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) {</pre>
           int mid = left + (right - left) / 2;
           if (arr[mid] == val)
               return mid+1;
           if (arr[mid] < val)</pre>
               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");
           System.out.println("Element found at "
                  + "index " + result);
   }
```

Output:-

```
public static void main(String args[])
          CountingSort
          © FractionalKnapSack
                                                                  BinarySearch ob = new BinarySearch();
           LinearSearch
                                                                  int arr[] = { 29, 38, 4, 10, 40, 67, 39 };
          MergeSort
                                                                  int n = arr.length;
                                                                  System.out.println("Enter the value");
        agitignore.
                                                                  Scanner value=new Scanner(System.in);
        A DAA Lab.iml
                                                                  int result = ob.binarySearch(arr, val);
     Scratches and Consoles
                                                                  if (result == -1)
       Scratches

scratch.java
                                                                       System.out.println("Element not present");
                                                                  else
          ascratch 1.java
                                                                      System.out.println("Element found at "
           scratch_3.java
                                                                                + "index " + result):
          acratch 4,iava
Run: BinarySearch ×

C:\Users\lenovo\
Entry the value
           C:\Users\lenovo\.jdks\openjdk-15.0.2\bin\java.exe ...
22 □ □ □ 40 Element found at index 5
           Process finished with exit code \theta
   P Git ▶ Run ≔ TODO ● Problems ☑ Terminal ♠ Profiler 	 Suild
                                                                                                                                                       € 7:1 LF UTF-8 4 spaces 12 master 1
```

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 1 = 2 * i + 1;
       int r = 2 * i + 2;
       if (1 < n && arr[max] < arr[1])</pre>
           max = 1;
       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:-
                                                                int r = 2 * 1 + 2;
if (l < n && arr[max] < arr[l])
           © SelectionSort
         .gitignore
                                                                     <u>max</u> = l;
                                                                if (r < n && arr[max] < arr[r])

max = r;
    > IIII External Libraries
    Scratches and Consoles

Scratches
                                                                if (<u>max</u> != i) {
                                                                    int t = arr[i];
arr[i] = arr[<u>max</u>];
          🚅 scratch.java
           acratch 2.java
                                                                     arr[\underline{max}] = t;
           scratch_3.java
                                                                    heapify(arr, n, max);
           acratch_5.java
                                                28
29 ►
           scratch_6.java
                                                            public static void main(String[] args) {
                                                                int[] arr = {89, 9, 11, 14, 76, 54, 22};
           ascratch 8.iava
                                                                 heapSort(arr);
           scratch_9.java
                                                                 System.out.println(Arrays.toString(arr));
           scratch_10.java
scratch_11.java
scratch_12.java
## scratch_12.java | 54 }

Run: | Main × |

C:\Users\Lenovo\.jdks\openjdk-15.0.2\bin\java.exe ...

[9, 11, 14, 22, 54, 76, 89]
```

Analysis:-

Process finished with exit code 0

‡ Git ▶ Run ≔ TODO ● Problems ☑ Terminal ♠ Profiler ▲ Build

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).

1 Event Log

🗞 22:18 LF UTF-8 4 spaces 🗜 master 🦫