Activity 4.3

José Carlos Martínez Núñez, Tania Sayuri Guizado Hernández

Tecnológico de Monterrey

November 14, 2022

Contents

Problem Description

In teams of two people and using the Randomized Algorithms programming technique, build a program in C++ that implements the Randomized Binary Search Algorithm to determine if an integer in a set of integers, through the Las Vegas Randomized Algorithm method, in the following reference you can consult this method: Randomized Algorithms

The program receives an integer number to be searched in the set of integers, in the following line the set of integers separated by commas.

The output of the program will indicate in which index is the number to be searched, with the format: "The element is present in the index: `i`", `i` will be the value of the index. In case the data is not found, the message: "The element is not present in the array" will be displayed.

Randomized Algorithms

The Randomized Algorithms are a class of algorithms that use random numbers to solve problems. There are two types of Randomized Algorithms:

- Las Vegas Randomized Algorithms: The algorithm always returns the correct answer.
- Monte Carlo Randomized Algorithms: The algorithm returns an answer that is correct with a certain probability.

The Randomized Binary Search Algorithm is a Las Vegas Randomized Algorithm, which means that it always returns the correct answer.

Algorithm Implementation

First, it is important to note that the Randomized Binary Search Algorithm is a variant of the Binary Search Algorithm, which means that it is necessary for the set of integers to be sorted.

The Randomized Binary Search Algorithm is based on the following steps:

1. Generate a random number between 0 and the size of the set of integers.
2. Compare the number to be searched with the number in the index generated in step 1.
3. If the number to be searched is greater than the number in the index generated in step 1, then the number to be searched is in the right half of the set of integers.
4. If the number to be searched is less than the number in the index generated in step 1, then the number to be searched is in the left half of the set of integers. Then, we generate a random number between the index generated in step 1 and the size of the set of integers.

5. We do the same for the right half of the set of integers, generating a random number between 0 and the index generated in step 1.
6. We repeat the steps 2 to 5 until we find the number to be searched.

We created two functions to implement the Randomized Binary Search Algorithm:

- `int getRandomNumber(int lower_bound, int upper_bound)`: This function generates a random number between the lower bound and the upper bound.
- `int randomizedBinarySearch(vector<int> &arr, int left, int right, int x)`: This function implements the Randomized Binary Search Algorithm. It is a recursive function that receives the set of integers, the left and right bounds of the set of integers, and the number to be searched.

The `randomizedBinarySearch` function is implemented as follows:

```
int randomizedBinarySearch(vector<int> &arr, int left, int right, int x)
{
    if (right >= left)
    {
        int mid = getRandomNumber(left, right);

        if (arr[mid] == x)
            return mid;
        if (arr[mid] > x)
            return randomizedBinarySearch(arr, left, mid - 1, x);
        return randomizedBinarySearch(arr, mid + 1, right, x);
    }
    return -1;
}
```

The time complexity of the Randomized Binary Search Algorithm is $O(\log n)$, the same as the Binary Search Algorithm.

Test Cases

The following test cases were used to test the program:

- `example_test_1.txt`: The example given in the problem description. The element is present in the index 3.

- `example_test_2.txt`: The example given in the problem description. The element is not present in the array.
- `array_not_ordered.txt`: The array is not ordered. The program will not work.
- `100_numbers.txt`: The element is present in the index 62.
- `50_numbers.txt`: The element is not present in the array.

The results of each test case are located in the `results` folder.