***Submitted By :***

**Name : Khushant Wankhede**

**Roll Number : 29**

**Division : 2**

**Batch : SECTION B**

***DAA Assignments***

# Aim : Write a program to implement Binary Search on a large set of elements. The elements can be read from a file. Show the output on the screen. Calculate the time taken to complete this search for

# element found and

# element not found cases.

# Use the sequential file from the dataset here as input to your code.

***Code :***

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Scanner;

public class binary {

public static int binarySearch(ArrayList<Integer> arr, int target) {

int left = 0;

int right = arr.size() - 1;

while (left <= right) {

int mid = (right + left) / 2;

if (arr.get(mid) == target) {

return mid;

}

if (arr.get(mid) < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return -55;

}

public static void main(String[] args) {

String sortedNumFile = "sortedNumber.txt";

try {

ArrayList<Integer> noList = new ArrayList<>();

BufferedReader reader = new BufferedReader(new FileReader(sortedNumFile));

Scanner sc = new Scanner(System.in);

System.out.println("enter value to search : ");

int target = sc.nextInt();

String line;

while ((line = reader.readLine()) != null) {

noList.add(Integer.parseInt(line.trim()));

}

reader.close();

int result = binarySearch(noList, target);

if (result == -55) {

System.out.println("Number " + target + " not found in the file.");

} else {

System.out.println("Number " + target + " found at index " + result + " in the file.");

}

} catch (IOException e) {

System.out.println("An error occurred while reading the file.");

e.printStackTrace();

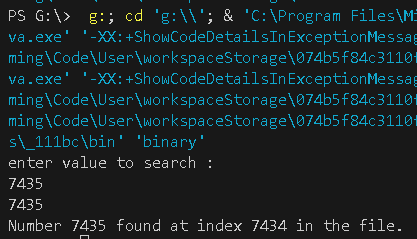
}

}

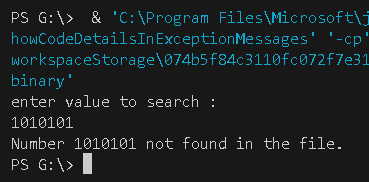
}

**problem 1 :** **Element is found**

Output :



**problem 2 :** Element is Not Found



**Conclusion:**

In conclusion, binary search is an efficient algorithm for searching a sorted array, with a time complexity of O(log n), making it significantly faster than linear search for large datasets. By repeatedly dividing the search interval in half, binary search ensures that only a portion of the data is considered at each step, resulting in fewer comparisons and faster results. However, it is important to note that this algorithm is only applicable to sorted arrays or lists. In cases where the data is unsorted, sorting must first be performed, which could increase the overall time complexity depending on the sorting method used. Therefore, binary search is ideal for situations where frequent searches are needed on static or pre-sorted datasets