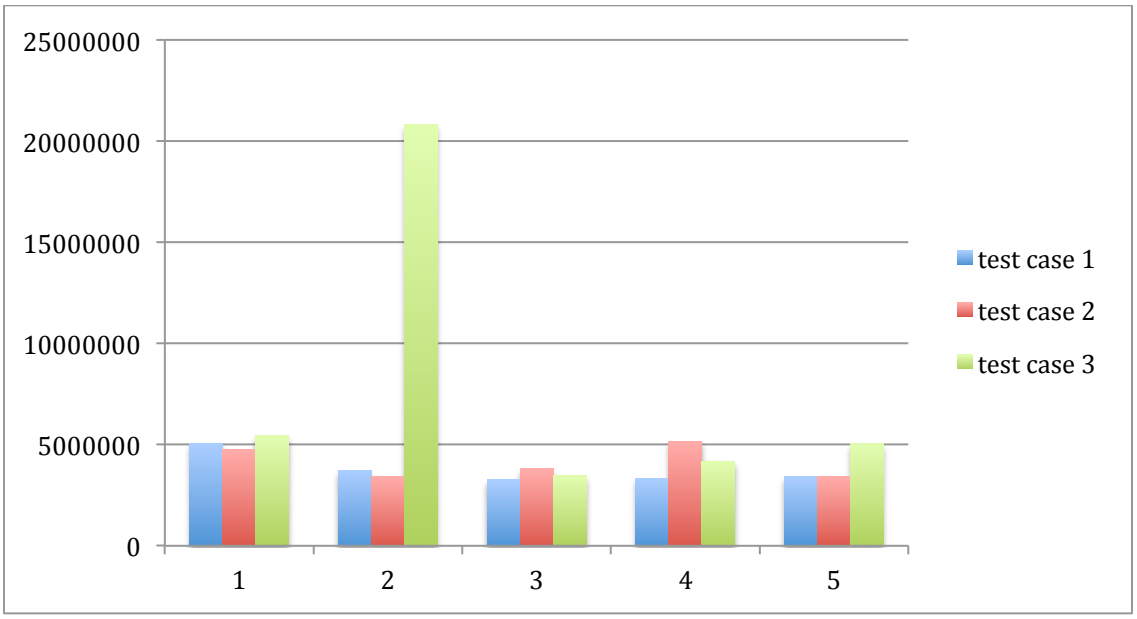


Figure-5

Figure-5 shows the graph plotted using values from table-3

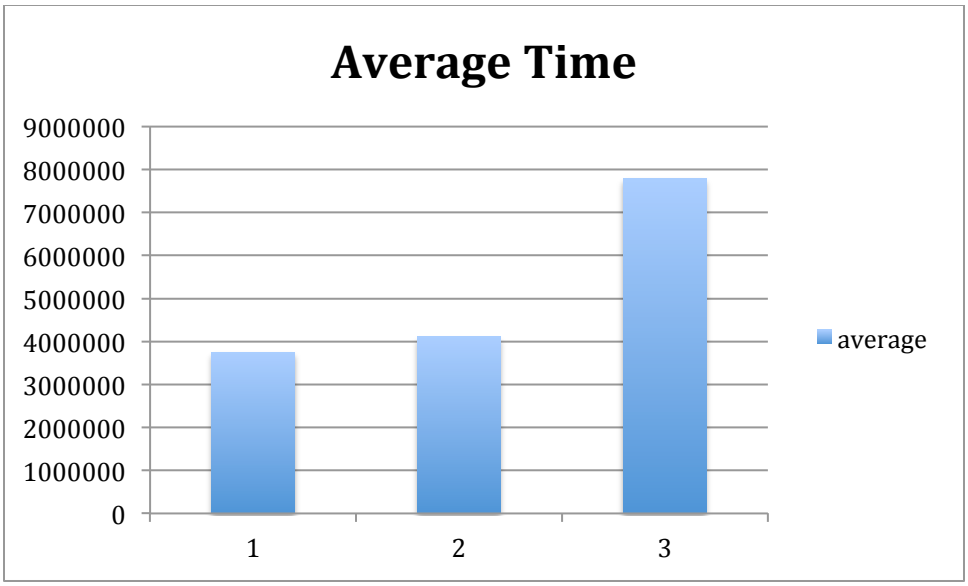


Figure-6

The first test case has the minimum number of elements; and it takes minimum time.

Test case 1 → minimum elements→ minimum time

Through figure-5, we can see that the three iterations take similar times. It can be inferred, that up to a number of elements, the system took similar time to compute if the two trees are mirror of each other or not.

Summary:
Successfully implemented all the algorithms, the test-cases are shown using screen-shots. The theoretic and experimental time complexities are calculated and discussed in this report.

References:

1. https://en.wikipedia.org/wiki/Binary_tree
2. www.geeksforgeeks.com
3. www.quora.com
4. www.stackoverlow.com