Bit Manipulation + Searching & Sorting [Day 11]

13 September 2024 17:35

Bit Manipulation + Searching & Sorting[Day 11]

Problem Statement: Game of XOR

Given an array of integers, the task is to find the XOR of all subarrays of the given array.

Definitions

- A **subarray** is a contiguous sequence of elements within an array.
- For example, if the array is `[1, 2, 3]`, the subarrays are:
  - `[1]`, `[2]`, `[3]`
  - `[1, 2]`, `[2, 3]`
  - `[1, 2, 3]`

The goal is to find the XOR of the XOR values of all these subarrays.

Example

Let's take an example with an array:

```
arr = [1, 2, 3]
```

output = 2

All subarrays and their XOR values:

- `[1] -> 1`
- `[2] -> 2`
- `[3] -> 3`
- `[1, 2] -> 1 ^ 2 = 3`
- `[2, 3] -> 2 ^ 3 = 1`
- `[1, 2, 3] -> 1 ^ 2 ^ 3 = 0`

Now, the XOR of all these subarrays is:

```
1 ^ 2 ^ 3 ^ 3 ^ 1 ^ 0 = 2
```

```
totalXor = 0;
for (i =0 to n):
    for(j = i to n):
        xor = 0
        for(k = i to j):
            xor ^= arr[k]
    totalXor  ^= xor;
```

```
public class GameOfXOR {
    public static int gameofXOR(int[] arr){
        int totalXor = 0;
        int n = arr.length;
        for(int i =0; i<n ; i++){
            for(int j = i; j< n ; j++){
                int xor = 0;
                for(int k = i ; k <=j; k++){
                    xor ^= arr[k];
                }
                totalXor ^= xor;
            }
```

```
                }
                totalXor ^= xor;
            }
        }
        return totalXor;
    }
    public static void main(String[] args) {
        int[] arr = {1,2,3};
        System.out.println(gameofXOR(arr));
    }
}
```

Hamming Distance Problem Explanation

Problem Statement:

The **Hamming Distance** between two integers is defined as the number of positions at which the corresponding bits differ. Given two integers `a` and `b`, the task is to compute the Hamming distance between them.

Example 1:

- **Input:** `a = 3`, `b = 5`
- **Binary Representation:**
  - `3` in binary is `011`
  - `5` in binary is `101`
- **Hamming Distance:** There are **2** differing bits

Example 2:

- **Input:** `a = 1`, `b = 4`
- **Binary Representation:**
  - `1` in binary is `001`
  - `4` in binary is `100`
- **Hamming Distance:** There are **2** differing bits

```
011
101
-----
110 & 1
0 & 1 = 0
1 & 1 = 1
1 & 1 = 1
```

```
10101010
01010101
---------
11111111
```

```
101010
010111
-------
111101
```

11111111 111101

000000 & 1

0

count = 5

```
Enter the first Integer (a) :
3
Enter the second Integer (b) :
5
Hamming Distance between : 3 & 5 is : 2
```

```
import java.util.Scanner;
public class HammingDistance {

    public static int hammingDistance(int a , int b){
        int xorResult = a ^ b;
        int count = 0;

        while(xorResult > 0){
            count += xorResult & 1;
            xorResult >>=1;
        }
        return count;
    }
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        System.out.println("Enter the first Integer (a) :");
        int a = scn.nextInt();
        System.out.println("Enter the second Integer (b) :");
        int b = scn.nextInt();

        int result = hammingDistance(a,b);
        System.out.println("Hamming Distance between : " + a + " & " + b + " is : " + result);
    }
}
```

```
int xorResult = a ^ b;
int count = 0;

while(xorResult > 0){
    count += xorResult & 1;
    xorResult >>=1;
}
return count;
```

Memory

xorResult = 0
a = 6
b = 8
count = 3

6 == 0110
8 == 1000

6 == 0110
8 == 1000
---
1 1 1 0 - 14

1. 1110 0001 ------ 0000
2. 0111 & 1 ------- 0001
3. 0011 &1 --------- 0001
4. 0001 & 1 --------- 0001
5. 0000

Problem Statement

Given an array of integers where every element appears exactly twice except for one element that appears only once, find the single integer using bit manipulation.

