Binary Searching in Java Without Recursion

See how binary searching works on your Java arrays and consider the approaches of implementing those searches both iteratively and recursively.

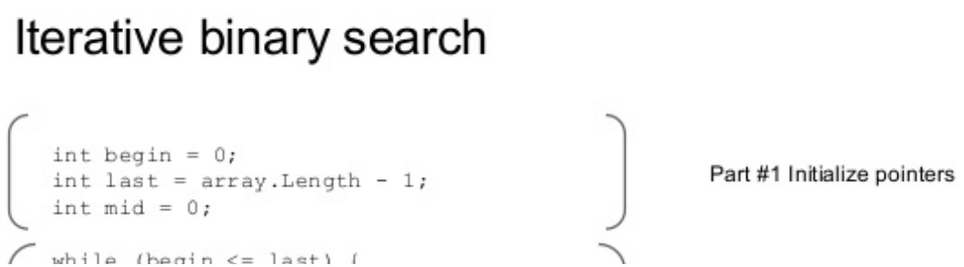
by Javin Paul MVB · **Jun. 05, 18 · Java Zone · Tutorial**

This week's task is to implement **binary search** in Java, you need to write both iterative and recursive binary search algorithm.

In computer science, a binary search, or half-interval search, is a divide and conquer algorithm that locates the position of an item in a sorted array. Binary searching works by comparing an input value to the middle element of the array. The comparison determines whether the element equals the input, is less than the input, or is greater than the input.

When the element being compared equals the input, the search stops and typically returns the position of the element. If the element is not equal to the input, then a comparison is made to determine whether the input is less than or greater than the element. Depending on which it is, the algorithm then starts over, but only searching the top or a bottom subset of the array's elements. If the input is not located within the array, the algorithm will usually output a unique value indicating this.

Binary search algorithms typically halve the number of items to check with each successive iteration, thus locating the given item (or determining its absence) in logarithmic time.



```
        mid = (begin + last) / 2;
        if (array[mid] < x) {
            begin = mid + 1;
        }
        else if (array[mid] > x) {
            last = mid - 1;
        }
        else {
            return mid;
        }
    }

    return -1;
```

Part #2 Search

Binary Search Implementation in Java

The algorithm is implemented recursively. Also, an interesting fact to to know about binary search implementation in Java is that Joshua Bloch, author of famous Effective Java book wrote the binary search in "java.util.Arrays".

```
import java.util.Arrays;
import java.util.Scanner;
```

```
/**
* Java program to implement Binary Search. We have implemented Iterative
* version of Binary Search Algorithm in Java
*
* @author Javin Paul
*/
public class IterativeBinarySearch {

    public static void main(String args[]) {

        int[] list = new int[]{23, 43, 31, 12};
        int number = 12;
```

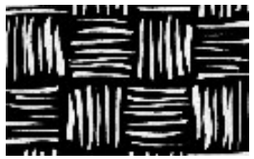
```
        Arrays.sort(list);
        System.out.printf("Binary Search %d in integer array %s %n", number,
                          Arrays.toString(list));
        binarySearch(list, 12);

        System.out.printf("Binary Search %d in integer array %s %n", 43,
                          Arrays.toString(list));
        binarySearch(list, 43);

        list = new int[]{123, 243, 331, 1298};
        number = 331;
        Arrays.sort(list);
        System.out.printf("Binary Search %d in integer array %s %n", number,
                          Arrays.toString(list));
        binarySearch(list, 331);

        System.out.printf("Binary Search %d in integer array %s %n", 331,
                          Arrays.toString(list));
        binarySearch(list, 1333);

        // Using Core Java API and Collection framework
        // Precondition to the Arrays.binarySearch
        Arrays.sort(list);

        // Search an element
        int index = Arrays.binarySearch(list, 3);

    }

    /**
     * Perform a binary Search in Sorted Array in Java
     *
     * @param input
```

```
     * @param number
     * @return location of element in array
     */
    public static void binarySearch(int[] input, int number) {
        int first = 0;
        int last = input.length - 1;
        int middle = (first + last) / 2;

        while (first <= last) {
            if (input[middle] &lt; number) {
                first = middle + 1;
            } else if (input[middle] == number) {
                System.out.printf(number + " found at location %d %n", middle);
                break;
            } else {
                last = middle - 1;
            }
            middle = (first + last) / 2;
        }
        if (first &gt; last) {
            System.out.println(number + " is not present in the list.\n");
        }
    }
}


Output
Binary Search 12 in integer array [12, 23, 31, 43]
12 found at location 0
Binary Search 43 in integer array [12, 23, 31, 43]
43 found at location 3
Binary Search 331 in integer array [123, 243, 331, 1298]
331 found at location 2
```

```
331 found at location 2
Binary Search 331 in integer array [123, 243, 331, 1298]
1333 is not present in the list.
```

That's all about how to implement an iterative binary search in Java.

Like This Article? Read More From DZone



The Previously Sorted Algorithm

Adding Terms to Javadoc Search With Java 9

Creating Powerful Solr Filters Using MVEL Expressions

Free DZone Refcard
Getting Started With Kotlin

Topics: JAVA , BINARY SEARCH , RECURSION , ITERATIVE SEARCHING , TUTORIAL

Published at DZone with permission of Javin Paul , DZone MVB. See the original article here.
Opinions expressed by DZone contributors are their own.