Question 1: Move all # to the front of the string

Problem Statement

Write a function to accept a string and move all # characters to the front while keeping the other characters in their relative positions.

Constraints:

- 1 ≤ length of string ≤ 100
- The string will contain only alphabets and #.

Input:
A string str of length n.

Output:
A modified string with all # at the front.

Example:
Input: Move#Hash#to#Front
Output: ###MoveHashtoFront

Code:

```
#include <iostream>

#include <cstring>

using namespace std;


char* moveHash(char str[], int n) {
   char result[101];
   int hashCount = 0, index = 0;


   for (int i = 0; i < n; i++) {
      if (str[i] == '#') {
         hashCount++;
      } else {
         result[index++] = str[i];
      }
   }
   for (int i = 0; i < hashCount; i++) {
      cout << "#";
   }
```

```
    for (int i = 0; i < index; i++) {
        cout << result[i];
    }
    cout << endl;
    return str;
}

int main() {
    char str[101];
    cin >> str;
    int len = strlen(str);
    moveHash(str, len);
    return 0;
}
```

Question 2: String Compression

Problem Statement

Compress a string such that consecutive repeated characters are represented by the character followed by its count.

Constraints:

- 1 ≤ length of string ≤ 100
- The string will contain only lowercase alphabets.

Input:
A string str of length n.

Output:
A compressed string.

Example:
Input: aabbbbeeeeffggg
Output: a2b4e4f2g3

Code:

```
#include <iostream>
#include <string>
using namespace std;
```

```
string compressString(string str) {
    string result = "";
    int count = 1;

    for (int i = 1; i <= str.length(); i++) {
        if (i < str.length() && str[i] == str[i - 1]) {
            count++;
        } else {
            result += str[i - 1];
            if (count > 1) {
                result += to_string(count);
            }
            count = 1;
        }
    }
    return result;
}

int main() {
    string str;
    cin >> str;
    cout << compressString(str) << endl;
    return 0;
}
```

Question 3: Matrix Spiral Traversal

Problem Statement

Traverse a matrix in a spiral order and print the elements.

Constraints:

- 1 ≤ rows, cols ≤ 100

- Each matrix element is an integer.

Input:
Two integers, rows and cols, followed by a matrix of size rows × cols.

Output:
The elements of the matrix in spiral order.

Example:
Input:

4 4

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

Code:

```
#include <iostream>

using namespace std;


void spiralTraversal(int rows, int cols, int matrix[100][100]) {
   int top = 0, bottom = rows - 1, left = 0, right = cols - 1;


   while (top <= bottom && left <= right) {
      for (int i = left; i <= right; i++) {
         cout << matrix[top][i] << " ";
      }
      top++;


      for (int i = top; i <= bottom; i++) {
         cout << matrix[i][right] << " ";
      }
      right--;


      if (top <= bottom) {
```

```
        for (int i = right; i >= left; i--) {
            cout << matrix[bottom][i] << " ";
        }
        bottom--;
    }

    if (left <= right) {
        for (int i = bottom; i >= top; i--) {
            cout << matrix[i][left] << " ";
        }
        left++;
    }
  }
  cout << endl;
}

int main() {
  int rows, cols, matrix[100][100];
  cin >> rows >> cols;

  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        cin >> matrix[i][j];
    }
  }

  spiralTraversal(rows, cols, matrix);
  return 0;
}
```

Question 4: Count Frequencies of Array Elements

Problem Statement

Given an array of integers, count the frequency of each unique element and print the results.

Constraints:

- 1 ≤ n ≤ 100
- Array elements range from -1000 to 1000.

Input:
An integer n, followed by n integers (the array).

Output:
Frequency of each unique element in the format:
element occurs x times.

Example:
Input:

10

1 2 3 3 4 1 4 5 1 2

Output:

1 occurs 3 times

2 occurs 2 times

3 occurs 2 times

4 occurs 2 times

5 occurs 1 times

Code:

```
#include <iostream>

#include <map>

using namespace std;


void countFrequencies(int arr[], int n) {

   map<int, int> freq;

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

      freq[arr[i]]++;

   }

   for (auto it : freq) {

      cout << it.first << " occurs " << it.second << " times" << endl;
```

```
    }
}

int main() {
    int n;
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    countFrequencies(arr, n);
    return 0;
}
```

Question 5: Solve Polynomial Equation

Problem Statement

Write a function to evaluate the equation:
a^3 + a^2*b + 2a^2*b + 2a*b^2 + a*b^2 + b^3.

Constraints:

- 1 ≤ a, b ≤ 100

Input:
Two integers a and b.

Output:
The result of the polynomial equation.

Example:
Input:

2 3

Output:

125

Code:

```
#include <iostream>
#include <cmath>
```

```
using namespace std;

int evaluateEquation(int a, int b) {
    return pow(a + b, 3);
}

int main() {
    int a, b;
    cin >> a >> b;
    cout << evaluateEquation(a, b) << endl;
    return 0;
}
```

Question 6: Find the Number of Tyres

Problem Statement

For multiple dealerships, calculate the total number of tyres based on the number of cars and bikes.

Constraints:

- $1 \leq t \leq 100$ (number of dealerships)
- $0 \leq$ cars, bikes ≤ 100

Input:
An integer t, followed by t lines containing two integers each (cars and bikes).

Output:
Total number of tyres for each dealership.

Example:
Input:

3

4 2

4 0

1 2

Output:

20

16

8

Code:

```
#include <iostream>

using namespace std;


void calculateTyres(int t, int data[][2]) {
   for (int i = 0; i < t; i++) {
      int cars = data[i][0], bikes = data[i][1];
      cout << (cars * 4 + bikes * 2) << endl;
   }
}


int main() {
   int t;
   cin >> t;
   int data[t][2];
   for (int i = 0; i < t; i++) {
      cin >> data[i][0] >> data[i][1];
   }
   calculateTyres(t, data);
   return 0;
}
```

Question 7: Largest Element in Array

Problem Statement

Find the largest element in an array of integers.

Constraints:

- 1 ≤ n ≤ 100
- Array elements range from -10^6 to 10^6.

Input:
An integer n, followed by n integers (the array).

Output:
The largest element in the array.

Example:
Input:

5

1 -4 3 6 2

Output:

6

Code:

```
#include <iostream>

#include <climits>

using namespace std;


int findLargest(int arr[], int n) {
   int largest = INT_MIN;
   for (int i = 0; i < n; i++) {
      if (arr[i] > largest) {
         largest = arr[i];
      }
   }
   return largest;
}


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

}

Question 8: Sum of All Subarrays

Problem Statement

Write a program to find the sum of all possible subarrays of an array. A subarray is a contiguous part of the array.

Constraints:

- $1 \leq n \leq 100$
- Array elements range from -1000 to 1000.

Input:
An integer n, followed by n integers (the array).

Output:
Sum of all possible subarrays.

Example:
Input:

3

1 2 3

Output:

20

Explanation:
The subarrays are:
[1], [2], [3], [1, 2], [2, 3], [1, 2, 3]
Sum: 1 + 2 + 3 + (1 + 2) + (2 + 3) + (1 + 2 + 3) = 20

Code:

```
#include <iostream>

using namespace std;


int sumOfSubarrays(int arr[], int n) {

   int sum = 0;

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

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

         for (int k = i; k <= j; k++) {

            sum += arr[k];
```

```
        }
      }
    }
    return sum;
}

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

Question 9: Rotate an Array

Problem Statement

Write a function to rotate an array k times to the right.

Constraints:

- 1 ≤ n ≤ 100
- 0 ≤ k ≤ n
- Array elements range from -1000 to 1000.

Input:
An integer n, followed by n integers (the array), and an integer k.

Output:
The rotated array after k rotations.

Example:
Input:

5

1 2 3 4 5

2

Output:

4 5 1 2 3

Code:

```
#include <iostream>
using namespace std;

void rotateArray(int arr[], int n, int k) {
    int temp[k];
    for (int i = 0; i < k; i++) {
        temp[i] = arr[n - k + i];
    }
    for (int i = n - 1; i >= k; i--) {
        arr[i] = arr[i - k];
    }
    for (int i = 0; i < k; i++) {
        arr[i] = temp[i];
    }
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int n, k;
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
```

```
    cin >> k;
    rotateArray(arr, n, k);
    return 0;
}
```

Question 10: Longest Substring Without Repeating Characters

Problem Statement

Given a string, find the length of the longest substring without repeating characters.

Constraints:

- 1 ≤ length of string ≤ 1000
- The string consists of printable ASCII characters.

Input:
A string s.

Output:
The length of the longest substring without repeating characters.

Example:
Input:

abcabcbb

Output:

3

Code:

```
#include <iostream>
#include <unordered_map>
#include <algorithm>
using namespace std;

int longestUniqueSubstring(string s) {
    unordered_map<char, int> map;
    int maxLength = 0, start = 0;

    for (int end = 0; end < s.length(); end++) {
        if (map.find(s[end]) != map.end()) {
```

```
            start = max(start, map[s[end]] + 1);
        }
        map[s[end]] = end;
        maxLength = max(maxLength, end - start + 1);
    }
    return maxLength;
}

int main() {
    string s;
    cin >> s;
    cout << longestUniqueSubstring(s) << endl;
    return 0;
}
```

Question 11: Find the Missing Number in an Array

Problem Statement

Given an array of n-1 numbers in the range from 1 to n, find the missing number.

Constraints:

- 1 ≤ n ≤ 1000
- Array elements range from 1 to n and only one element is missing.

Input:
An integer n, followed by n-1 integers (the array).

Output:
The missing number.

Example:
Input:

5

1 2 4 5

Output:

3

Code:

```
#include <iostream>
using namespace std;

int findMissingNumber(int arr[], int n) {
    int totalSum = n * (n + 1) / 2;
    int arrSum = 0;
    for (int i = 0; i < n - 1; i++) {
        arrSum += arr[i];
    }
    return totalSum - arrSum;
}

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

Question 12: Merge Two Sorted Arrays

Problem Statement

Write a function to merge two sorted arrays into one sorted array.

Constraints:

- 1 ≤ n, m ≤ 1000
- Array elements range from -1000 to 1000.

Input:
Two sorted arrays arr1[] and arr2[] of sizes n and m.

Output:
The merged sorted array.

Example:
Input:

3

1 3 5

3

2 4 6

Output:

1 2 3 4 5 6

Code:

```
#include <iostream>

using namespace std;


void mergeArrays(int arr1[], int arr2[], int n, int m) {
   int i = 0, j = 0;
   while (i < n && j < m) {
      if (arr1[i] < arr2[j]) {
         cout << arr1[i] << " ";
         i++;
      } else {
         cout << arr2[j] << " ";
         j++;
      }
   }
   while (i < n) {
      cout << arr1[i] << " ";
      i++;
   }
   while (j < m) {
      cout << arr2[j] << " ";
      j++;
```

```
    }
    cout << endl;
}

int main() {
    int n, m;
    cin >> n;
    int arr1[n];
    for (int i = 0; i < n; i++) {
        cin >> arr1[i];
    }
    cin >> m;
    int arr2[m];
    for (int i = 0; i < m; i++) {
        cin >> arr2[i];
    }
    mergeArrays(arr1, arr2, n, m);
    return 0;
}
```

Question 13: Maximum Product Subarray

Problem Statement

Given an integer array, find the contiguous subarray (containing at least one number) that has the largest product.

Constraints:

- $1 \leq n \leq 1000$
- $-1000 \leq arr[i] \leq 1000$.

Input:
An integer n, followed by n integers (the array).

Output:
The maximum product of a contiguous subarray.

Example:
Input:

5

2 3 -2 4 -1

Output:

48

Explanation:
The subarray [2, 3, -2, 4] has the largest product 2 * 3 * -2 * 4 = 48.

Code:

