**Searching in an array- Linear& Binary Search**

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.

Solution: import java.io.\*;

import java.util.\*;

public class Main{

public static void main(String args[]){

int m;

Scanner sc=new Scanner(System.in);

System.out.print("Enter the number of elements you want to add : ");

m=sc.nextInt();

int []arr = new int[m];

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

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

arr[i] = sc.nextInt();

}

int element;

System.out.print("Enter the elements to be searched in array");

element = sc.nextInt();

int idx = -1;

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

if(arr[i]==element){

idx = i;

break;

}

}

if(idx!=-1){

System.out.println(idx);

}

else{

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

}

}

}

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

Output 2: -1

Solution: import java.io.\*;

import java.util.\*;

public class Main{

public static int lastOccurrence(int[] nums, int low, int high , int target){

int answer = -1;

while(low <= high){

int mid = low + (high - low)/2;

if(nums[mid] == target){

answer = mid;

low = mid + 1; //if you found the target or if target is greater than the current element, to find last occurrence move to right half of the array

}

else if(nums[mid] > target){

high = mid - 1;

}

else low = mid + 1;

}

return answer;

}

public static void main(String args[]){

int m;

Scanner sc=new Scanner(System.in);

System.out.print("Enter the number of elements you want to add : ");

m=sc.nextInt();

int []arr = new int[m];

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

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

arr[i] = sc.nextInt();

}

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

int target;

Scanner s1 = new Scanner(System.in);

target = s1.nextInt();

System.out.println("The last occurrence of target is at index : " + lastOccurrence(arr , 0 , m - 1 , target));

}

}

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

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

Input 2: arr = [ ^ ^ ^ ^ ^ 1 1]

Output 2: 2

Solution: import java.io.\*;

import java.util.\*;

public class Main{

public static int numberOf1s(int[] nums, int low, int high){

while(low <= high){

int mid = low + (high - low)/2;

if(nums[mid] == 0){

low = mid + 1;

}

else {

high = mid - 1;

}

}

return (nums.length - low);

}

public static void main(String args[]){

int m;

Scanner sc=new Scanner(System.in);

System.out.print("Enter the number of elements you want to add : ");

m=sc.nextInt();

int []arr = new int[m];

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

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

arr[i] = sc.nextInt();

}

System.out.println("The number of one's in the given array is/are: " + numberOf1s(arr , 0 , m - 1));

}

}

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

Input: nums[] = [2, 5, 5, 5, 6, 6, 8, 9, 9, 9] target = 6

Output: Target 6 occurs 2 times

Solution: import java.io.\*;

import java.util.\*;

public class Main{

public static int lastOccurrence(int[] a, int low, int high, int target){

int answer = -1;

while(low <= high){

int mid = low + (high - low)/2;

if(a[mid] == target){

answer = mid;

low = mid + 1; //if you found the target or if target is greater than the current element, to find last occurrence move to right half of the array

}

else if(a[mid] > target){

high = mid - 1;

}

else low = mid + 1;

}

return answer;

}

public static int firstOccurrence(int[] a, int low , int high , int target){

int answer = -1;

while(low <= high) {

int mid = (low + high)/2;

if(a[mid] == target) {

answer = mid;

high = mid - 1; // trying to find the minimum index at which value x is present

}

else if(a[mid] > target) {

high = mid - 1;

}

else low = mid + 1;

}

return answer;

}

public static void main(String args[]){

int m;

Scanner sc=new Scanner(System.in);

System.out.print("Enter the number of elements you want to add : ");

m=sc.nextInt();

int []arr = new int[m];

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

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

arr[i] = sc.nextInt();

}

int target;

Scanner sc1 = new Scanner(System.in);

System.out.print("enter the target: ");

target = sc1.nextInt();

int first = firstOccurrence(arr , 0 , m - 1 , target);

int last = lastOccurrence(arr , 0 , m - 1 , target);

if(first == last && first == -1)System.out.println("The target does not exist in the array.");

else System.out.println("The frequency of target in the given array is " + (last - first + 1) + " time/times");

}

}

Q5. Given a posipive inpeger num, repurn prue if num is a perfecp square or false opherwise.

A perfecp square is an inpeger phap is phe square of an inpeger. In opher words, ip is phe producp of some inpeger

wiph ipself.

Example 1:

Inpup: num = 16

Ouppup: prue

Explanapion: We return true becauGe 4 \* 4 = 16 and 4 iG an integer.

Example 2:

Inpup: num = 14

Ouppup: false

Solution: import java.io.\*;

import java.util.\*;

public class Main{

public static boolean isPerfectSquare(int num) {

if(num == 1) return true ; // edge case

long start = 0 ; // Take Long as the inputs have large value

long end = num /2 ;

while(start <= end ){

long mid = start + ( end - start ) / 2 ;

if (mid \* mid > num) end = mid - 1 ; // Mid is greater than the squareroot of the number

else if (mid \* mid < num) start = mid + 1 ; // Mid is less than our required number

else return true; // we found our squareroot number

}

return false ;

}

public static void main(String args[]){

int m;

Scanner sc=new Scanner(System.in);

System.out.print("Enter the number : ");

m=sc.nextInt();

System.out.println("The given number is a perfect square: " + isPerfectSquare(m));

}

}

**Number Systems**

Problem 1: given a number, print its binary representation. [easy]

Input 1: number = 5

Output 1: 101

Input 2: number = 10

Output 2: 1010

Problem 2: given a number ‘n’, predict whether it is a power of two or not. [medium]

Input 1: n = 15

Output 1: False

Input 2: n = 32

Output 2: True

Solution: 5 is the decimal number

2 | 5

—----

2 | 2 1

—----

2 | 1 0

Number 10

2 | 10

—-----

2 | 5 0

—-----

2 | 2 1

—-----

2 | 1 0

Reading in reverse order from bottom to top gives us 1010 as the binary representation of 10.

