Searching sorting and Bitmanupulation Solutions

Q1. Given an array. Find the number X in the array. If the element is present, return the index of the element, else print “Element not found in array”. Input the size of array, array from user and the element X from user. Use Linear Search to find the element. import java.util.Scanner;

public class LinearSearchExample {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the size of the array

System.out.print("Enter the size of the array: ");

int size = scanner.nextInt();

// Initialize the array

int[] array = new int[size];

// Input the elements of the array

System.out.println("Enter " + size + " elements of the array:");

for (int i = 0; i < size; i++) {

array[i] = scanner.nextInt();

}

// Input the element to search for

System.out.print("Enter the element to find: ");

int x = scanner.nextInt();

// Perform linear search

int index = linearSearch(array, x);

// Output the result

if (index != -1) {

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

} else {

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

}

scanner.close();

}

// Linear search method

public static int linearSearch(int[] array, int x) {

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

if (array[i] == x) {

return i; // Return index if element is found

}

}

return -1; // Return -1 if element is not found

}

}

Q2. Given an array and an integer “target”, return the last occurrence of “target” in the array. If the target is not present return -1.

Input 1: arr = [1 1 1 2 3 4 4 5 6 6 6 6] , target = 4 Output 1: 6 Input 2: arr = [2 2 2 6 6 18 29 30 30 30] , target = 15

import java.util.Scanner;

public class LastOccurrenceFinder {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the size of the array

System.out.print("Enter the size of the array: ");

int size = scanner.nextInt();

// Initialize the array

int[] array = new int[size];

// Input the elements of the array

System.out.println("Enter " + size + " elements of the array:");

for (int i = 0; i < size; i++) {

array[i] = scanner.nextInt();

}

// Input the target element

System.out.print("Enter the target element: ");

int target = scanner.nextInt();

// Find the last occurrence of the target

int index = findLastOccurrence(array, target);

// Output the result

System.out.println("The last occurrence of the target is at index: " + index);

scanner.close();

}

// Method to find the last occurrence of the target in the array

public static int findLastOccurrence(int[] array, int target) {

// Traverse the array from the end to the beginning

for (int i = array.length - 1; i >= 0; i--) {

if (array[i] == target) {

return i; // Return the index of the last occurrence

}

}

return -1; // Return -1 if the target is not found

}

}

Q3. Given a sorted binary array, efficiently count the total number of 1’s in it.

Input 1: arr = [0 0 0 0 1 1 1 1 1 1] Output 1: 6

Input 2: arr = [ 0 0 0 0 0 1 1] Output 2: 2

public class CountOnesInSortedBinaryArray {

public static void main(String[] args) {

// Test inputs

int[] arr1 = {0, 0, 0, 0, 1, 1, 1, 1, 1, 1};

int[] arr2 = {0, 0, 0, 0, 0, 1, 1};

// Count the number of 1's in the arrays

System.out.println("Number of 1's in arr1: " + countOnes(arr1));

System.out.println("Number of 1's in arr2: " + countOnes(arr2));

}

// Method to count the number of 1's in a sorted binary array

public static int countOnes(int[] arr) {

int n = arr.length;

// Perform binary search to find the first occurrence of 1

int left = 0, right = n - 1;

int firstOneIndex = -1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == 1) {

firstOneIndex = mid;

right = mid - 1; // Search in the left half for the first occurrence

} else {

left = mid + 1; // Search in the right half

}

}

// If firstOneIndex is -1, there are no 1's in the array

if (firstOneIndex == -1) {

return 0;

}

// Number of 1's is the length of the array minus the index of the first 1

return n - firstOneIndex;

}

}

Q4. Given a sorted integer array containing duplicates, count occurrences of a given number. If the element is not found in the array, report that as well. Input: nums[] = [2, 5, 5, 5, 6, 6, 8, 9, 9, 9] target = 5 Output: Target 5 occurs 3 times

public class CountOccurrences {

public static void main(String[] args) {

// Test inputs

int[] nums1 = {2, 5, 5, 5, 6, 6, 8, 9, 9, 9};

int[] nums2 = {2, 5, 5, 5, 6, 6, 8, 9, 9, 9};

// Test cases

System.out.println(countOccurrences(nums1, 5)); // Output: Target 5 occurs 3 times

System.out.println(countOccurrences(nums2, 6)); // Output: Target 6 occurs 2 times

}

// Method to count occurrences of a target in a sorted array

public static String countOccurrences(int[] nums, int target) {

int firstIndex = findFirstOccurrence(nums, target);

if (firstIndex == -1) {

return "Target " + target + " not found in the array";

}

int lastIndex = findLastOccurrence(nums, target);

int count = lastIndex - firstIndex + 1;

return "Target " + target + " occurs " + count + " times";

}

// Method to find the first occurrence of the target using binary search

private static int findFirstOccurrence(int[] nums, int target) {

int left = 0, right = nums.length - 1;

int result = -1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (nums[mid] == target) {

result = mid;

right = mid - 1; // Continue searching in the left half

} else if (nums[mid] < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return result;

}

// Method to find the last occurrence of the target using binary search

private static int findLastOccurrence(int[] nums, int target) {

int left = 0, right = nums.length - 1;

int result = -1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (nums[mid] == target) {

result = mid;

left = mid + 1; // Continue searching in the right half

} else if (nums[mid] < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return result;

}

}

Q5: Given a positive integer num, return true if num is a perfect square or false otherwise. A perfect square is an integer that is the square of an integer. In other words, it is the product of some integer with itself. Example 1: Input: num = 16 Output: true Explanation: We return true because 4 \* 4 = 16 and 4 is an integer.:

public class PerfectSquareChecker {

public static void main(String[] args) {

// Test cases

System.out.println(isPerfectSquare(16)); // Output: true

System.out.println(isPerfectSquare(14)); // Output: false

}

// Method to check if a number is a perfect square using integer arithmetic

public static boolean isPerfectSquare(int num) {

if (num < 0) {

return false; // Negative numbers cannot be perfect squares

}

int sqrt = (int) Math.sqrt(num);

return sqrt \* sqrt == num;

}

}

Question 6. T(n) = 2T(n/2) + K, Solve using Recurrence tree method.