```
#include <iostream>

#include <algorithm>

using namespace std;


int maxProductSubarray(int arr[], int n) {
   int maxProduct = arr[0], minProduct = arr[0], result = arr[0];


   for (int i = 1; i < n; i++) {
      if (arr[i] < 0) {
         swap(maxProduct, minProduct);
      }


      maxProduct = max(arr[i], maxProduct * arr[i]);
      minProduct = min(arr[i], minProduct * arr[i]);


      result = max(result, maxProduct);
   }
   return result;
}


int main() {
   int n;
   cin >> n;
```

```
    int arr[n];
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    cout << maxProductSubarray(arr, n) << endl;
    return 0;
}
```

Question 14: Find the Duplicate in an Array

Problem Statement

You are given an array of n elements, where each element is in the range 1 to n-1. Find the one element that is repeated.

Constraints:

- $1 \leq n \leq 1000$
- Array elements range from 1 to n-1.

Input:
An integer n, followed by n integers (the array).

Output:
The duplicate element.

Example:
Input:

5

1 2 3 4 4

Output:

4

Code:

```
#include <iostream>
using namespace std;

int findDuplicate(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        if (arr[abs(arr[i])] >= 0)
```

```
            arr[abs(arr[i])] = -arr[abs(arr[i])];
        else
            return abs(arr[i]);
    }
    return -1; // No duplicate found
}

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

Question 15: Reverse Words in a String

Problem Statement

Given a string, reverse the order of words in it.

Constraints:

- 1 ≤ length of string ≤ 1000.

Input:
A string s containing words separated by spaces.

Output:
The string with words reversed.

Example:
Input:

the sky is blue

Output:

blue is sky the

Code:

```
#include <iostream>
#include <sstream>
#include <vector>
#include <algorithm>
using namespace std;

string reverseWords(string s) {
    stringstream ss(s);
    vector<string> words;
    string word;

    while (ss >> word) {
        words.push_back(word);
    }

    reverse(words.begin(), words.end());

    string result;
    for (const string& w : words) {
        result += w + " ";
    }
    result.pop_back(); // Remove trailing space
    return result;
}

int main() {
    string s;
    getline(cin, s);
    cout << reverseWords(s) << endl;
    return 0;
```

}

Question 16: Find Intersection of Two Arrays

Problem Statement

Given two arrays, find the intersection of these two arrays (elements that are present in both arrays).

Constraints:

- 1 ≤ n, m ≤ 1000
- Array elements range from -1000 to 1000.

Input:
Two arrays arr1[] and arr2[] of sizes n and m.

Output:
The intersection of the two arrays.

Example:
Input:

5

1 2 2 1 3

4

2 2 3 4

Output:

2 2 3

Code:

```
#include <iostream>
#include <unordered_map>
#include <vector>
using namespace std;

vector<int> intersectArrays(int arr1[], int arr2[], int n, int m) {
    unordered_map<int, int> freq;
    vector<int> result;

    for (int i = 0; i < n; i++) {
        freq[arr1[i]]++;
```

```
    }

    for (int i = 0; i < m; i++) {
        if (freq[arr2[i]] > 0) {
            result.push_back(arr2[i]);
            freq[arr2[i]]--;
        }
    }
    return result;
}

int main() {
    int n, m;
    cin >> n;
    int arr1[n];
    for (int i = 0; i < n; i++) {
        cin >> arr1[i];
    }
    cin >> m;
    int arr2[m];
    for (int i = 0; i < m; i++) {
        cin >> arr2[i];
    }

    vector<int> result = intersectArrays(arr1, arr2, n, m);
    for (int val : result) {
        cout << val << " ";
    }
    cout << endl;

    return 0;
```

```
}
```

Question 17: Sum of Diagonals in a Matrix

Problem Statement

Given a square matrix, find the sum of the primary and secondary diagonals.

Constraints:

- $1 \leq n \leq 1000$
- Matrix elements range from -1000 to 1000.

Input:
An integer n, followed by n x n integers (the matrix).

Output:
The sum of the diagonals.

Example:
Input:

3

1 2 3

4 5 6

7 8 9

Output:

30

Code:

```
#include <iostream>

using namespace std;


int sumOfDiagonals(int arr[][1000], int n) {
   int sum = 0;
   for (int i = 0; i < n; i++) {
      sum += arr[i][i]; // Primary diagonal
      sum += arr[i][n - 1 - i]; // Secondary diagonal
   }
   return sum;
}
```

```
int main() {
    int n;
    cin >> n;
    int arr[1000][1000];

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> arr[i][j];
        }
    }

    cout << sumOfDiagonals(arr, n) << endl;
    return 0;
}
```

Question 18: Count Set Bits

Problem Statement

Write a program to count the number of set bits (1s) in the binary representation of an integer.

Constraints:

- $1 \leq n \leq 1000$.

Input:
An integer n.

Output:
The number of set bits (1s) in the binary representation of n.

Example:
Input:

5

Output:

2

Code:

```
#include <iostream>
```

```
using namespace std;


int countSetBits(int n) {
   int count = 0;
   while (n) {
      count += n & 1;
      n >>= 1;
   }
   return count;
}


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

Question 19: Find the Missing Number in an Array

Problem Statement

You are given an array of n integers containing numbers from 1 to n+1, but one of the numbers is missing. Your task is to find the missing number.

Constraints:

- 1 ≤ n ≤ 1000
- Array elements are in the range from 1 to n+1.

Input:
An integer n, followed by n integers (the array).

Output:
The missing number.

Example:
Input:

5

1 2 4 5

Output:

3

Code:

```
#include <iostream>
using namespace std;

int findMissingNumber(int arr[], int n) {
   int sum = (n + 1) * (n + 2) / 2;
   for (int i = 0; i < n; i++) {
      sum -= arr[i];
   }
   return sum;
}

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

Question 20: Longest Substring Without Repeating Characters

Problem Statement

Given a string, find the length of the longest substring without repeating characters.

Constraints:

- 1 ≤ length of string ≤ 1000.

Input:
A string s containing only alphabets and numbers.

Output:
The length of the longest substring without repeating characters.

Example:
Input:

abcabcbb

Output:

3

Code:

```
#include <iostream>

#include <unordered_set>

using namespace std;


int lengthOfLongestSubstring(string s) {

   unordered_set<char> set;

   int left = 0, right = 0, maxLength = 0;


   while (right < s.length()) {

      if (set.find(s[right]) == set.end()) {

         set.insert(s[right]);

         right++;

         maxLength = max(maxLength, right - left);

      } else {

         set.erase(s[left]);

         left++;

      }

   }

   return maxLength;

}


int main() {
```

```
    string s;
    cin >> s;
    cout << lengthOfLongestSubstring(s) << endl;
    return 0;
}
```

Question 21: Find the Kth Smallest Element in an Array

Problem Statement

Given an unsorted array, find the Kth smallest element.

Constraints:

- 1 ≤ n ≤ 1000
- 1 ≤ k ≤ n.

Input:
An integer n followed by n integers (the array), and an integer k.

Output:
The Kth smallest element in the array.

Example:
Input:

6

12 3 5 7 19 1

2

Output:

3

Code:

```
#include <iostream>
#include <algorithm>
using namespace std;

int findKthSmallest(int arr[], int n, int k) {
    sort(arr, arr + n);
    return arr[k - 1];
}
```

```
int main() {
    int n, k;
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    cin >> k;
    cout << findKthSmallest(arr, n, k) << endl;
    return 0;
}
```

Question 22: Rotate a Matrix 90 Degrees

Problem Statement

Given an n x n matrix, rotate it 90 degrees clockwise.

Constraints:

- 1 ≤ n ≤ 1000
- The matrix elements range from -1000 to 1000.

Input:
An integer n, followed by an n x n matrix.

Output:
The matrix rotated by 90 degrees.

Example:
Input:

3

1 2 3

4 5 6

7 8 9

Output:

7 4 1

8 5 2

9 6 3

Code:

```
#include <iostream>
using namespace std;

void rotateMatrix(int matrix[][1000], int n) {
    for (int i = 0; i < n / 2; i++) {
        for (int j = i; j < n - i - 1; j++) {
            int temp = matrix[i][j];
            matrix[i][j] = matrix[n - j - 1][i];
            matrix[n - j - 1][i] = matrix[n - i - 1][n - j - 1];
            matrix[n - i - 1][n - j - 1] = matrix[j][n - i - 1];
            matrix[j][n - i - 1] = temp;
        }
    }
}

int main() {
    int n;
    cin >> n;
    int matrix[1000][1000];

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

    rotateMatrix(matrix, n);

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

```
        for (int j = 0; j < n; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Question 23: Subarray Sum Equals K

Problem Statement

Given an array of integers and an integer k, find the total number of contiguous subarrays whose sum equals k.

Constraints:

- 1 ≤ n ≤ 1000
- -1000 ≤ arr[i] ≤ 1000.

Input:
An integer n, followed by n integers (the array), and an integer k.

Output:
The total number of subarrays whose sum equals k.

Example:
Input:

5

1 1 1 1 1

3

Output:

3

Code:

```
#include <iostream>
#include <unordered_map>
using namespace std;

int subarraySum(int arr[], int n, int k) {
```

```
    unordered_map<int, int> map;
    map[0] = 1;
    int sum = 0, count = 0;

    for (int i = 0; i < n; i++) {
        sum += arr[i];

        if (map.find(sum - k) != map.end()) {
            count += map[sum - k];
        }

        map[sum]++;
    }

    return count;
}

int main() {
    int n, k;
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    cin >> k;
    cout << subarraySum(arr, n, k) << endl;
    return 0;
}
```

Question 24: Find Majority Element

Problem Statement

Given an array of size n, find the majority element in the array. The majority element is the element that appears more than n / 2 times in the array.

Constraints:

- 1 ≤ n ≤ 1000.

Input:
An integer n, followed by n integers (the array).

Output:
The majority element, or -1 if no majority element exists.

Example:
Input:

5

3 3 4 2 3

Output:

3

Code:

```
#include <iostream>

using namespace std;


int majorityElement(int arr[], int n) {
   int count = 0, candidate = -1;


   for (int i = 0; i < n; i++) {
      if (count == 0) {
         candidate = arr[i];
      }
      count += (arr[i] == candidate) ? 1 : -1;
   }


   count = 0;
   for (int i = 0; i < n; i++) {
      if (arr[i] == candidate) {
         count++;
```

```
        }
    }

    return (count > n / 2) ? candidate : -1;
}

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

Question 25: Maximum Product Subarray

Problem Statement

Given an integer array nums, find the contiguous subarray within the array (containing at least one number) which has the largest product.

Constraints:

- 1 ≤ n ≤ 1000
- -10^3 ≤ nums[i] ≤ 10^3.

Input:
An integer n, followed by n integers (the array).

Output:
The maximum product of any contiguous subarray.

Example:
Input:

5

2 3 -2 4 -1

Output:

48

Code:

```
#include <iostream>
#include <algorithm>
using namespace std;

int maxProduct(int arr[], int n) {
    int maxProd = arr[0], minProd = arr[0], result = arr[0];

    for (int i = 1; i < n; i++) {
        if (arr[i] < 0) {
            swap(maxProd, minProd);
        }

        maxProd = max(arr[i], maxProd * arr[i]);
        minProd = min(arr[i], minProd * arr[i]);

        result = max(result, maxProd);
    }

    return result;
}

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

```
    cout << maxProduct(arr, n) << endl;
    return 0;
}
```

Question 26: Longest Palindromic Substring

Problem Statement

Given a string s, return the longest palindromic substring in s.