Examples

Example 1:

- **Input:** `[2, 2, 1]`
- **Output:** `1`

Example 2:

- **Input:** `[4, 1, 2, 1, 2]`
- **Output:** `4`

Example 3:

- **Input:** `[7, 8, 7]`
- **Output:** `8`

2 1 2

A ^ A = 0

4 1 2 3 3 2 1

0

A ^ 0 = A

7 2 2 8 8 7 9

0

A ^ B ^ C = (A^B) ^ C = A ^ (B ^C)

$$A \wedge B \wedge C = (A \wedge B) \wedge C = A \wedge (B \wedge C)$$

```
public class FindingSingleInteger {
    public static int findSingleInteger(int[] nums){
        int result = 0;
        for(int num : nums){
            result ^= num;
        }
        return result;
    }
    public static void main(String[] args) {
        int[] nums1 = {2,2,1};
        int[] nums2 = {4,1,2,1,2};
        int[] nums3 = {7,8,7};

        System.out.println("Single Integer in nums1 : " +
        findSingleInteger(nums1));
        System.out.println("Single Integer in nums2 : " +
        findSingleInteger(nums2));
        System.out.println("Single Integer in nums3 : " +
        findSingleInteger(nums3));

    }
}
```

```
Single Integer in nums1 : 1
Single Integer in nums2 : 4
Single Integer in nums3 : 8
```

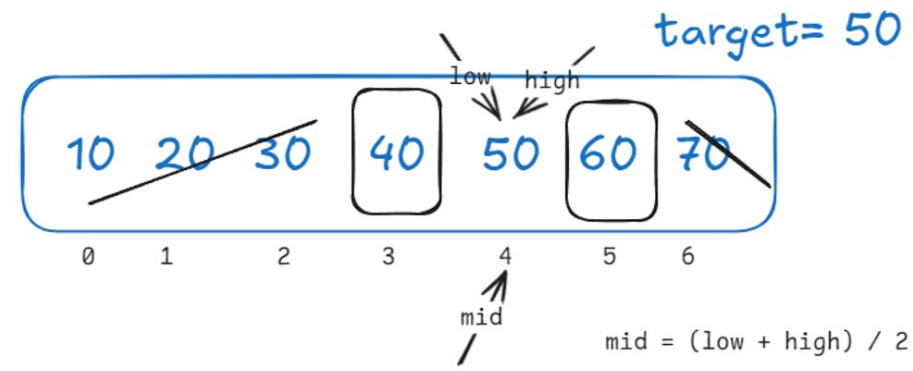
Linear Search

target= 50


```
public class LinearSearch {
    public static int linearSearch(int[] arr , int target){
        for(int i =0; i<arr.length  ; i++){
            if(arr[i] == target){
                return i;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        int[] arr= {10,20,30,40,50,60,70};
```

```
    }
    public static void main(String[] args) {
        int[] arr= {10,20,30,40,50,60,70};
        int target = 50;
        int result = linearSearch(arr , target);
        if ((result == -1)) {
            System.out.println("Element Not Found");
        }else{
            System.out.println("Element Found at index : " + result);
        }
    }
}
```

Binary Search




```
public class BinarySearch {
    public static int binarySearch(int[] arr , int target){
        int low = 0;
        int high = arr.length - 1;

        while(low <= high){
            int mid = (low + high) / 2;
            if(arr[mid] == target){
                return mid;
            }else if(arr[mid] < target){
                low = mid + 1;
            }else {
                high = mid - 1;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        int[] arr= {10,20,30,40,50,60,70};
        int target = 50;
        int result = binarySearch(arr , target);
        if ((result == -1)) {
            System.out.println("Element Not Found");
        }else{
            System.out.println("Element Found at index : " + result);
        }
    }
}
```

Problem Statement: Valid Perfect Square

You are given a positive number n. Your task is to determine whether n is a perfect square. If it is, output "YES". Otherwise, output "NO".

