Analysis of Algorithm

Practical no 4 : Binary Search

Code :

public class Binarysearch {

    public static int binarySearch(int[] arr, int target) {

        int left = 0;

        int right = arr.length - 1;

        int mid;

        while (left <= right) {

            mid = (left + right) / 2;

            if (arr[mid] == target)

                return mid;

                else if (arr[mid] < target)

                    left = mid + 1;

                else

                    right = mid - 1;

        }

        return 0;

    }

    public static void main(String[] args) {

        int[] arr = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};

        int target = 13;

        int result = binarySearch(arr, target);

        if (result == 0)

            System.out.println("Element not found");

        else

            System.out.println("Element "+ target + " found at index: " + result);

    }

}

Output :-

Element 13 found at index: 6

Analysis :

1. binarySearch Method:

This method takes two arguments:

* int[] arr: A sorted array of integers.
* int target: The value to search for in the array.

Variables:

* left: Initialized to 0, it marks the start of the current search interval.
* right: Initialized to arr.length - 1, it marks the end of the current search interval.
* mid: The midpoint index of the current search interval.

While Loop:

* The loop continues as long as left <= right, meaning the search interval is not empty.
* Inside the loop, the middle element is calculated as mid = (left + right) / 2.
* The algorithm then checks the value at the mid index:
  + If the value at arr[mid] is equal to target, the method returns the index mid (i.e., the target is found).
  + If arr[mid] is less than the target, it means the target is in the right half of the current search interval. So, left is updated to mid + 1.
  + If arr[mid] is greater than the target, it means the target is in the left half of the current search interval. So, right is updated to mid - 1.

Return:

* If the target is found, its index is returned.
* If the target is not found (when the left pointer exceeds the right pointer), 0 is returned (which is a logical error — more on this below).

2. main Method:

* Initializes an array arr with some integers: {1, 3, 5, 7, 9, 11, 13, 15, 17, 19}.
* Sets target = 13 (the value to search for in the array).
* Calls the binarySearch method and stores the result in result.
* If result == 0, the program prints "Element not found", which is incorrect since a return value of 0 is being used to signal both a successful and unsuccessful search.
* If result != 0, it prints the index where the target is found.

Time Complexity:

* Best Case: O(1) (when the target is found at the first midpoint)
* Average Case: O(log n) (on average, log₂(n) comparisons to find the target)
* Worst Case: O(log n) (in the worst case, log₂(n) comparisons)

Space Complexity:

* Space Complexity: O(1) because binary search only uses a constant amount of space (a few integer variables like left, right, and mid).