Constraints:

- 1 ≤ length of s ≤ 1000.

Input:
A string s.

Output:
The longest palindromic substring.

Example:
Input:

babad

Output:

bab

Code:

```
#include <iostream>
using namespace std;

string expandAroundCenter(string s, int left, int right) {
    while (left >= 0 && right < s.size() && s[left] == s[right]) {
        left--;
        right++;
    }
    return s.substr(left + 1, right - left - 1);
}

string longestPalindrome(string s) {
    string longest = "";
```

```
    for (int i = 0; i < s.size(); i++) {
        string odd = expandAroundCenter(s, i, i);
        string even = expandAroundCenter(s, i, i + 1);

        if (odd.size() > longest.size()) longest = odd;
        if (even.size() > longest.size()) longest = even;
    }

    return longest;
}

int main() {
    string s;
    cin >> s;
    cout << longestPalindrome(s) << endl;
    return 0;
}
```

Question 27: Merge Intervals

Problem Statement

Given a collection of intervals, merge all overlapping intervals.

Constraints:

- 1 ≤ n ≤ 1000.
- Intervals are represented as pairs of integers, and the intervals are not necessarily sorted.

Input:
An integer n, followed by n pairs of integers.

Output:
A list of merged intervals.

Example:
Input:

5

1 3

2 4

6 8

5 7

9 10

Output:

1 4

5 8

9 10

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

vector<pair<int, int>> mergeIntervals(vector<pair<int, int>>& intervals) {
    if (intervals.empty()) return {};

    sort(intervals.begin(), intervals.end());
    vector<pair<int, int>> result;

    result.push_back(intervals[0]);

    for (int i = 1; i < intervals.size(); i++) {
        if (result.back().second >= intervals[i].first) {
            result.back().second = max(result.back().second, intervals[i].second);
        } else {
            result.push_back(intervals[i]);
        }
    }
```

```
    return result;
}

int main() {
    int n;
    cin >> n;
    vector<pair<int, int>> intervals(n);

    for (int i = 0; i < n; i++) {
        cin >> intervals[i].first >> intervals[i].second;
    }

    vector<pair<int, int>> result = mergeIntervals(intervals);

    for (auto interval : result) {
        cout << interval.first << " " << interval.second << endl;
    }

    return 0;
}
```

Question 28: 3Sum Problem

Problem Statement

Given an array nums, return all unique triplets in the array which give the sum of zero.

Constraints:

- $1 \leq n \leq 1000$.
- The array may contain duplicates.

Input:
An integer n, followed by n integers (the array).

Output:
A list of all unique triplets that sum to zero.

Example:

Input:

6

-1 0 1 2 -1 -4

Output:

-1 -1 2

-1 0 1

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

vector<vector<int>> threeSum(vector<int>& nums) {
    vector<vector<int>> result;
    sort(nums.begin(), nums.end());

    for (int i = 0; i < nums.size(); i++) {
        if (i > 0 && nums[i] == nums[i - 1]) continue;

        int left = i + 1, right = nums.size() - 1;
        while (left < right) {
            int sum = nums[i] + nums[left] + nums[right];
            if (sum == 0) {
                result.push_back({nums[i], nums[left], nums[right]});
                while (left < right && nums[left] == nums[left + 1]) left++;
                while (left < right && nums[right] == nums[right - 1]) right--;
                left++;
                right--;
            } else if (sum < 0) {
                left++;
```

```
            } else {
                right--;
            }
        }
    }

    return result;
}

int main() {
    int n;
    cin >> n;
    vector<int> nums(n);

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

    vector<vector<int>> result = threeSum(nums);

    for (auto triplet : result) {
        cout << triplet[0] << " " << triplet[1] << " " << triplet[2] << endl;
    }

    return 0;
}
```

Question 29: Coin Change Problem

Problem Statement

You are given an array of integers representing coins of different denominations and an integer amount. Write a function to compute the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

Constraints:

- 1 ≤ amount ≤ 1000.
- The number of coin denominations can vary.

Input:
An integer amount, followed by a list of coin denominations.

Output:
The minimum number of coins needed to make up the amount. If not possible, return -1.

Example:
Input:

11

1 2 5

Output:

3

Code:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;


int coinChange(vector<int>& coins, int amount) {

   vector<int> dp(amount + 1, amount + 1);

   dp[0] = 0;


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

      for (int coin : coins) {

         if (i - coin >= 0) {

            dp[i] = min(dp[i], dp[i - coin] + 1);

         }

      }

   }


   return dp[amount] > amount ? -1 : dp[amount];
```

```
}

int main() {
   int amount, n;
   cin >> amount;
   cin >> n;
   vector<int> coins(n);

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

   cout << coinChange(coins, amount) << endl;
   return 0;
}
```

Question 30: Find All Anagrams in a String

Problem Statement

Given a string s and a string p, find all the start indices of p's anagrams in s.

Constraints:

- 1 ≤ s.length, p.length ≤ 1000.

Input:
Strings s and p.

Output:
A list of starting indices of p's anagrams in s.

Example:
Input:

cbaebabacd

abc

Output:

0 6

Code:

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;

vector<int> findAnagrams(string s, string p) {
    vector<int> result;
    if (s.size() < p.size()) return result;

    unordered_map<char, int> pMap, sMap;

    for (char c : p) pMap[c]++;

    for (int i = 0; i < p.size(); i++) {
        sMap[s[i]]++;
    }

    if (sMap == pMap) result.push_back(0);

    for (int i = p.size(); i < s.size(); i++) {
        sMap[s[i]]++;
        sMap[s[i - p.size()]]--;

        if (sMap[s[i - p.size()]] == 0) sMap.erase(s[i - p.size()]);

        if (sMap == pMap) result.push_back(i - p.size() + 1);
    }

    return result;
```

```
}

int main() {
    string s, p;
    cin >> s >> p;

    vector<int> result = findAnagrams(s, p);

    for (int idx : result) {
        cout << idx << " ";
    }
    cout << endl;

    return 0;
}
```

Question 31: Combination Sum

Problem Statement

Given an array of distinct integers candidates and a target integer target, find all unique combinations in candidates where the sum of the numbers is equal to the target.

Constraints:

- 1 ≤ candidates.length ≤ 100.
- 1 ≤ candidates[i] ≤ 50.
- 1 ≤ target ≤ 500.

Input:
An integer target, followed by a list of integers candidates.

Output:
A list of all unique combinations of candidates that sum up to the target.

Example:
Input:

7

2 3 6 7

Output:

2 2 3

7

Code:

```
#include <iostream>
#include <vector>
using namespace std;

void combinationSumHelper(vector<int>& candidates, int target, int start, vector<int>& current, vector<vector<int>>& result) {
    if (target == 0) {
        result.push_back(current);
        return;
    }

    for (int i = start; i < candidates.size(); i++) {
        if (candidates[i] > target) continue;
        current.push_back(candidates[i]);
        combinationSumHelper(candidates, target - candidates[i], i, current, result);
        current.pop_back();
    }
}

vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
    vector<vector<int>> result;
    vector<int> current;
    combinationSumHelper(candidates, target, 0, current, result);
    return result;
}

int main() {
```

```
    int target, n;
    cin >> target >> n;
    vector<int> candidates(n);

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

    vector<vector<int>> result = combinationSum(candidates, target);

    for (auto combination : result) {
        for (int num : combination) {
            cout << num << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Question 32: Valid Sudoku

Problem Statement

Determine if a 9x9 Sudoku board is valid. Only the following conditions need to be checked:

- Each row must contain the digits 1-9 without repetition.
- Each column must contain the digits 1-9 without repetition.
- Each of the 9 subgrids (3x3) must contain the digits 1-9 without repetition.

Constraints:

- The board is a 9x9 matrix.

Input:
A 9x9 matrix representing the Sudoku board.

Output:
True if the board is valid, False otherwise.

Example:

Input:

5 3 . . 7 . . . .

6 . . 1 9 5 . . .

. 9 8 . . . . 6 .

8 . . . 6 . . . 3

4 . . 8 . 3 . . 1

7 . . . 2 . . . 6

. 6 . . . . 2 8 .

. . . 4 1 9 . . 5

. . . . 8 . . 7 9

Output:

True

Code:

```
#include <iostream>
#include <vector>
#include <unordered_set>
using namespace std;

bool isValidSudoku(vector<vector<char>>& board) {
   for (int i = 0; i < 9; i++) {
      unordered_set<char> rowSet, colSet, boxSet;
      for (int j = 0; j < 9; j++) {
         if (board[i][j] != '.' && !rowSet.insert(board[i][j]).second) return false;
         if (board[j][i] != '.' && !colSet.insert(board[j][i]).second) return false;

         int row = 3 * (i / 3) + j / 3;
         int col = 3 * (i % 3) + j % 3;
         if (board[row][col] != '.' && !boxSet.insert(board[row][col]).second) return false;
      }
   }
```

```
    return true;
}

int main() {
    vector<vector<char>> board(9, vector<char>(9));

    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            cin >> board[i][j];
        }
    }

    if (isValidSudoku(board)) cout << "True" << endl;
    else cout << "False" << endl;

    return 0;
}
```

Question 33: Subarray Sum Equals K

Problem Statement

Given an array of integers nums and an integer k, return the total number of continuous subarrays whose sum equals k.

Constraints:

- 1 ≤ nums.length ≤ 2000.
- -1000 ≤ nums[i] ≤ 1000.
- -10^7 ≤ k ≤ 10^7.

Input:
An integer k, followed by an array of integers nums.

Output:
The number of subarrays whose sum equals k.

Example:
Input:

7

1 2 3 4 5

Output:

3

Code:

```
#include <iostream>
#include <unordered_map>
#include <vector>
using namespace std;

int subarraySum(vector<int>& nums, int k) {
    unordered_map<int, int> prefixSum;
    int count = 0, sum = 0;

    prefixSum[0] = 1;

    for (int num : nums) {
        sum += num;
        if (prefixSum.find(sum - k) != prefixSum.end()) {
            count += prefixSum[sum - k];
        }
        prefixSum[sum]++;
    }

    return count;
}

int main() {
    int k, n;
    cin >> k >> n;
    vector<int> nums(n);
```

```
    for (int i = 0; i < n; i++) {
        cin >> nums[i];
    }

    cout << subarraySum(nums, k) << endl;
    return 0;
}
```

Question 34: Word Break

Problem Statement

Given a string s and a dictionary of strings wordDict, return true if s can be segmented into a space-separated sequence of one or more dictionary words.

Constraints:

- 1 ≤ s.length ≤ 300.
- 1 ≤ wordDict.length ≤ 1000.
- 1 ≤ wordDict[i].length ≤ 10.

Input:
A string s followed by a list of strings wordDict.

Output:
True if s can be segmented into a sequence of one or more words from wordDict, otherwise False.

Example:
Input:

leetcode

["leet", "code"]

Output:

True

Code:

```
#include <iostream>
#include <vector>
#include <unordered_set>
using namespace std;
```

