Binary search - 1 | Week 10

1.Given a sorted array of n elements and a target ‘x’. Find the last occurrence of ‘x’ in the array. If

```
#include<bits/stdc++.h>
using namespace std;

int lastOccurrence(vector<int>& a, int tgt) {
    int low = 0, high = a.size() - 1;
    int answer = -1;

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

        if (a[mid] == tgt) {
            answer = mid;
            low = mid + 1; // Move to the right half to find the last occurrence
        } else if (a[mid] < tgt) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    return answer;
}

int main() {
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    int tgt;
    cin >> tgt;

    int result = lastOccurrence(a, tgt);
    cout << result << endl;

    return 0;
}
```

2.Given a sorted binary array, efficiently count the total number of 1’s in it.
Input 1 : a = [0,0,0,0,1,1]

Output 1: 2
CODE:-

```
#include <bits/stdc++.h>
using namespace std;

int firstOccurrence(vector<int>& a, int n, int tgt) {
    int low = 0, high = n - 1;
    int ans = n;  // Initialize with n, assuming target might not be found

    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (a[mid] == tgt) {
            ans = mid;
            high = mid - 1;  // Search in the left half to find the first occurrence
        } else {
            low = mid + 1;
        }
    }
    return ans;
}

int main() {
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }

    int firstIndex = firstOccurrence(a, n, 1);
    int countOfOnes = (firstIndex == n) ? 0 : (n - firstIndex);
    cout << countOfOnes << endl;

    return 0;
}
```

3.Given a matrix having 0-1 only where each row is sorted in increasing order, find the row with the maximum number of 1's.

Input matrix : 0 1 1 1
0 0 1 1
1 1 1 1 // this row has maximum 1s
0 0 0 0

Output: 2

CODE:-

```
#include <bits/stdc++.h>
using namespace std;

// Function to find the index of the first occurrence of 1 in a sorted binary array
int firstOccurrence(vector<int>& arr, int low, int high) {
    while (high >= low) {
        int mid = low + (high - low) / 2;
        // Check if the element at middle index is first 1
        if ((mid == 0 || arr[mid - 1] == 0) && arr[mid] == 1)
            return mid;
        // If the element is 0, search in the right half
        else if (arr[mid] == 0)
            low = mid + 1;
        // If element is not first 1, search in the left half
        else
            high = mid - 1;
    }
    return -1;
}

// Function to find the row with the maximum number of 1's
int rowWithMax1s(vector<vector<int>>& matrix) {
    int maxRowIndex = -1;
    int max1sCount = 0;
    int n = matrix.size();
    int m = matrix[0].size();

    for (int i = 0; i < n; i++) {
        int first1Index = firstOccurrence(matrix[i], 0, m - 1);
        if (first1Index != -1) { // if there is at least one '1' in the row
            int count1s = m - first1Index;
            if (count1s > max1sCount) {
                max1sCount = count1s;
        maxRowIndex = i;
      }
    }
  }

  return maxRowIndex;
}

int main() {
  int n, m;
  cin >> n >> m;
  vector<vector<int>> matrix(n, vector<int>(m));

  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
      cin >> matrix[i][j];

  int result = rowWithMax1s(matrix);
  cout << result << endl;

  return 0;
}
```

4.
Given an array of integers nums containing n + 1 integers where each integer is in the range [1, n]
inclusive in sorted order.
There is only one repeated number in nums, return this repeated number.
Input 1: arr[] = {1,2,3,3,4}
Output 1: 3
Input 2: arr[] = {1,2,2,3,4,5}
Output 2: 2
CODE:-

```
#include <bits/stdc++.h>
using namespace std;

int findDuplicate(vector<int>& nums) {
  int low = 1, high = nums.size() - 1, cnt;
  while (low <= high) {
    int mid = low + (high - low) / 2;
    cnt = 0;
    // Count numbers less than or equal to mid
    for (int n : nums) {
      if (n <= mid) ++cnt;
    }
    // Binary search on the left
    if (cnt <= mid)
      low = mid + 1;
    else
      high = mid - 1;
  }
  return low;
}

int main() {
  int n;
  cin >> n;
  vector<int> arr(n + 1);
  for (int i = 0; i < n + 1; ++i) {
    cin >> arr[i];
  }
  cout << findDuplicate(arr) << endl;
  return 0;
}
```

5.

Given a number ‘n’. Predict whether ‘n’ is a valid perfect square or not.

Input 1: n = 36

Output 1: yes

Input 2: n = 45

Output 2: no

Solution:

```
#include <bits/stdc++.h>
using namespace std;

bool isPerfectSquare(int num) {
    int low = 1, high = num;
    while (low <= high) {
        long long mid = low + (high - low) / 2;
        if (mid * mid == num) return true;
        else if (mid * mid < num) low = mid + 1;
        else high = mid - 1;
    }
    return false;
}

int main() {
    int n;
    cin >> n;
    cout << (isPerfectSquare(n) ? "Yes" : "No") << endl;
    return 0;
}
```

6.

You have n coins and you want to build a staircase with these coins. The staircase consists of k
rows where the ith row has exactly i coins. The last row of the staircase may be incomplete.

Given the integer n, return the number of complete rows of the staircase you will build.

Example 1:

Input: n = 5

Output: 2

Explanation: Because the 3rd row is incomplete, we return 2.

Example 2:

Input: n = 8

Output: 3

Explanation: Because the 4th row is incomplete, we return 3.

SOLUTION:-

```
#include <bits/stdc++.h>
using namespace std;

int arrangeCoins(int n) {
    long low = 0;
    long high = n;
    while (low <= high) {
        long mid = low + (high - low) / 2;
        long coinsUsed = mid * (mid + 1) / 2;
        if (coinsUsed == n) {
            return (int)mid;
        }
        if (n < coinsUsed) {
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }
    return (int)high;
}

int main() {
    int n;
    cin >> n;
    cout << arrangeCoins(n) << endl;
    return 0;
}
```