A perfect square is a number that is the square of an integer, i.e., there exists some integer x such that $x \times x = n$.

Input:

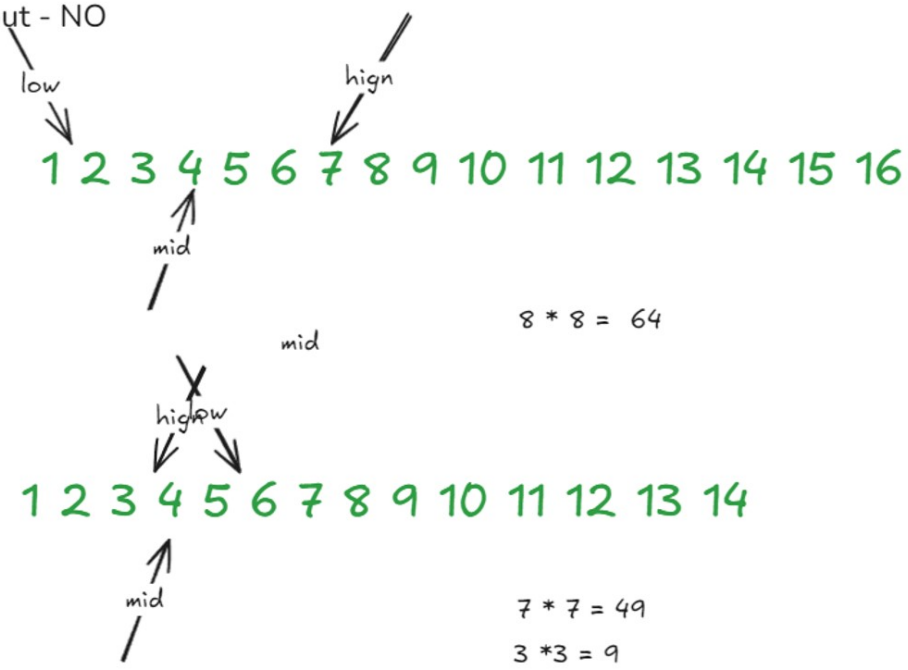
- A single integer n, where $n \geq 1$.

Output:

- A single line containing "YES" if n is a perfect square, otherwise "NO".

input - 16
output - YES

input - 14
output - NO




```
public class ValidSquare {
    public static String validatePerfectSquare(int num){
        int low = 1;
        int high = num;

        while(low <= high){
            int mid = (low + high)/2;
```

```
        while(low <= high){
            int mid = (low + high)/2;
            int midSquared = mid * mid;

            if(midSquared == num){
                return "Yes";
            }else if(midSquared > num){
                high = mid - 1;
            }else{
                low = mid + 1;
            }
        }
        return "No";
    }
    public static void main(String[] args) {
        int n1 = 16;
        System.out.println(validatePerfectSquare(n1));
        int n2 = 14;
        System.out.println(validatePerfectSquare(n2));
    }
}
```

Problem Statement: Count 1 in a Sorted Array

You are given a sorted binary array containing only `0`s and `1`s. Your task is to find the count of `1`s in the array. Since the array is sorted, all `0`s will appear before any `1`s.