```
bool wordBreak(string s, unordered_set<string>& wordDict) {
    vector<bool> dp(s.length() + 1, false);
    dp[0] = true;

    for (int i = 1; i <= s.length(); i++) {
        for (int j = i - 1; j >= 0; j--) {
            if (dp[j] && wordDict.find(s.substr(j, i - j)) != wordDict.end()) {
                dp[i] = true;
                break;
            }
        }
    }

    return dp[s.length()];
}

int main() {
    string s;
    int n;
    cin >> s >> n;
    unordered_set<string> wordDict;

    for (int i = 0; i < n; i++) {
        string word;
        cin >> word;
        wordDict.insert(word);
    }

    if (wordBreak(s, wordDict)) cout << "True" << endl;
    else cout << "False" << endl;
```

```
    return 0;
}
```

Question 35: Clone Graph

Problem Statement

Clone an undirected graph. Each node in the graph contains a label and a list of its neighbors.

Constraints:

- The graph is a simple undirected graph.
- 1 ≤ node.val ≤ 100.

Input:
A graph represented by an adjacency list.

Output:
The deep copy of the graph.

Example:
Input:

1 -> 2 -> 3

Output:

1 -> 2 -> 3

Code:

```
#include <iostream>
#include <unordered_map>
#include <vector>
#include <queue>
using namespace std;

class Node {
public:
    int val;
    vector<Node*> neighbors;
    Node(int x) : val(x) {}
};
```

```
Node* cloneGraph(Node* node) {
    if (!node) return nullptr;

    unordered_map<Node*, Node*> visited;
    queue<Node*> q;
    q.push(node);
    visited[node] = new Node(node->val);

    while (!q.empty()) {
        Node*
current = q.front(); q.pop();
    for (Node* neighbor : current->neighbors) {
        if (visited.find(neighbor) == visited.end()) {
            visited[neighbor] = new Node(neighbor->val);
            q.push(neighbor);
        }
        visited[current]->neighbors.push_back(visited[neighbor]);
    }
}

return visited[node];
}
int main() { int n; cin >> n; vector<Node*> nodes(n);
for (int i = 0; i < n; i++) {
    nodes[i] = new Node(i + 1);
}

for (int i = 0; i < n; i++) {
    int m;
    cin >> m;
```

```
    for (int j = 0; j < m; j++) {
        int neighbor;
        cin >> neighbor;
        nodes[i]->neighbors.push_back(nodes[neighbor - 1]);
    }
}

Node* clonedGraph = cloneGraph(nodes[0]);

cout << "Graph cloned!" << endl;

return 0;
}
```

Question 36: Maximum Subarray Product

Problem Statement

Given an integer array nums, find the contiguous subarray within an array (containing at least one number) which has the largest product.

Constraints:

- 1 ≤ nums.length ≤ 2 * 10^4.
- -10 ≤ nums[i] ≤ 10.

Input:
An array of integers nums.

Output:
The maximum product of any subarray.

Example:
Input:

[2, 3, -2, 4]

Output:

6

Code:

```
#include <iostream>
```

```
#include <vector>
#include <algorithm>
using namespace std;

int maxProduct(vector<int>& nums) {
    int maxProd = nums[0], minProd = nums[0], result = nums[0];

    for (int i = 1; i < nums.size(); i++) {
        if (nums[i] < 0) swap(maxProd, minProd);

        maxProd = max(nums[i], maxProd * nums[i]);
        minProd = min(nums[i], minProd * nums[i]);

        result = max(result, maxProd);
    }

    return result;
}

int main() {
    int n;
    cin >> n;
    vector<int> nums(n);

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

    cout << maxProduct(nums) << endl;

    return 0;
```

}

Question 37: Largest Rectangle in Histogram

Problem Statement

Given an array of integers representing the heights of bars in a histogram, find the area of the largest rectangle in the histogram.

Constraints:

- 1 ≤ heights.length ≤ 10000.
- 0 ≤ heights[i] ≤ 10000.

Input:
An array heights[] representing the heights of the histogram bars.

Output:
The area of the largest rectangle in the histogram.

Example:
Input:

[2, 1, 5, 6, 2, 3]

Output:

10

Code:

```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;

int largestRectangleArea(vector<int>& heights) {
    stack<int> s;
    heights.push_back(0);
    int maxArea = 0;

    for (int i = 0; i < heights.size(); i++) {
        while (!s.empty() && heights[s.top()] > heights[i]) {
            int h = heights[s.top()];
```

```
            s.pop();
            int width = s.empty() ? i : i - s.top() - 1;
            maxArea = max(maxArea, h * width);
        }
        s.push(i);
    }

    return maxArea;
}

int main() {
    int n;
    cin >> n;
    vector<int> heights(n);

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

    cout << largestRectangleArea(heights) << endl;

    return 0;
}
```

Question 38: N-Queens Problem

Problem Statement

The N-Queens puzzle is the problem of placing N queens on an N×N chessboard so that no two queens threaten each other. Implement a function to return all distinct solutions to the N-Queens problem.

Constraints:

- $1 \leq N \leq 10$.

Input:
An integer N, representing the number of queens and the size of the board.

Output:
All distinct solutions to the N-Queens problem.

Example:
Input:

4

Output:

[[".Q..", "...Q", "Q...", "..Q."]]

Code:

```
#include <iostream>

#include <vector>

using namespace std;


void solveNQueensHelper(int N, int row, vector<string>& board, vector<vector<string>>& result,
vector<bool>& col, vector<bool>& diag1, vector<bool>& diag2) {

  if (row == N) {

    result.push_back(board);

    return;

  }


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

    if (col[i] || diag1[row - i + N - 1] || diag2[row + i]) continue;


    board[row][i] = 'Q';

    col[i] = diag1[row - i + N - 1] = diag2[row + i] = true;


    solveNQueensHelper(N, row + 1, board, result, col, diag1, diag2);


    board[row][i] = '.';

    col[i] = diag1[row - i + N - 1] = diag2[row + i] = false;

  }
```

```
}

vector<vector<string>> solveNQueens(int N) {
    vector<vector<string>> result;
    vector<string> board(N, string(N, '.'));
    vector<bool> col(N, false), diag1(2 * N - 1, false), diag2(2 * N - 1, false);

    solveNQueensHelper(N, 0, board, result, col, diag1, diag2);

    return result;
}

int main() {
    int N;
    cin >> N;

    vector<vector<string>> result = solveNQueens(N);

    for (auto& solution : result) {
        for (auto& row : solution) {
            cout << row << endl;
        }
        cout << endl;
    }

    return 0;
}
```

Question 39: Word Ladder

Problem Statement

Given two words beginWord and endWord, and a dictionary of words wordList, find the length of the shortest transformation sequence from beginWord to endWord, such that:

- Only one letter can be changed at a time.
- Each transformed word must exist in the word list.

Constraints:

- 1 ≤ beginWord.length ≤ 10.
- 1 ≤ endWord.length ≤ 10.
- 1 ≤ wordList.length ≤ 5000.

Input:
Strings beginWord, endWord, and a list of words wordList.

Output:
The length of the shortest transformation sequence, or 0 if no such sequence exists.

Example:
Input:

"hit" "cog" ["hot","dot","dog","lot","log","cog"]

Output:

5

Code:

```
#include <iostream>
#include <vector>
#include <unordered_set>
#include <queue>
using namespace std;


int wordLadderLength(string beginWord, string endWord, vector<string>& wordList) {
    unordered_set<string> wordSet(wordList.begin(), wordList.end());
    if (wordSet.find(endWord) == wordSet.end()) return 0;


    queue<string> q;
    q.push(beginWord);
    int length = 1;
```

```
    while (!q.empty()) {
        int n = q.size();
        for (int i = 0; i < n; i++) {
            string word = q.front();
            q.pop();

            if (word == endWord) return length;

            for (int j = 0; j < word.length(); j++) {
                char original = word[j];
                for (char c = 'a'; c <= 'z'; c++) {
                    word[j] = c;
                    if (wordSet.find(word) != wordSet.end()) {
                        q.push(word);
                        wordSet.erase(word);
                    }
                }
                word[j] = original;
            }
        }
        length++;
    }

    return 0;
}

int main() {
    string beginWord, endWord;
    int n;
    cin >> beginWord >> endWord >> n;
    vector<string> wordList(n);
```

```
    for (int i = 0; i < n; i++) {
        cin >> wordList[i];
    }

    cout << wordLadderLength(beginWord, endWord, wordList) << endl;

    return 0;
}
```

Question 40: Binary Search Tree Iterator

Problem Statement

Design an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST. Implement the BSTIterator class:

- BSTIterator(TreeNode root) Initializes an object of the BSTIterator class.
- boolean hasNext() Returns true if there is a next node in the in-order traversal.
- int next() Returns the next smallest number in the BST.

Constraints:

- The tree is a binary search tree (BST).
- The number of nodes in the tree is between 1 and 10^5.

Input:
A binary search tree represented as a root node root.

Output:
A sequence of function calls to hasNext() and next().

Example:
Input:

[7,3,15,null,null,9,20]

Output:

[3, 7, 9, 15, 20]

Code:

```
#include <iostream>

#include <stack>
```

```
using namespace std;

struct TreeNode {
   int val;
   TreeNode* left;

TreeNode* right; TreeNode(int x) : val(x), left(NULL), right(NULL) {} };
class BSTIterator { public: BSTIterator(TreeNode* root) { while (root) { s.push(root); root = root->left; }
}
bool hasNext() {
   return !s.empty();
}

int next() {
   TreeNode* node = s.top();
   s.pop();

   TreeNode* temp = node->right;
   while (temp) {
      s.push(temp);
      temp = temp->left;
   }

   return node->val;
}
private: stack<TreeNode*> s; };
int main() { TreeNode* root = new TreeNode(7); root->left = new TreeNode(3); root->right = new
TreeNode(15); root->right->left = new TreeNode(9); root->right->right = new TreeNode(20);
BSTIterator it(root);

while (it.hasNext()) {
   cout << it.next() << " ";
```

```
}


return 0;

}
```

Question 41: Combination Sum

Problem Statement

Given an array of distinct integers candidates and a target number target, find all unique combinations of numbers from candidates where the sum is equal to target. The same number may be chosen from candidates an unlimited number of times.

Constraints:

- 1 ≤ candidates.length ≤ 30.
- 2 ≤ target ≤ 500.
- 1 ≤ candidates[i] ≤ 50.

Input:
An array candidates[] and an integer target.

Output:
A list of all unique combinations that sum to target.

Example:
Input:

candidates = [2, 3, 6, 7], target = 7

Output:

[[2, 2, 3], [7]]

Code:

```
#include <iostream>

#include <vector>

using namespace std;


void combinationSumHelper(vector<int>& candidates, int target, int start, vector<int>& current, vector<vector<int>>& result) {

   if (target == 0) {

      result.push_back(current);

      return;
```

```
    }

    for (int i = start; i < candidates.size(); i++) {
        if (candidates[i] > target) continue;

        current.push_back(candidates[i]);
        combinationSumHelper(candidates, target - candidates[i], i, current, result);
        current.pop_back();
    }
}

vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
    vector<vector<int>> result;
    vector<int> current;
    combinationSumHelper(candidates, target, 0, current, result);
    return result;
}

int main() {
    int n, target;
    cin >> n >> target;
    vector<int> candidates(n);

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

    vector<vector<int>> result = combinationSum(candidates, target);

    for (auto& combination : result) {
        for (auto& num : combination) {
```

```
            cout << num << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Question 42: Climbing Stairs

Problem Statement

You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Constraints:

- $1 \leq n \leq 45$.