Number 15. Converting it in binary 1111, as its more than 1 set bits in the binary representation therefore it’s

not power of 2.

Number 32. Converting it to binary 10000, as it has only 1 set bit, therefore its power of 2.

Q3. Problem 1: Given a number. Using bit manipulation, check whether it is odd or even.

Input 8, Even

3, False

Solution: import java.io.\*;

import java.util.\*;

import java.util.Scanner;

public class Main

{

public static void main(String[] args) {

int number;

System.out.println("Enter the integer: ");

// Create Scanner object

Scanner s = new Scanner(System.in);

// Read the next integer from the screen

number = s.nextInt();

if((number & 1) == 1) System.out.println("Given number is odd.");

else System.out.println("Given number is even.");

}

}

Q4. Given a number, count the number of set bits in that number without using an extra space.

Note : bit ‘1’ is also known as set bit.

Solution: import java.io.\*;

import java.util.\*;

import java.util.Scanner;

public class Main

{ public static int countSetBits(int n){

int count = 0;

while (n > 0) {

count += n & 1;

n >>= 1;

}

return count;

}

public static void main(String[] args) {

int number;

System.out.println("Enter the integer: ");

// Create Scanner object

Scanner s = new Scanner(System.in);

// Read the next integer from the screen

number = s.nextInt();

int answer = countSetBits(number);

System.out.println("The number of set bits in the given number are " + answer);

}

}

Q5. Given an integer array, duplicates are present in it in a way that all duplicates appear an even

number of times except one which appears an odd number of times. Find that odd appearing

element in linear time and without using any extra memory.

For example,

Input : arr[] = [4, 3, 6, 2, 6, 4, 2, 3, 4, 3, 3]

Output : The odd occurring element is 4.

Solution: import java.io.\*;

import java.util.\*;

import java.util.Scanner;

public class Main

{ public static int findOddOccuring(int[] arr)

{

int xor = 0;

for (int i: arr) {

xor = xor ^ i;

}

return xor;

}

public static void main(String[] args) {

int n;

Scanner sc=new Scanner(System.in);

System.out.print("Enter the number of elements you want to store: ");

//reading the number of elements from the that we want to enter

n=sc.nextInt();

//creates an array in the memory of length 10

int[] array = new int[10];

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

for(int i=0; i<n; i++)

{

//reading array elements from the user

array[i]=sc.nextInt();

}

System.out.println("The odd occurring element is " + findOddOccuring(array));

}

}

**Sorting Array**

Q1. Write a program to sort an array in descending order using bubble sort.

Input Array {3,5,1,6,0}

Output Array: {6, 5, 3, 1, 0}

Solution: import java.io.\*;

import java.util.\*;

public class Sort {

// 0 based indexing used

public static void bubbleSort(int[] a) {

int n = a.length;

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

boolean flag = false;

for (int j = 0; j < n - i - 1; j++) {

if (a[j] < a[j + 1]) {

flag = true;

// swap the values of a[j] and a[j+1]

int temp = a[j];

a[j] = a[j + 1];

a[j + 1] = temp;

}

}

// No Swapping happened, array is sorted

if (!flag) {

return;

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] arr = new int[n];

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

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

arr[i] = sc.nextInt();

}

bubbleSort(arr);

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

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

}

}

}

Q2. WAP to sort an array in descending order using selection sort

Input Array {3,5,1,6,0}

Output Array: {6, 5, 3, 1, 0}

Solution: import java.io.\*;

import java.util.\*;

public class Sort {

// 0 based indexing used

public static void selectionSort(int[] a) {

int n = a.length;

for (int i = 0; i < n - 1; i++)

// i represents the current index

{

// Find the maximum element in unsorted part of the array

int max\_index = i;

for (int j = i + 1; j < n; j++) {

if (a[j] > a[max\_index])

max\_index = j;

}

// Swap the found maximum element with the current element

if (max\_index != i) {

int temp = a[max\_index];

a[max\_index] = a[i];

a[i] = temp;

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] arr = new int[n];

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

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

arr[i] = sc.nextInt();

}

selectionSort(arr);

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

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

}

System.out.print("\n");

}

}

Q3. WAP to sort an array in decreasing order using insertion sort

Input Array {3,5,1,6,0}

Output Array: {6, 5, 3, 1, 0}

Solution: import java.io.\*;

import java.util.\*;

public class Sort {

public static void insertionSort(int[] a) {

int n = a.length;

for (int i = 1; i < n; i++) {

int j = i;

// Insert a[i] into sorted left part 0..i-1

while (j > 0 && a[j] > a[j - 1]) {

// Swap a[j] and a[j-1]

int temp = a[j];

a[j] = a[j - 1];

a[j - 1] = temp;

// Decrement j by 1

j--;

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] arr = new int[n];

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

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

arr[i] = sc.nextInt();

}

insertionSort(arr);

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

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

}

System.out.print("\n ");

}

}

Q4. Find out how many pass would be required to sort the following array in decreasing order

using bubble sort

Input Array {3,5,1,6,0}

Solution: Original Array is {3 5 1 6 0}

In first iteration array is {5 3 6 1 0}

In second iteration array is {5 6 3 1 0}

In third iteration array is {6 5 3 1 0}

Q5. Find out the number of iterations to sort the array in descending order using selection sort.

Input Array {3,5,1,6,0}

Solution: 3 iterations are required.

Original Array is {3 5 1 6 0}

In first iteration array is {6 5 1 3 0}

In second iteration array is {6 5 1 3 0}

In third iteration array is {6 5 3 1 0}

Now the array is sorted.