Design And Analysis Of Algorithms

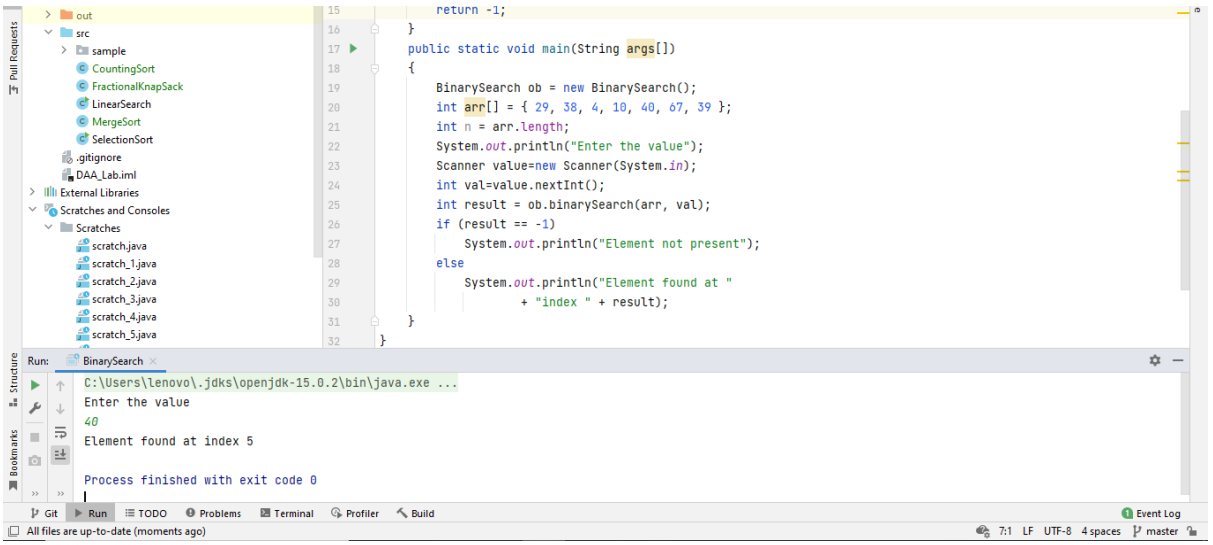
Practical

Objective:- Implement and analyze the complexity of Binary Search.

Code:-

```
import java.util.Scanner;
class BinarySearch {
   int binarySearch(int arr[], int val)
   {
       int left = 0, right = arr.length - 1;
       while (left <= right) {
           int mid = left + (right - left) / 2;
           if (arr[mid] == val)
               return mid+1;
           if (arr[mid] < val)
               left = mid + 1;
           else
               right = mid - 1;
       }
       return -1;
   }
   public static void main(String args[])
   {
       BinarySearch ob = new BinarySearch();
       int arr[] = { 29, 38, 4, 10, 40, 67, 39 };
       int n = arr.length;
       System.out.println("Enter the value");
       Scanner value=new Scanner(System.in);
       int val=value.nextInt();
       int result = ob.binarySearch(arr, val);
       if (result == -1)
           System.out.println("Element not present");
       else
           System.out.println("Element found at "
                   + "index " + result);
   }
}
```

Output:-



Analysis:-

The time complexity of the binary search algorithm is **O(log n)**. The best-case time complexity would be O(1) when the central index would directly match the desired value.

Objective:- Implement and analyze the complexity of Heap Sort.

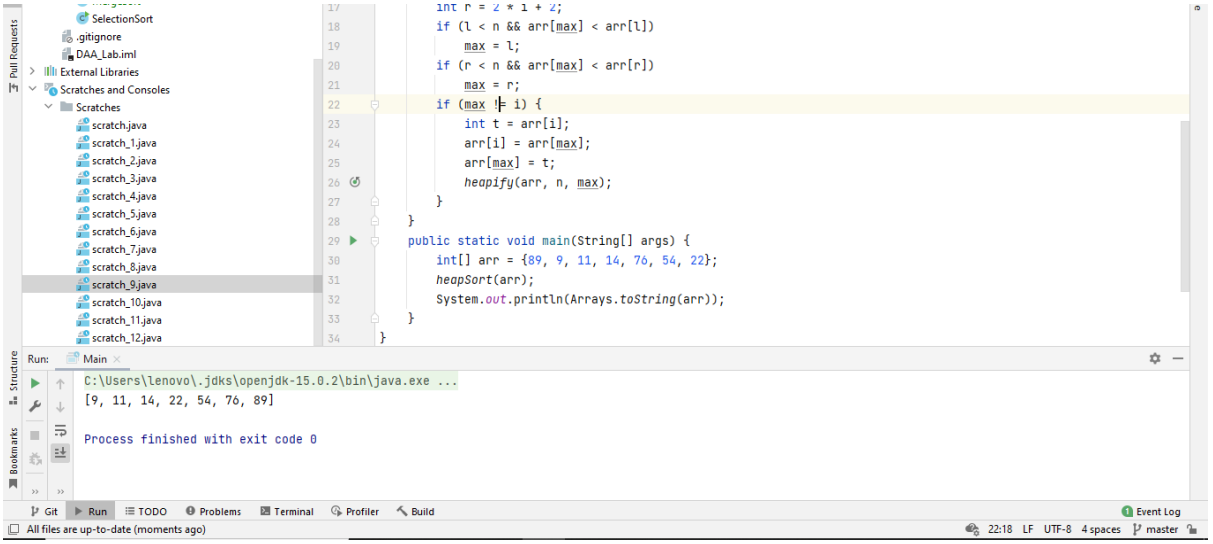
Code:-

```
import java.util.Arrays;
class Main {
   public static void heapSort(int[] arr) {
        for (int i = arr.length / 2 - 1;i >= 0;i--)
            heapify(arr, arr.length, i);

        for (int i = arr.length - 1;i >= 0;i--) {
            int t = arr[0];
            arr[0] = arr[i];
            arr[i] = t;
            heapify(arr, i, 0);
        }
    }
   public static void heapify(int[] arr, int n, int i) {
        int max = i;
        int l = 2 * i + 1;
        int r = 2 * i + 2;
        if (l < n && arr[max] < arr[l])
            max = l;
        if (r < n && arr[max] < arr[r])
            max = r;
        if (max != i) {
            int t = arr[i];
            arr[i] = arr[max];
            arr[max] = t;
            heapify(arr, n, max);
        }
    }
```

```
    public static void main(String[] args) {
        int[] arr = {89, 9, 11, 14, 76, 54, 22};
        heapSort(arr);
        System.out.println(Arrays.toString(arr));
    }
}
```

Output:-



Analysis:-

Heap sort is an in-place algorithm. Time Complexity: Time complexity of heapify is **O(Logn)**. Time complexity of createAndBuildHeap() is O(n) and the overall time complexity of Heap Sort is O(nLogn).