Input:
An integer n, representing the number of steps.

Output:
The number of distinct ways to reach the top.

Example:
Input:

n = 4

Output:

5

Code:

```
#include <iostream>
using namespace std;

int climbStairs(int n) {
    if (n == 1) return 1;
    if (n == 2) return 2;

    int a = 1, b = 2;
```

```
    for (int i = 3; i <= n; i++) {
        int temp = a + b;
        a = b;
        b = temp;
    }

    return b;
}

int main() {
    int n;
    cin >> n;

    cout << climbStairs(n) << endl;

    return 0;
}
```

Question 43: Serialize and Deserialize Binary Tree

Problem Statement

Design an algorithm to serialize and deserialize a binary tree. A binary tree is a tree where each node has at most two children. Your algorithm should support the following operations:

- serialize(root) which serializes the tree into a string.
- deserialize(data) which deserializes the string back into the tree.

Constraints:

- The tree is a binary tree (nodes may be null).

Input:
A binary tree with root node root.

Output:
A string representing the serialized tree and the reconstructed tree from the string.

Example:
Input:

[1, 2, 3, null, null, 4, 5]

Output:

"1,2,3,null,null,4,5"

Code:

```
#include <iostream>
#include <string>
#include <sstream>
#include <queue>
using namespace std;

struct TreeNode {
   int val;
   TreeNode* left;
   TreeNode* right;
   TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Codec {
public:
   string serialize(TreeNode* root) {
      if (!root) return "";

      stringstream ss;
      queue<TreeNode*> q;
      q.push(root);

      while (!q.empty()) {
         TreeNode* node = q.front();
         q.pop();

         if (node) {
```

```
            ss << node->val << ",";
            q.push(node->left);
            q.push(node->right);
        } else {
            ss << "null,";
        }
    }

    return ss.str();
}

TreeNode* deserialize(string data) {
    if (data.empty()) return NULL;

    stringstream ss(data);
    string str;
    getline(ss, str, ',');
    TreeNode* root = new TreeNode(stoi(str));
    queue<TreeNode*> q;
    q.push(root);

    while (getline(ss, str, ',')) {
        TreeNode* parent = q.front();
        q.pop();

        if (str != "null") {
            TreeNode* left = new TreeNode(stoi(str));
            parent->left = left;
            q.push(left);
        }
```

```
            if (getline(ss, str, ',')) {
                if (str != "null") {
                    TreeNode* right = new TreeNode(stoi(str));
                    parent->right = right;
                    q.push(right);
                }
            }
        }

        return root;
    }
};

int main() {
    Codec codec;
    string data;
    cin >> data;

    TreeNode* root = codec.deserialize(data);
    string serialized = codec.serialize(root);
    cout << serialized << endl;

    return 0;
}
```

Question 44: Longest Increasing Subsequence

Problem Statement

Given an integer array nums, return the length of the longest strictly increasing subsequence.

Constraints:

- 1 ≤ nums.length ≤ 2500.
- -10^4 ≤ nums[i] ≤ 10^4.

Input:
An array nums[] of integers.

Output:
The length of the longest increasing subsequence.

Example:
Input:

[10, 9, 2, 5, 3, 7, 101, 18]

Output:

4

Code:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;


int lengthOfLIS(vector<int>& nums) {

  if (nums.empty()) return 0;


  vector<int> dp(nums.size(), 1);


  for (int i = 1; i < nums.size(); i++) {

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

      if (nums[i] > nums[j]) {

        dp[i] = max(dp[i], dp[j] + 1);

      }

    }

  }


  return *max_element(dp.begin(), dp.end());

}


int main() {
```

```
    int n;
    cin >> n;
    vector<int> nums(n);

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

    cout << lengthOfLIS(nums) << endl;

    return 0;
}
```

Question 45: Kth Largest Element in an Array

Problem Statement

Given an integer array nums and an integer k, return the kth largest element in the array.

Constraints:

- 1 ≤ k ≤ nums.length ≤ 10^4.
- -10^4 ≤ nums[i] ≤ 10^4.

Input:
An array nums[] and an integer k.

Output:
The kth largest element in the array.

Example:
Input:

[3,2,1,5,6,4], k = 2

Output:

5

Code:

```
#include <iostream>
#include <vector>
#include <queue>
```

```
using namespace std;

int findKthLargest(vector<int>& nums, int k) {
    priority_queue<int> pq;

    for (int num : nums) {
        pq.push(num);
    }

    for (int i = 1; i < k; i++) {
        pq.pop();
    }

    return pq.top();
}

int main() {
    int n, k;
    cin >> n >> k;
    vector<int> nums(n);

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

    cout << findKthLargest(nums, k) << endl;

    return 0;
}
```

Question 46: Maximum Subarray Sum (Kadane's Algorithm)

Problem Statement

Given an integer array nums, find the contiguous subarray (containing at least one number) that has the largest sum and return its sum.

Constraints:

- 1 ≤ nums.length ≤ 10^5.
- -10^4 ≤ nums[i] ≤ 10^4.

Input:
An array nums[] of integers.

Output:
The maximum sum of the contiguous subarray.

Example:
Input:

nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

Output:

6

Code:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;


int maxSubArray(vector<int>& nums) {

   int maxSum = nums[0];

   int currentSum = nums[0];


   for (int i = 1; i < nums.size(); i++) {

      currentSum = max(nums[i], currentSum + nums[i]);

      maxSum = max(maxSum, currentSum);

   }


   return maxSum;
```

```
}

int main() {
   int n;
   cin >> n;
   vector<int> nums(n);

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

   cout << maxSubArray(nums) << endl;

   return 0;
}
```

Question 47: Merge Intervals

Problem Statement

Given a collection of intervals, merge all overlapping intervals.

Constraints:

- 1 ≤ intervals.length ≤ 10^4.
- -10^4 ≤ intervals[i][0] ≤ intervals[i][1] ≤ 10^4.

Input:
A 2D array intervals[][] where each interval is a pair [start, end].

Output:
A 2D array of merged intervals.

Example:
Input:

intervals = [[1,3],[2,6],[8,10],[15,18]]

Output:

[[1,6],[8,10],[15,18]]

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

vector<vector<int>> merge(vector<vector<int>>& intervals) {
    if (intervals.empty()) return {};

    sort(intervals.begin(), intervals.end());
    vector<vector<int>> result;
    result.push_back(intervals[0]);

    for (int i = 1; i < intervals.size(); i++) {
        if (result.back()[1] >= intervals[i][0]) {
            result.back()[1] = max(result.back()[1], intervals[i][1]);
        } else {
            result.push_back(intervals[i]);
        }
    }

    return result;
}

int main() {
    int n;
    cin >> n;
    vector<vector<int>> intervals(n, vector<int>(2));

    for (int i = 0; i < n; i++) {
        cin >> intervals[i][0] >> intervals[i][1];
    }
```

```
    vector<vector<int>> result = merge(intervals);

    for (auto& interval : result) {
        cout << "[" << interval[0] << "," << interval[1] << "] ";
    }
    cout << endl;

    return 0;
}
```

Question 48: Count Inversions in an Array

Problem Statement

Given an array nums, return the number of inversions in the array. An inversion is a pair of indices (i, j) such that i < j and nums[i] > nums[j].

Constraints:

- 1 ≤ nums.length ≤ 10^5.
- -10^6 ≤ nums[i] ≤ 10^6.

Input:
An array nums[] of integers.

Output:
The number of inversions in the array.

Example:
Input:

nums = [2, 4, 1, 3, 5]

Output:

3

Code:

```
#include <iostream>

#include <vector>

using namespace std;
```

```
int mergeAndCount(vector<int>& nums, vector<int>& temp, int left, int right) {
    int count = 0;
    if (left < right) {
        int mid = left + (right - left) / 2;

        count += mergeAndCount(nums, temp, left, mid);
        count += mergeAndCount(nums, temp, mid + 1, right);

        count += merge(nums, temp, left, mid, right);
    }

    return count;
}

int merge(vector<int>& nums, vector<int>& temp, int left, int mid, int right) {
    int i = left, j = mid + 1, k = left;
    int count = 0;

    while (i <= mid && j <= right) {
        if (nums[i] <= nums[j]) {
            temp[k++] = nums[i++];
        } else {
            temp[k++] = nums[j++];
            count += (mid - i + 1);
        }
    }

    while (i <= mid) {
        temp[k++] = nums[i++];
    }
```

```
    while (j <= right) {
        temp[k++] = nums[j++];
    }

    for (int i = left; i <= right; i++) {
        nums[i] = temp[i];
    }

    return count;
}

int countInversions(vector<int>& nums) {
    vector<int> temp(nums.size());
    return mergeAndCount(nums, temp, 0, nums.size() - 1);
}

int main() {
    int n;
    cin >> n;
    vector<int> nums(n);

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

    cout << countInversions(nums) << endl;

    return 0;
}
```

Question 49: Longest Palindromic Substring

Problem Statement

Given a string s, return the longest palindromic substring.

Constraints:

- 1 ≤ s.length ≤ 1000.
- s consists of only digits and English letters.

Input:
A string s of length n.

Output:
The longest palindromic substring in s.

Example:
Input:

s = "babad"

Output:

"bab"

Code:

```
#include <iostream>

#include <string>

using namespace std;


string expandAroundCenter(string& s, int left, int right) {

   while (left >= 0 && right < s.length() && s[left] == s[right]) {

      left--;

      right++;

   }

   return s.substr(left + 1, right - left - 1);

}


string longestPalindrome(string s) {

   if (s.empty()) return "";


   string longest = "";
```

```
    for (int i = 0; i < s.length(); i++) {
        string odd = expandAroundCenter(s, i, i);
        string even = expandAroundCenter(s, i, i + 1);

        if (odd.length() > longest.length()) {
            longest = odd;
        }
        if (even.length() > longest.length()) {
            longest = even;
        }
    }

    return longest;
}

int main() {
    string s;
    cin >> s;

    cout << longestPalindrome(s) << endl;

    return 0;
}
```

Question 50: Find the Missing Number

Problem Statement

Given an array containing n distinct numbers taken from the range 1 to n + 1, find the one number that is missing from the array.

Constraints:

- 2 ≤ nums.length ≤ 10^5.
- 1 ≤ nums[i] ≤ n.

Input:
An array nums[] of length n.

Output:
The missing number in the array.

Example:
Input:

nums = [3, 7, 1, 2, 8, 4, 5]

Output:

6

Code:

```
#include <iostream>

#include <vector>

using namespace std;


int findMissingNumber(vector<int>& nums) {

   int n = nums.size() + 1;

   int totalSum = (n * (n + 1)) / 2;

   int arraySum = 0;


   for (int num : nums) {

      arraySum += num;

   }


   return totalSum - arraySum;

}


int main() {

   int n;

   cin >> n;

   vector<int> nums(n);


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

```
        cin >> nums[i];
    }

    cout << findMissingNumber(nums) << endl;

    return 0;
}
```

Question 51: Maximum Product Subarray

Problem Statement

Given an integer array nums, find the contiguous subarray within the array that has the largest product.

Constraints:

- 1 ≤ nums.length ≤ 10^5.
- -10^4 ≤ nums[i] ≤ 10^4.

Input:
An array nums[] of integers.

Output:
The maximum product of any subarray.

