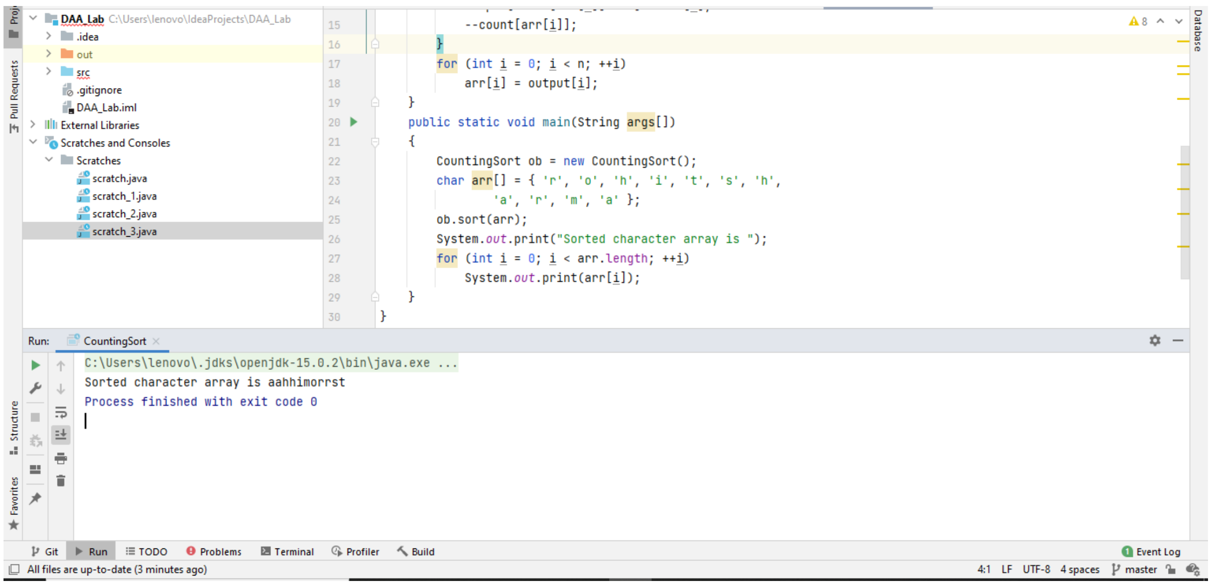
<u>Design And Analysis Of Algorithms</u>
<u>Practical</u>

<u>Objective:-</u> Implement and analyze the complexity of Counting Sort.

Code:-

```java
class CountingSort {
   void sort(char arr[])
   {
       int n = arr.length;
       char output[] = new char[n];
       int count[] = new int[256];
       for (int i = 0; i < 256; ++i)
           count[i] = 0;
       for (int i = 0; i < n; ++i)
           ++count[arr[i]];
       for (int i = 1; i <= 255; ++i)
           count[i] += count[i - 1];
       for (int i = n - 1; i >= 0; i--) {
           output[count[arr[i]] - 1] = arr[i];
           --count[arr[i]];
       }
       for (int i = 0; i < n; ++i)
           arr[i] = output[i];
   }
   public static void main(String args[])
   {
       CountingSort ob = new CountingSort();
       char arr[] = { 'r', 'o', 'h', 'i', 't', 's', 'h',
                'a', 'r', 'm', 'a' };
       ob.sort(arr);
       System.out.print("Sorted character array is ");
       for (int i = 0; i < arr.length; ++i)
           System.out.print(arr[i]);
   }
}
```

Output:-



```java
            --count[arr[i]];
        }
        for (int i = 0; i < n; ++i)
            arr[i] = output[i];
    }
    public static void main(String args[])
    {
        CountingSort ob = new CountingSort();
        char arr[] = { 'r', 'o', 'h', 'i', 't', 's', 'h',
                'a', 'r', 'm', 'a' };
        ob.sort(arr);
        System.out.print("Sorted character array is ");
        for (int i = 0; i < arr.length; ++i)
            System.out.print(arr[i]);
    }
}
```

```
C:\Users\lenovo\.jdks\openjdk-15.0.2\bin\java.exe ...
Sorted character array is aahhimorrst
Process finished with exit code 0
```

Analyze:-The time complexity of counting sort algorithm is **O(n+k)** where n is the number of elements in the array and k is the range of the elements. Counting sort is most efficient if the range of input values is not greater than the number of values to be sorted.