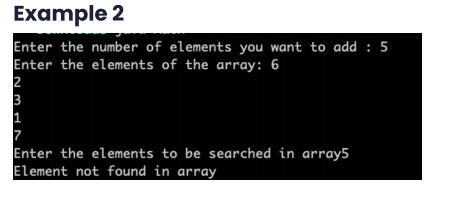
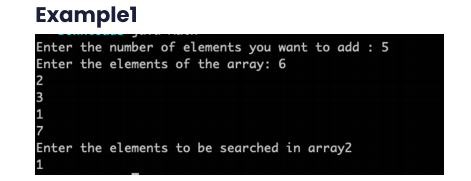
**Assignment**

**Searching in an array- Linear and 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:

**package** Assignment\_Searching;

**import** java.util.Scanner;

**public** **class** Linear\_Search\_Problem1

{

**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 = 1;

**break**;

}

}

**if**(idx != -1)

{

System.***out***.print("Searched element index number is : "+idx);

}

**else**

{

System.***out***.print("Element not found in an array");

}

}

}

/\*

Output:

Enter the number of elements you want to add : 5

Enter the elements of the array : 6 2 3 1 7

Enter the elements to be searched in array : 2

Searched element index number is : 1

\*/

/\*

Output:

Enter the number of elements you want to add : 5

Enter the elements of the array : 6 2 3 1 7

Enter the elements to be searched in array : 5

Element not found in an 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:

**package** Assignment\_Searching;

**import** java.util.Scanner;

**public** **class** Last\_Occurrence\_Problem2

{

**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 sc1 = **new** Scanner(System.***in***);

target = sc1.nextInt();

System.***out***.print("The Last Occurrence of Target is at Index : "+*lastOccurrence*(arr,0,m-1,target));

}

}

/\*

Output:

Enter the number of elements you want to add : 12

Enter the elements of the array : 1 1 1 2 3 4 4 5 6 6 6 6

Enter the target : 4

The Last Occurrence of Target is at Index : 6

\*/

/\*

Output:

Enter the number of elements you want to add : 10

Enter the elements of the array : 2 2 2 6 6 18 29 30 30 30

Enter the target : 15

The Last Occurrence of Target is at Index : -1

\*/

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

Solution:

**package** Assignment\_Searching;

**import** java.util.Scanner;

**public** **class** Number\_Of\_One\_Problem3

{

**public** **static** **int** findNumOfOne(**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***.print("The number of one's in the given array is : "+*findNumOfOne*(arr,0,m-1));

}

}

/\*

1st Output:

Enter the number of elements you want to add : 10

Enter the elements of the array : 0 0 0 0 1 1 1 1 1 1

The number of one's in the given array is : 6

2nd Output:

Enter the number of elements you want to add : 7

Enter the elements of the array : 0 0 0 0 0 1 1

The number of one's in the given array is : 2

\*/

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:

**package** Assignment\_Searching;

**import** java.util.Scanner;

**public** **class** Dublicate\_Count

{

**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 currentt 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***.print("The target does not exist in the array.");

}

**else**

{

System.***out***.print("The frequency of target in the gven array is : "+(last - first+1)+" time/times");

}

}

}

/\*

Output:

Enter the number of elements you want to add : 10

Enter the elements of the array : 2 5 5 5 6 6 8 9 9 9

Enter the Target :5

The frequency of target in the gven array is : 3 time/times

\*/

/\*

Output:

Enter the number of elements you want to add : 10

Enter the elements of the array : 2 5 5 5 6 6 8 9 9 9

Enter the Target :6

The frequency of target in the gven array is : 2 time/times

\*/

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.

Example 2:

Input: num = 14

Output: false

Explanation: We return false because 3.742 \* 3.742 = 14 and 3.742 is not an integer.

Solution:

**package** Assignment\_Searching;

**import** java.util.Scanner;

**public** **class** Perfact\_Square\_or\_not

{

**public** **static** **boolean** isPerfectSquare(**int** num)

{

**if**(num == 1) //edge case

{

**return** **true**;

}

**long** start = 0;//Take long as the input 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 square root of the number

}

**else** **if**(mid \* mid < num)

{

start = mid + 1; //mid is less than our required number

}

**else**

{

**return** **true**; //we found our square root 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***.print("The given number is a perfect square : "+*isPerfectSquare*(m));

}

}

/\*

Output:

Enter the Number : 16

The given number is a perfect square : true

\*/

/\*

Output:

Enter the Number : 14

The given number is a perfect square : false

\*/

**Assignment**

**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:

**package** Assignment\_Sorting\_Array;

**import** java.util.Scanner;

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

**int** n = sc.nextInt();

**int**[] arr = **new** **int**[n];

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

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

{

arr[i] = sc.nextInt();

}

*bubbleSort*(arr);

System.***out***.print("Print Array Element in Descending Order : ");

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

{

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

}

}

}

/\*

Output:

Enter the size of array : 5

Enter the elements of array : 3 5 1 6 0

Print Array Element in Descending Order (Using Bubble Sort)

: 6 5 3 1 0

\*/

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:

**package** Assignment\_Sorting\_Array;

**import** java.util.Scanner;

**public** **class** Selection\_Sort\_Problem2

{

// 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***.print("Enter the size of array : ");