Example:
Input:

nums = [2, 3, -2, 4]

Output:

6

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int maxProduct(vector<int>& nums) {
    int maxProd = nums[0], minProd = nums[0], result = nums[0];
```

```
    for (int i = 1; i < nums.size(); i++) {
        if (nums[i] < 0) {
            swap(maxProd, minProd);
        }
        maxProd = max(nums[i], maxProd * nums[i]);
        minProd = min(nums[i], minProd * nums[i]);

        result = max(result, maxProd);
    }

    return result;
}

int main() {
    int n;
    cin >> n;
    vector<int> nums(n);

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

    cout << maxProduct(nums) << endl;

    return 0;
}
```

Question 52: Valid Parentheses

Problem Statement

Given a string s containing just the characters '(', ')', '{', '}', '[', and ']', determine if the input string is valid. An input string is valid if:

- Open brackets must be closed by the corresponding closing brackets.
- Open brackets must be closed in the correct order.

Constraints:

- 1 ≤ s.length ≤ 10^4.
- s[i] is one of '(', ')', '{', '}', '[', ']'.

Input:
A string s.

Output:
Return true if the string is valid, otherwise return false.

Example:
Input:

s = "()[]{}"

Output:

true

Code:

```
#include <iostream>

#include <stack>

#include <unordered_map>

using namespace std;


bool isValid(string s) {

   stack<char> st;

   unordered_map<char, char> bracketPair = {{')', '('}, {']', '['}, {'}', '{'}};


   for (char c : s) {

      if (bracketPair.count(c)) {

         if (st.empty() || st.top() != bracketPair[c]) {

            return false;

         }

         st.pop();
```

```
        } else {
            st.push(c);
        }
    }

    return st.empty();
}

int main() {
    string s;
    cin >> s;

    cout << (isValid(s) ? "true" : "false") << endl;

    return 0;
}
```

Question 53: Search in Rotated Sorted Array

Problem Statement

Suppose an array of length n sorted in ascending order is rotated between 1 and n times. Given a target value, search in the rotated sorted array and return its index. If the target is not found, return -1.

Constraints:

- 1 ≤ nums.length ≤ 10^5.
- -10^4 ≤ nums[i] ≤ 10^4.
- All elements in the array are unique.

Input:
An array nums[] and a target integer target.

Output:
The index of the target in the rotated array. If not found, return -1.

Example:
Input:

nums = [4, 5, 6, 7, 0, 1, 2], target = 0

Output:

4

Code:

```
#include <iostream>
#include <vector>
using namespace std;

int search(vector<int>& nums, int target) {
    int left = 0, right = nums.size() - 1;

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

        if (nums[mid] == target) {
            return mid;
        }

        if (nums[left] <= nums[mid]) {
            if (nums[left] <= target && target < nums[mid]) {
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        } else {
            if (nums[mid] < target && target <= nums[right]) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
    }
```

```
    return -1;
}

int main() {
    int n, target;
    cin >> n >> target;
    vector<int> nums(n);

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

    cout << search(nums, target) << endl;

    return 0;
}
```

Question 54: Coin Change

Problem Statement

Given an integer array coins[] representing coins of different denominations and an integer amount, return the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

Constraints:

- 1 ≤ coins.length ≤ 12.
- 1 ≤ coins[i] ≤ 500.
- 0 ≤ amount ≤ 5000.

Input:
An array coins[] of coin denominations and an integer amount.

Output:
The fewest number of coins that you need to make up the amount. If no solution exists, return -1.

Example:
Input:

coins = [1, 2, 5], amount = 11

Output:

3

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int coinChange(vector<int>& coins, int amount) {
    vector<int> dp(amount + 1, amount + 1);
    dp[0] = 0;

    for (int i = 1; i <= amount; i++) {
        for (int coin : coins) {
            if (i - coin >= 0) {
                dp[i] = min(dp[i], dp[i - coin] + 1);
            }
        }
    }

    return dp[amount] == amount + 1 ? -1 : dp[amount];
}

int main() {
    int n, amount;
    cin >> n >> amount;
    vector<int> coins(n);

    for (int i = 0; i < n; i++) {
        cin >> coins[i];
```

```
    }

    cout << coinChange(coins, amount) << endl;

    return 0;
}
```

Question 55: Longest Common Prefix

Problem Statement

Write a function to find the longest common prefix string amongst an array of strings. If there is no common prefix, return an empty string "".

Constraints:

- 1 ≤ strs.length ≤ 200.
- 0 ≤ strs[i].length ≤ 200.

Input:
An array of strings strs[].

Output:
The longest common prefix string.

Example:
Input:

strs = ["flower", "flow", "flight"]

Output:

"fl"

Code:

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

string longestCommonPrefix(vector<string>& strs) {
    if (strs.empty()) return "";
```

```
    string prefix = strs[0];
    for (int i = 1; i < strs.size(); i++) {
        while (strs[i].find(prefix) != 0) {
            prefix = prefix.substr(0, prefix.length() - 1);
            if (prefix.empty()) return "";
        }
    }

    return prefix;
}

int main() {
    int n;
    cin >> n;
    vector<string> strs(n);

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

    cout << longestCommonPrefix(strs) << endl;

    return 0;
}
```

Question 56: Find the Duplicate Number

Problem Statement

Given an array of integers nums containing n + 1 integers where each integer is between 1 and n, inclusive, find the duplicate number. You must solve it without modifying the input array and using only constant extra space.

Constraints:

- 2 ≤ nums.length ≤ 10^5.

- 1 ≤ nums[i] ≤ n where n is the length of the array.

Input:
An array nums[] with integers between 1 and n.

Output:
The duplicate number in the array.

Example:
Input:

nums = [1, 3, 4, 2, 2]

Output:

2

Code:

```
#include <iostream>

#include <vector>

using namespace std;


int findDuplicate(vector<int>& nums) {

   int slow = nums[0], fast = nums[0];


   do {

      slow = nums[slow];

      fast = nums[nums[fast]];

   } while (slow != fast);


   slow = nums[0];

   while (slow != fast) {

      slow = nums[slow];

      fast = nums[fast];

   }


   return slow;

}
```

```
int main() {
    int n;
    cin >> n;
    vector<int> nums(n);

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

    cout << findDuplicate(nums) << endl;

    return 0;
}
```

Question 57: Longest Substring Without Repeating Characters

Problem Statement

Given a string s, find the length of the longest substring without repeating characters.

Constraints:

- 0 ≤ s.length ≤ 5 * 10^4.
- s consists of English letters, digits, symbols, and spaces.

Input:
A string s.

Output:
The length of the longest substring without repeating characters.

Example:
Input:

s = "abcabcbb"

Output:

3

Code:

```
#include <iostream>
#include <unordered_map>
```

```
using namespace std;

int lengthOfLongestSubstring(string s) {
    unordered_map<char, int> charIndex;
    int maxLength = 0, left = 0;

    for (int right = 0; right < s.length(); right++) {
        if (charIndex.count(s[right])) {
            left = max(left, charIndex[s[right]] + 1);
        }
        charIndex[s[right]] = right;
        maxLength = max(maxLength, right - left + 1);
    }

    return maxLength;
}

int main() {
    string s;
    cin >> s;

    cout << lengthOfLongestSubstring(s) << endl;

    return 0;
}
```

Question 58: Unique Paths

Problem Statement

A robot is located at the top-left corner of a m x n grid. The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner. How many possible unique paths are there?

Constraints:

- $1 \leq m, n \leq 100$.

Input:
Two integers m and n, representing the dimensions of the grid.

Output:
The number of unique paths to reach the bottom-right corner.

Example:
Input:

m = 3, n = 7

Output:

28

Code:

```
#include <iostream>

#include <vector>

using namespace std;


int uniquePaths(int m, int n) {

   vector<vector<int>> dp(m, vector<int>(n, 1));


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

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

         dp[i][j] = dp[i - 1][j] + dp[i][j - 1];

      }

   }


   return dp[m - 1][n - 1];

}


int main() {

   int m, n;

   cin >> m >> n;


   cout << uniquePaths(m, n) << endl;
```

```
    return 0;
}
```

Question 59: Move Zeroes

Problem Statement

Given an array nums, write a function that moves all 0's to the end of it while maintaining the relative order of the non-zero elements.

Constraints:

- 1 ≤ nums.length ≤ 10^4.
- -2^31 ≤ nums[i] ≤ 2^31 - 1.

Input:
An array nums[].

Output:
The array after moving all zeros to the end.

Example:
Input:

nums = [0, 1, 0, 3, 12]

Output:

[1, 3, 12, 0, 0]

Code:

```
#include <iostream>
#include <vector>
using namespace std;

void moveZeroes(vector<int>& nums) {
    int index = 0;

    for (int i = 0; i < nums.size(); i++) {
        if (nums[i] != 0) {
            nums[index++] = nums[i];
        }
```

```
    }

    for (int i = index; i < nums.size(); i++) {
        nums[i] = 0;
    }
}

int main() {
    int n;
    cin >> n;
    vector<int> nums(n);

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

    moveZeroes(nums);

    for (int num : nums) {
        cout << num << " ";
    }
    cout << endl;

    return 0;
}
```

Question 60: Subset Sum

Problem Statement

Given a set of positive numbers, find all subsets that sum to a given number.

Constraints:

- 1 ≤ n ≤ 30.

- The sum S is not greater than 10^6.

Input:
An array nums[] and a number S.

Output:
All subsets that sum up to S.

Example:
Input:

nums = [2, 3, 7], S = 7

Output:

[7]

[2, 3]

Code:

```
#include <iostream>

#include <vector>

using namespace std;


void findSubsets(vector<int>& nums, int sum, vector<int>& current, int index) {
   if (sum == 0) {
      for (int num : current) {
         cout << num << " ";
      }
      cout << endl;
      return;
   }

   for (int i = index; i < nums.size(); i++) {
      if (nums[i] <= sum) {
         current.push_back(nums[i]);
         findSubsets(nums, sum - nums[i], current, i + 1);
         current.pop_back();
      }
   }
```

```
}

int main() {
    int n, sum;
    cin >> n >> sum;
    vector<int> nums(n);

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

    vector<int> current;
    findSubsets(nums, sum, current, 0);

    return 0;
}
```

Question 61: Find the Peak Element

Problem Statement

An element is a peak if it is greater than or equal to its neighbors. Given an integer array nums, find a peak element, and return its index.
Your solution should not exceed O(log n) time complexity.

Constraints:

- 1 ≤ nums.length ≤ 1000.
- -10^5 ≤ nums[i] ≤ 10^5.

Input:
An array nums[].

Output:
The index of a peak element.

Example:
Input:

nums = [1, 2, 3, 1]

Output:

2

Code:

```
#include <iostream>
#include <vector>
using namespace std;

int findPeakElement(vector<int>& nums) {
    int left = 0, right = nums.size() - 1;

    while (left < right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] > nums[mid + 1]) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }

    return left;
}

int main() {
    int n;
    cin >> n;
    vector<int> nums(n);

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

    cout << findPeakElement(nums) << endl;
```

```
    return 0;
}
```

Question 62: Trapping Rain Water

Problem Statement

Given an array height[] representing the elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Constraints:

- n = height.length
- 2 ≤ n ≤ 10^5.
- 0 ≤ height[i] ≤ 10^5.

Input:
An array height[].

Output:
The total amount of trapped rainwater.

Example:
Input:

height = [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]

Output:

6

Code:

