|  |  |  |  |
| --- | --- | --- | --- |
|  | Bansilal Ramnath Agarwal Charitable Trust's  Vishwakarma Institute of Information Technology  **Department of**  **Artificial Intelligence and Data Science** | | |
| Student Name: Shreyash Sanjay Rahinj | | | |
| Class: TY | Division: A | | Roll No: 371045 |
| Semester: 5th | | Academic Year: 2022-23 | |
| Subject Name & Code: Design & Analysis of Algorithm ADUA31202 | | | |
| Title of Assignment: Assignment 1 | | | |
| Date of Performance: 02/08/2023 | | Date of Submission: 22/08/2023 | |

Aim: Write a program to perform binary search on an unsorted random list of at least 5000 elements. The key element should be user input. Use the Divide & Conquer method to implement this program.

Background Information

Theory :

***Binary Search****is defined as a*[*searching algorithm*](https://www.geeksforgeeks.org/searching-algorithms/)*used in a sorted array by****repeatedly dividing the search interval in half****. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(log N).*

**Conditions for when to apply Binary Search in a Data Structure:**

To apply Binary Search algorithm:

* The data structure must be sorted.
* Access to any element of the data structure takes constant time.

How binary search works? :

In this algorithm,

* Divide the search space into two halves by [**finding the middle index “mid”**](https://www.geeksforgeeks.org/problem-binary-search-implementations/).
* Compare the middle element of the search space with the key.
* If the key is found at middle element, the process is terminated.
* If the key is not found at middle element, choose which half will be used as the next search space.
  + If the key is smaller than the middle element, then the left side is used for next search.
  + If the key is larger than the middle element, then the right side is used for next search.
* This process is continued until the key is found or the total search space is exhausted.

**How to Implement Binary Search?**

The **Binary Search Algorithm** can be implemented in the following two ways

* Iterative Binary Search Algorithm
* Recursive Binary Search Algorithm

Time Complexity Analysis :

1. Iterative  Binary Search Algorithm:

Here we use a while loop to continue the process of comparing the key and splitting the search space in two halves.

* Time Complexity: O(log N)
* Auxiliary Space: O(1)

2. Recursive  Binary Search Algorithm:

Create a recursive function and compare the mid of the search space with the key. And based on the result either return the index where the key is found or call the recursive function for the next search space.

* Time Complexity:
  + Best Case: O(1)
  + Average Case: O(log N)
  + Worst Case: O(log N)
* Auxiliary Space: O(1), If the recursive call stack is considered then the auxiliary space will be O(logN).

Code :

import java.util.Random;

import java.util.Scanner;

class Assignment{

    static void quickSort(int [] arr,int low,int high){

        if(low<high){

            int pivot = partition(arr,low,high);

            quickSort(arr, low, pivot-1);

            quickSort(arr, pivot+1, high);

        }

    }

    static int partition(int [] arr,int low,int high){

        int pivot = arr[high];

        int i = low - 1;

        for(int j=low;j<high;j++){

            if(arr[j]<pivot){

                i++;

                swap(arr,i,j);

            }

        }

        swap(arr, i+1, high);

        return i+1;

    }

    static void swap(int[] arr,int i,int j){

        int temp = arr[j];

        arr[j] = arr[i];

        arr[i] = temp;

    }

    static int binarySearch(int num,int[] arr){

        int startIndex = 0;

        int endIndex = arr.length - 1;

        int middleIndex;

        while(startIndex<=endIndex){

            middleIndex = startIndex + (endIndex - startIndex)/2;

            if(num==arr[middleIndex]) return middleIndex;

            else if(num<arr[middleIndex]) endIndex = middleIndex - 1;

            else if(num>arr[middleIndex]) startIndex = middleIndex + 1;

        }

        return -1;

    }

    static void printN(int n,int [] arr){

        System.out.print("[ ");

        for(int i=0;i<arr.length;i++){

            if(arr[i] > n) {

                System.out.print(arr[i] + " ");

                break;

            }

            System.out.print(arr[i] + " ");

        }

        System.out.println("]");

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        Random rand =new Random();

        int [] rand\_int1= new int[5000];

        for(int i=0;i<5000;i++){

            rand\_int1[i] = rand.nextInt(5000);

        }

        quickSort(rand\_int1, 0, 4999);

        System.out.println("Enter number you would like to search");

        int n = sc.nextInt();

        printN(n,rand\_int1);

        int index = binarySearch(n, rand\_int1);

        if(index != -1) System.out.println("The element is found at index = " + index);

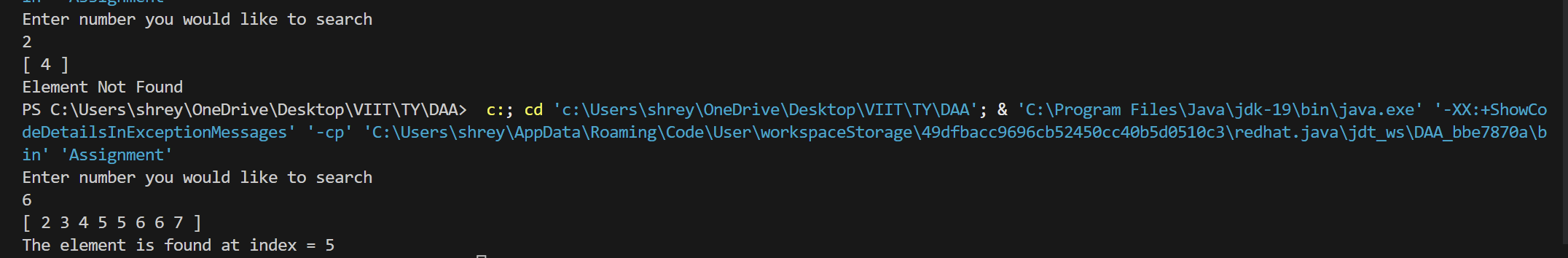
        else System.out.println("Element Not Found");

        sc.close();

    }

}

Results or Experimentation:



Conclusion:

Thus we successfully performed all the operations on tree.