Example:
Input 1: arr = [0, 0, 0, 1, 1, 1, 1]
Output 1: 4

Input 2: arr = [1, 1, 1, 1, 1, 1, 1]
Output 2: 7

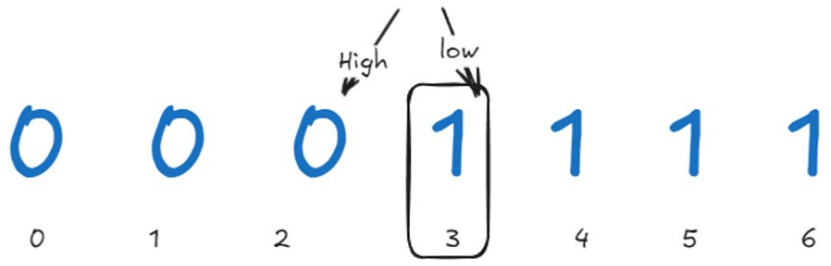
Input 3: arr = [0, 0, 0, 0, 0]
Output 3: 0

Input:

- A sorted array `arr[]` consisting of `0`s and `1`s.

Output:

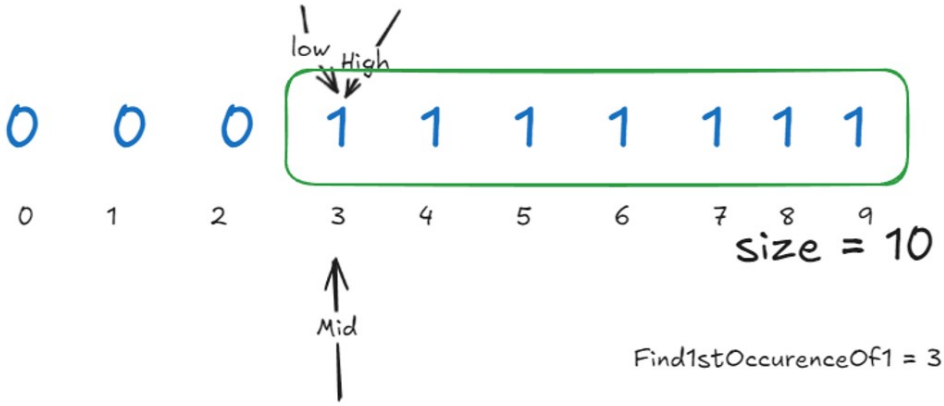
- The count of `1`s in the array.



Find1stOccurenceOf1 = 3

Count= length of an array - firstOccurence =

7 - 3 = 4



Find1stOccurenceOf1 = 3

Count = 10 -3 = 7

```
import java.util.Scanner;
public class CountOneSortedArray {

    public static int countOnes(int[] arr, int n){
        int low = 0;
        int high = n -1;
        int firstOneIndex = -1;

        while(low <= high){
            int mid = (low + high )/2;
            if(arr[mid] == 1){
                firstOneIndex = mid;
                high = mid - 1;
            }else{
                low = mid + 1;
            }
        }
        if(firstOneIndex == -1){
            return 0;
        }
        return n - firstOneIndex;
    }
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        System.out.println("Enter the size of an Array : ");
        int n = scn.nextInt();

        int[] arr = new int[n];
        System.out.println("Enter the elements of an array : ");
```

```
        int[] arr = new int[n];
        System.out.println("Enter the elements of an array : ");
        for(int i =0 ; i<n ; i++){
            arr[i] = scn.nextInt();
        }

        int result = countOnes(arr , n);
        System.out.println("Count of 1's : " + result);
    }
}
```

Completed

Attendance Code: F88FA3DE

Problem Statement: Binary Search

You are given a sorted array of unique integers with a size of `n`. Your task is to find and return the 0-based index of a given key using the binary search algorithm. If the key is not present in the array, return `-1`.

Input Format:

- The first line consists of two space-separated integers, `n` (array size) and `key` (the integer to be found).
- The second line contains `n` space-separated integers representing the sorted array.

Output Format:

- Return the index of the key if present; otherwise, return `-1`.

Example 1:
Input:
7 500
43 120 210 500 582 730 997
Output:
3

Example 2:
Input:
5 96
3 43 55 68 96
Output:
4