```
#include <iostream>
#include <vector>
using namespace std;

int trap(vector<int>& height) {
    int n = height.size();
    if (n == 0) return 0;

    vector<int> leftMax(n), rightMax(n);
    leftMax[0] = height[0];
```

```
    rightMax[n - 1] = height[n - 1];

    for (int i = 1; i < n; i++) {
        leftMax[i] = max(leftMax[i - 1], height[i]);
    }

    for (int i = n - 2; i >= 0; i--) {
        rightMax[i] = max(rightMax[i + 1], height[i]);
    }

    int water = 0;
    for (int i = 0; i < n; i++) {
        water += min(leftMax[i], rightMax[i]) - height[i];
    }

    return water;
}

int main() {
    int n;
    cin >> n;
    vector<int> height(n);

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

    cout << trap(height) << endl;

    return 0;
}
```

Question 63: Merge Intervals

Problem Statement

Given a collection of intervals, merge all overlapping intervals. For example, given intervals [(1, 3), (2, 4), (5, 7)], return [(1, 4), (5, 7)].

Constraints:

- 1 ≤ intervals.length ≤ 10^4.
- intervals[i].length == 2.
- 0 ≤ intervals[i][0] ≤ intervals[i][1] ≤ 10^4.

Input:
A list of intervals [[start, end], ...].

Output:
A list of merged intervals.

Example:
Input:

intervals = [[1, 3], [2, 4], [5, 7]]

Output:

[[1, 4], [5, 7]]

Code:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;


vector<vector<int>> mergeIntervals(vector<vector<int>>& intervals) {

  if (intervals.empty()) return {};


  sort(intervals.begin(), intervals.end());

  vector<vector<int>> merged;


  merged.push_back(intervals[0]);

  for (int i = 1; i < intervals.size(); i++) {
```

```
        if (merged.back()[1] >= intervals[i][0]) {
            merged.back()[1] = max(merged.back()[1], intervals[i][1]);
        } else {
            merged.push_back(intervals[i]);
        }
    }

    return merged;
}

int main() {
    int n;
    cin >> n;
    vector<vector<int>> intervals(n, vector<int>(2));

    for (int i = 0; i < n; i++) {
        cin >> intervals[i][0] >> intervals[i][1];
    }

    vector<vector<int>> result = mergeIntervals(intervals);

    for (auto& interval : result) {
        cout << "[" << interval[0] << ", " << interval[1] << "] ";
    }
    cout << endl;

    return 0;
}
```

Question 64: Longest Palindromic Substring

Problem Statement

Given a string s, find the longest palindromic substring.

Constraints:

- 1 ≤ s.length ≤ 1000.
- s consists of only uppercase and lowercase English letters.

Input:
A string s.

Output:
The longest palindromic substring.

Example:
Input:

s = "babad"

Output:

"bab" (or "aba")

Code:

```
#include <iostream>

#include <string>

using namespace std;


string expandAroundCenter(string& s, int left, int right) {

   while (left >= 0 && right < s.length() && s[left] == s[right]) {

      left--;

      right++;

   }

   return s.substr(left + 1, right - left - 1);

}


string longestPalindrome(string s) {

   string result;

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

      string odd = expandAroundCenter(s, i, i);

      string even = expandAroundCenter(s, i, i + 1);
```

```
        if (odd.length() > result.length()) result = odd;
        if (even.length() > result.length()) result = even;
    }
    return result;
}

int main() {
    string s;
    cin >> s;

    cout << longestPalindrome(s) << endl;

    return 0;
}
```

Question 65: Rotate Image

Problem Statement

You are given an n x n 2D matrix representing an image. Rotate the image by 90 degrees (clockwise).

Constraints:

- 1 ≤ n ≤ 100.
- The matrix elements are between -1000 and 1000.

Input:
An n x n matrix matrix[][].

Output:
The matrix rotated by 90 degrees clockwise.

Example:
Input:

```
matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
]
```

Output:

[

[7, 4, 1],

[8, 5, 2],

[9, 6, 3]

]

Code:

```
#include <iostream>
#include <vector>
using namespace std;

void rotate(vector<vector<int>>& matrix) {
    int n = matrix.size();

    // Transpose the matrix
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            swap(matrix[i][j], matrix[j][i]);
        }
    }

    // Reverse each row
    for (int i = 0; i < n; i++) {
        reverse(matrix[i].begin(), matrix[i].end());
    }
}

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

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

    rotate(matrix);

    for (const auto& row : matrix) {
        for (int num : row) {
            cout << num << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Question 66: Subarray Sum Equals K

Problem Statement

Given an array of integers and an integer k, return the total number of continuous subarrays whose sum equals k.

Constraints:

- 1 ≤ nums.length ≤ 2 * 10^4.
- -1000 ≤ nums[i] ≤ 1000.
- -10^7 ≤ k ≤ 10^7.

Input:
An integer array nums[] and an integer k.

Output:
The total number of subarrays whose sum equals k.

Example:

Input:

nums = [1, 1, 1], k = 2

Output:

2

Code:

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;

int subarraySum(vector<int>& nums, int k) {
    unordered_map<int, int> sumCount;
    sumCount[0] = 1;
    int sum = 0, count = 0;

    for (int num : nums) {
        sum += num;
        if (sumCount.find(sum - k) != sumCount.end()) {
            count += sumCount[sum - k];
        }
        sumCount[sum]++;
    }

    return count;
}

int main() {
    int n, k;
    cin >> n >> k;
    vector<int> nums(n);
```

```
    for (int i = 0; i < n; i++) {
        cin >> nums[i];
    }

    cout << subarraySum(nums, k) << endl;

    return 0;
}
```

Question 67: Longest Common Prefix

Problem Statement

Write a function to find the longest common prefix string amongst an array of strings.

Constraints:

- 1 ≤ strs.length ≤ 200.
- 0 ≤ strs[i].length ≤ 200.
- strs[i] consists of only lowercase English letters.

Input:
A list of strings strs[].

Output:
The longest common prefix string.

Example:
Input:

strs = ["flower", "flow", "flight"]

Output:

"fl"

Code:

```
#include <iostream>

#include <vector>

using namespace std;


string longestCommonPrefix(vector<string>& strs) {
```

```
    if (strs.empty()) return "";

    string prefix = strs[0];
    for (int i = 1; i < strs.size(); i++) {
        int j = 0;
        while (j < prefix.size() && j < strs[i].size() && prefix[j] == strs[i][j]) {
            j++;
        }
        prefix = prefix.substr(0, j);
        if (prefix == "") return "";
    }

    return prefix;
}

int main() {
    int n;
    cin >> n;
    vector<string> strs(n);

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

    cout << longestCommonPrefix(strs) << endl;

    return 0;
}
```

Question 68: Search in Rotated Sorted Array

Problem Statement

You are given an integer array nums sorted in ascending order, then rotated at some pivot index. Write a function to search for a target value in the array. If the target exists, return its index, otherwise return -1.

Constraints:

- 1 ≤ nums.length ≤ 5000.
- -10^4 ≤ nums[i] ≤ 10^4.
- The array may contain duplicates.

Input:
An array nums[] and an integer target.

Output:
The index of the target if it exists, otherwise -1.

Example:
Input:

nums = [4, 5, 6, 7, 0, 1, 2], target = 0

Output:

4

Code:

```
#include <iostream>

#include <vector>

using namespace std;


int search(vector<int>& nums, int target) {

  int left = 0, right = nums.size() - 1;


  while (left <= right) {

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


    if (nums[mid] == target) return mid;


    if (nums[left] <= nums[mid]) {

      if (nums[left] <= target && target < nums[mid]) {

        right = mid - 1;

      } else {
```

```
            left = mid + 1;
        }
    } else {
        if (nums[mid] < target && target <= nums[right]) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
}

return -1;
}

int main() {
int n, target;
cin >> n >> target;
vector<int> nums(n);

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

cout << search(nums, target) << endl;

return 0;
}
```

Question 69: Maximum Subarray

Problem Statement

Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

Constraints:

- 1 ≤ nums.length ≤ 10^5.
- -10^4 ≤ nums[i] ≤ 10^4.

Input:
An integer array nums[].

Output:
The largest sum of a contiguous subarray.

Example:
Input:

nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

Output:

6

Code:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;


int maxSubArray(vector<int>& nums) {

   int maxSum = nums[0], currentSum = nums[0];


   for (int i = 1; i < nums.size(); i++) {

      currentSum = max(nums[i], currentSum + nums[i]);

      maxSum = max(maxSum, currentSum);

   }


   return maxSum;

}


int main() {
```

```
    int n;
    cin >> n;
    vector<int> nums(n);

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

    cout << maxSubArray(nums) << endl;

    return 0;
}
```

Question 70: Kth Largest Element in an Array

Problem Statement

Find the kth largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

Constraints:

- 1 ≤ nums.length ≤ 10^5.
- -10^4 ≤ nums[i] ≤ 10^4.
- 1 ≤ k ≤ nums.length.

Input:
An integer array nums[] and an integer k.

Output:
The kth largest element.

Example:
Input:

nums = [3, 2, 1, 5, 6, 4], k = 2

Output:

5

Code:

```
#include <iostream>
```

```
#include <vector>
#include <queue>
using namespace std;

int findKthLargest(vector<int>& nums, int k) {
    priority_queue<int> maxHeap;

    for (int num : nums) {
        maxHeap.push(num);
    }

    for (int i = 1; i < k; i++) {
        maxHeap.pop();
    }

    return maxHeap.top();
}

int main() {
    int n, k;
    cin >> n >> k;
    vector<int> nums(n);

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

    cout << findKthLargest(nums, k) << endl;

    return 0;
}
```

Question 72: Merge Intervals

Problem Statement

Given a collection of intervals, merge all overlapping intervals.

Constraints:

- 1 ≤ intervals.length ≤ 10^4.
- intervals[i].length == 2.
- -10^4 ≤ intervals[i][0] ≤ intervals[i][1] ≤ 10^4.

Input:
A list of intervals, where each interval is represented as a pair of integers.

Output:
A list of merged intervals.

Example:
Input:

intervals = [[1, 3], [2, 6], [8, 10], [15, 18]]

Output:

[[1, 6], [8, 10], [15, 18]]

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;


vector<vector<int>> merge(vector<vector<int>>& intervals) {
    if (intervals.empty()) return {};

    sort(intervals.begin(), intervals.end());
    vector<vector<int>> merged;

    for (auto& interval : intervals) {
        if (merged.empty() || merged.back()[1] < interval[0]) {
            merged.push_back(interval);
```

```
        } else {
            merged.back()[1] = max(merged.back()[1], interval[1]);
        }
    }

    return merged;
}

int main() {
    int n;
    cin >> n;
    vector<vector<int>> intervals(n, vector<int>(2));

    for (int i = 0; i < n; i++) {
        cin >> intervals[i][0] >> intervals[i][1];
    }

    vector<vector<int>> result = merge(intervals);

    for (auto& interval : result) {
        cout << "[" << interval[0] << ", " << interval[1] << "] ";
    }
    cout << endl;

    return 0;
}
```

Question 73: Reverse Words in a String

Problem Statement

Given a string s, reverse the order of the words in the string. A word is defined as a sequence of characters delimited by spaces.

Constraints:

- 1 ≤ s.length ≤ 10^4.
- The input string may contain leading/trailing spaces, but you should ignore them.

Input:
A string s.

Output:
A string where the words are reversed in order.