**int** n = sc.nextInt();

**int**[] arr = **new** **int**[n];

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

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

{

arr[i] = sc.nextInt();

}

*selectionSort*(arr);

System.***out***.print("Print Array Element in Descending Order : ");

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

{

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

}

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

}

}

/\*

Output:

Enter the size of array : 5

Enter the elements of array : 3 5 1 6 0

Print Array Element in Descending Order (Using Selection Sort)

: 6 5 3 1 0

\*/

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:

**package** Assignment\_Sorting\_Array;

**import** java.util.Scanner;

**public** **class** Insertion\_Sort\_Problem3

{

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

**int** n = sc.nextInt();

**int**[] arr = **new** **int**[n];

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

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

{

arr[i] = sc.nextInt();

}

*insertionSort*(arr);

System.***out***.print("Print Array Element in Descending Order(Using Insertion Sort) : ");

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

{

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

}

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

}

}

/\*

/\*

Output:

Enter the size of array : 5

Enter the elements of array : 3 5 1 6 0

Print Array Element in Descending Order(Using Insertion Sort) :6 5 3 1 0

\*/

**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:**

**3 iterations are required**

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

**Assignment**

**Number System**

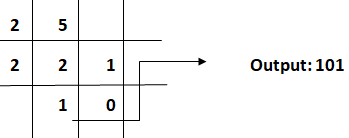
**Problem 1: Given a number, print its binary representation. [easy]**

**Input 1: number = 5**

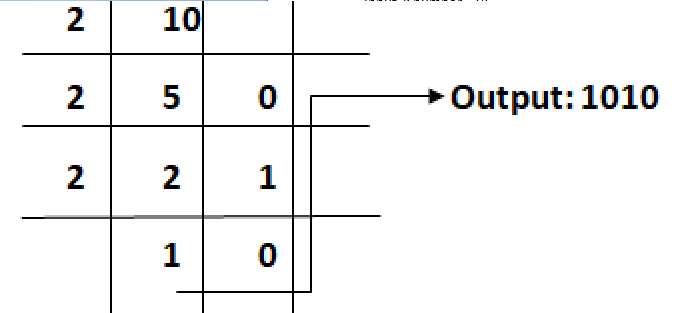
**Output 1: 101**

**Input 2: number = 10**

**Output 2: 1010**

**Solution:**

Reading numbers is reverse order gives us 101 as the binary representation



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

**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:** 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.

**Problem 3: Given a number. Using bit manipulation; check whether it is odd or even.**

**Input 8, Even**

**3, False**

**Solution:**

Approach :

We know that any odd number in its binary representation has the last digit (from right) as 1.

We used that concept and simply used the AND operator and operated the given number with 1.

We know the binary representation of 1 is also 1 and if the result of AND operator is 1, we can be sure that the given number is also odd since AND only returns C when both the operating bits are 1.

**package** Assignment\_NumberSystem;

**import** java.io.\*;

**import** java.util.\*;

**import** java.util.Scanner;

**public** **class** Given\_Even\_Odd\_Number

{

**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.");

}

}

}

/\*

Output:

Enter the integer:

5

Given number is odd.

\*/

/\*

Enter the integer:

6

Given number is even.

\*/

**Problem 4. 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:**

Approach:

We have extracted the last digit (from right), of the number using the AND operator. By operating AND operator on 1 and the number the corresponding bits of 1 and the number will be AND.

Whatever will be the result of AND we will add it to our “count” variable because the result can only be 0 and 1. So, if there is a set bit it will automatically get summed up in the variable “count”.

Once we are done with the last digit, we need to check upon other digits as well.

We know the right shift will remove the last bit from the right end and a new bit will take its place.

This way we can check for each bit whether it is set or not.

**package** Assignment\_NumberSystem;

**import** java.io.\*;

**import** java.util.\*;

**import** java.util.Scanner;

**public** **class** Count\_Bits\_Problem4

{

**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);

}

}

/\*

Output:

Enter the integer:

5

The number of set bits in the given number are 2

\*/

**Problem 5:. 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:**

Some key observations about 1or operator:

* XOR of ‘x’ with 0:
* X^0 = x
* XOR of ‘x’ with itself even number of times:
* X^x = 0
* XOR of ‘x’ with itself odd number of times:
* x^x^x = (x^(x^x)) = (x^0) = x
* (x ^ x ^ x ^ x ^ x) = (x ^ (x ^ x) ^ (x ^ x)) = (x ^ 0 ^0 )) = x
* This shows taking XOR of the same number an odd number of times results in the number itself whereas taking XOR even number of times will result in 0.
* So, if we take XOR of all array elements, even appearing elements will cancel each other, and we are left with the only odd appearing element.
* We will simply return that element.

**package** Assignment\_NumberSystem;

**import** java.io.\*;

**import** java.util.\*;

**import** java.util.Scanner;

**public** **class** Odd\_Occurrence\_Problem5

{

**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 11

**int**[] array = **new** **int**[11];

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));

}

}

/\*

Output:

Enter the number of elements you want to store: 11

Enter the elements of the array:

4 3 6 2 6 4 2 3 4 3 3

The odd occurring element is : 4

\*/