Example:
Input:

s = "the sky is blue"

Output:

"blue is sky the"

Code:

```
#include <iostream>

#include <string>

#include <sstream>

#include <vector>

using namespace std;


string reverseWords(string s) {
   stringstream ss(s);
   string word;
   vector<string> words;

   while (ss >> word) {
      words.push_back(word);
   }

   string result = "";
   for (int i = words.size() - 1; i >= 0; i--) {
      result += words[i];
      if (i != 0) result += " ";
```

```
    }

    return result;
}

int main() {
    string s;
    getline(cin, s);

    cout << reverseWords(s) << endl;

    return 0;
}
```

Question 74: Find Peak Element

Problem Statement

A peak element in an array is an element that is greater than its neighbors. Given an integer array nums, find a peak element and return its index.

Constraints:

- 1 ≤ nums.length ≤ 5000.
- -2^31 ≤ nums[i] ≤ 2^31 - 1.

Input:
An array nums[].

Output:
The index of a peak element.

Example:
Input:

nums = [1, 2, 3, 1]

Output:

2

Code:

```
#include <iostream>
```

```
#include <vector>
using namespace std;

int findPeakElement(vector<int>& nums) {
    int left = 0, right = nums.size() - 1;

    while (left < right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] > nums[mid + 1]) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }

    return left;
}

int main() {
    int n;
    cin >> n;
    vector<int> nums(n);

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

    cout << findPeakElement(nums) << endl;

    return 0;
}
```

Question 75: Maximum Product Subarray

Problem Statement

Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest product and return the product.

Constraints:

- 1 ≤ nums.length ≤ 10^4.
- -10^4 ≤ nums[i] ≤ 10^4.

Input:
An integer array nums[].

Output:
The largest product of a contiguous subarray.

Example:
Input:

nums = [2, 3, -2, 4]

Output:

6

Code:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;


int maxProduct(vector<int>& nums) {

   int maxProduct = nums[0], minProduct = nums[0], result = nums[0];


   for (int i = 1; i < nums.size(); i++) {

      if (nums[i] < 0) swap(maxProduct, minProduct);


      maxProduct = max(nums[i], maxProduct * nums[i]);

      minProduct = min(nums[i], minProduct * nums[i]);
```

```
        result = max(result, maxProduct);
    }

    return result;
}

int main() {
    int n;
    cin >> n;
    vector<int> nums(n);

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

    cout << maxProduct(nums) << endl;

    return 0;
}
```

Question 76: Word Break

Problem Statement

Given a string s and a dictionary of strings wordDict, return true if s can be segmented into a space-separated sequence of one or more dictionary words.

Constraints:

- 1 ≤ s.length ≤ 300.
- 1 ≤ wordDict.length ≤ 1000.
- 1 ≤ wordDict[i].length ≤ 20.

Input:
A string s and a list of strings wordDict[].

Output:
True if the string can be segmented using the dictionary, otherwise false.

Example:

Input:

s = "leetcode", wordDict = ["leet", "code"]

Output:

true

Code:

```
#include <iostream>
#include <vector>
#include <unordered_set>
using namespace std;

bool wordBreak(string s, vector<string>& wordDict) {
    unordered_set<string> dict(wordDict.begin(), wordDict.end());
    vector<bool> dp(s.length() + 1, false);
    dp[0] = true;

    for (int i = 1; i <= s.length(); i++) {
        for (int j = 0; j < i; j++) {
            if (dp[j] && dict.find(s.substr(j, i - j)) != dict.end()) {
                dp[i] = true;
                break;
            }
        }
    }

    return dp[s.length()];
}

int main() {
    string s;
    int n;
```

```
    cin >> s >> n;
    vector<string> wordDict(n);

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

    cout << (wordBreak(s, wordDict) ? "true" : "false") << endl;

    return 0;
}
```

Question 77: Top K Frequent Elements

Problem Statement

Given an integer array nums and an integer k, return the k most frequent elements.

Constraints:

- 1 ≤ nums.length ≤ 10^5.
- 1 ≤ nums[i] ≤ 10^4.

Input:
An array nums[] and an integer k.

Output:
A list of the k most frequent elements.

Example:
Input:

nums = [1, 1, 1, 2, 2, 3], k = 2

Output:

[1, 2]

Code:

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <queue>
```

```
using namespace std;

vector<int> topKFrequent(vector<int>& nums, int k) {
    unordered_map<int, int> freqMap;
    for (int num : nums) {
        freqMap[num]++;
    }

    priority_queue<pair<int, int>> maxHeap;
    for (auto& pair : freqMap) {
        maxHeap.push({pair.second, pair.first});
    }

    vector<int> result;
    while (k-- > 0) {
        result.push_back(max
Heap.top().second); maxHeap.pop(); }
return result;
}
int main() { int n, k; cin >> n >> k; vector nums(n);
for (int i = 0; i < n; i++) {
    cin >> nums[i];
}

vector<int> result = topKFrequent(nums, k);

for (int num : result) {
    cout << num << " ";
}
cout << endl;
```

return 0;

}

Question 78: Longest Palindromic Substring

Problem Statement

Given a string s, find the longest palindromic substring in s.

Constraints:

- 1 ≤ s.length ≤ 1000.

Input:
A string s.

Output:
The longest palindromic substring.

Example:
Input:

s = "babad"

Output:

"bab" or "aba"

Code:

```
#include <iostream>

#include <string>

using namespace std;


string longestPalindrome(string s) {

   int start = 0, maxLength = 1;


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

      for (int j = i; j < s.size(); j++) {

         bool isPalindrome = true;

         for (int k = 0; k < (j - i + 1) / 2; k++) {

            if (s[i + k] != s[j - k]) {

               isPalindrome = false;

               break;
```

```
            }
        }
        if (isPalindrome && j - i + 1 > maxLength) {
            start = i;
            maxLength = j - i + 1;
        }
    }
  }

  return s.substr(start, maxLength);
}

int main() {
  string s;
  cin >> s;

  cout << longestPalindrome(s) << endl;

  return 0;
}
```

Question 79: Regular Expression Matching

Problem Statement

Given two strings s and p, implement regular expression matching with support for . and * where:

- . matches any single character.
- * matches zero or more of the preceding element.

Constraints:

- 1 ≤ s.length, p.length ≤ 20.

Input:
Two strings s and p.

Output:
True if p matches s, otherwise false.

Example:

Input:

s = "aab", p = "c*a*b"

Output:

true

Code:

```
#include <iostream>
#include <string>
using namespace std;

bool isMatch(string s, string p) {
   int m = s.size(), n = p.size();
   vector<vector<bool>> dp(m + 1, vector<bool>(n + 1, false));
   dp[0][0] = true;

   for (int j = 1; j <= n; j++) {
      if (p[j - 1] == '*') dp[0][j] = dp[0][j - 2];
   }

   for (int i = 1; i <= m; i++) {
      for (int j = 1; j <= n; j++) {
         if (p[j - 1] == '.' || s[i - 1] == p[j - 1]) {
            dp[i][j] = dp[i - 1][j - 1];
         } else if (p[j - 1] == '*') {
            dp[i][j] = dp[i][j - 2] || (dp[i - 1][j] && (p[j - 2] == '.' || s[i - 1] == p[j - 2]));
         }
      }
   }

   return dp[m][n];
}
```

```
int main() {
    string s, p;
    cin >> s >> p;

    cout << (isMatch(s, p) ? "true" : "false") << endl;

    return 0;
}
```

Question 80: Subarray Sum Equals K

Problem Statement

Given an integer array nums and an integer k, find the total number of continuous subarrays whose sum equals k.

Constraints:

- 1 ≤ nums.length ≤ 10^4.
- -10^4 ≤ nums[i] ≤ 10^4.
- -10^4 ≤ k ≤ 10^4.

Input:
An array nums[] and an integer k.

Output:
The number of continuous subarrays that sum to k.

Example:
Input:

nums = [1, 1, 1], k = 2

Output:

2

Code:

```
#include <iostream>

#include <vector>

#include <unordered_map>

using namespace std;
```

```
int subarraySum(vector<int>& nums, int k) {
    unordered_map<int, int> prefixSum;
    prefixSum[0] = 1;
    int sum = 0, count = 0;

    for (int num : nums) {
        sum += num;
        if (prefixSum.find(sum - k) != prefixSum.end()) {
            count += prefixSum[sum - k];
        }
        prefixSum[sum]++;
    }

    return count;
}

int main() {
    int n, k;
    cin >> n >> k;
    vector<int> nums(n);

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

    cout << subarraySum(nums, k) << endl;

    return 0;
}
```

Question 81: Binary Tree Inorder Traversal

Problem Statement

Given the root of a binary tree, return the inorder traversal of its nodes' values.

Constraints:

- The number of nodes in the tree is 1 ≤ nodes ≤ 10^4.
- -1000 ≤ Node.val ≤ 1000.

Input:
The root of the binary tree.

Output:
An array of integers representing the inorder traversal.

Example:
Input:

```
  1
 / \
2   3
/ \
4   5
```

Output:

[4, 2, 5, 1, 3]

Code:

```
#include <iostream>
#include <vector>
using namespace std;

struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

void inorderTraversal(TreeNode* root, vector<int>& result) {
```

```
    if (!root) return;
    inorderTraversal(root->left, result);
    result.push_back(root->val);
    inorderTraversal(root->right, result);
}

int main() {
    int n;
    cin >> n;
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);

    vector<int> result;
    inorderTraversal(root, result);

    for (int val : result) {
        cout << val << " ";
    }
    cout << endl;

    return 0;
}
```

Question 82: Group Anagrams

Problem Statement

Given an array of strings strs, group the anagrams together. You can return the answer in any order.

Constraints:

- 1 ≤ strs.length ≤ 10^4.

- 1 ≤ strs[i].length ≤ 100.

Input:
An array of strings strs[].

Output:
A list of groups of anagrams.

Example:
Input:

strs = ["eat", "tea", "tan", "ate", "nat", "bat"]

Output:

[["bat"], ["nat", "tan"], ["ate", "eat", "tea"]]

Code:

```
#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>
using namespace std;

vector<vector<string>> groupAnagrams(vector<string>& strs) {
    unordered_map<string, vector<string>> anagramMap;

    for (string& str : strs) {
        string sortedStr = str;
        sort(sortedStr.begin(), sortedStr.end());
        anagramMap[sortedStr].push_back(str);
    }

    vector<vector<string>> result;
    for (auto& pair : anagramMap) {
        result.push_back(pair.second);
    }

    return result;
```

```
}

int main() {
    int n;
    cin >> n;
    vector<string> strs(n);

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

    vector<vector<string>> result = groupAnagrams(strs);

    for (const auto& group : result) {
        for (const auto& word : group) {
            cout << word << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Question 83: Best Time to Buy and Sell Stock

Problem Statement

You are given an array prices where prices[i] is the price of a given stock on the i-th day. You want to maximize your profit by choosing a single day to buy one stock and choosing a different day to sell that stock.

Constraints:

- 1 ≤ prices.length ≤ 10^5.
- 0 ≤ prices[i] ≤ 10^4.

Input:
An array prices[].

Output:
The maximum profit you can achieve.

Example:
Input:

prices = [

7, 1, 5, 3, 6, 4]

Output:

5

Code:

```cpp
#include <iostream>
#include <vector>
using namespace std;

int maxProfit(vector<int>& prices) {
    int minPrice = INT_MAX, maxProfit = 0;

    for (int price : prices) {
        if (price < minPrice) {
            minPrice = price;
        } else {
            maxProfit = max(maxProfit, price - minPrice);
        }
    }

    return maxProfit;
}

int main() {
```

```
    int n;
    cin >> n;
    vector<int> prices(n);

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

    cout << maxProfit(prices) << endl;

    return 0;
}
```
