**Question 1:**

Imagine you are managing a vending machine that accepts various coin denominations. Your vending machine dispenses snacks and beverages, and customers can pay using coins. You want to optimize the change-giving process to minimize the number of coins given as change. Given a set of coin denominations (e.g., 1 cent, 5 cents, 10 cents, and 25 cents) and a total amount a customer needs as change, you design an algorithm to find the minimum number of coins required to make the change efficiently.

**Input Format:**

- The first line of input contains an integer n, representing the number of coins in the set.
- The second line contains n space-separated integers, representing the denominations of the coins in the set.
- The third line contains an integer total, representing the total amount for which the minimum number of coins is to be calculated.

**Output Format:**

- If it's possible to make the change for the given amount, the output will be a single line containing the minimum number of coins required in the format:
- Minimum number of coins required: <result>
- If it's not possible to make the change for the given amount, the output will be:
- It's not possible to make the change for the given amount.

**Title for Question 1:** Minimum Number of Coins

**Solution:**

```
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

int minCoins(vector<int>& coins, int total) {
    int n = coins.size();

    vector<int> dp(total + 1, INT_MAX);
    dp[0] = 0;

    for (int i = 1; i <= total; i++) {
        for (int j = 0; j < n; j++) {
            if (coins[j] <= i && dp[i - coins[j]] != INT_MAX) {
                dp[i] = min(dp[i], 1 + dp[i - coins[j]]);
            }
        }
    }
```

```
    return (dp[total] == INT_MAX) ? -1 : dp[total];
}

int main() {
    int n, total;
    cin >> n;
    vector<int> coins(n);
    for (int i = 0; i < n; i++) {
        cin >> coins[i];
    }
    cin >> total;
    int result = minCoins(coins, total);
    if (result == -1) {
        cout << "It's not possible to make the change for the given amoun
    } else {
        cout << "Minimum number of coins required: " << result << endl;
    }
    return 0;
}
```

**TestCases:**

| S.No | Inputs | Outputs |
|---|---|---|
| 1 | 3 1 2 5 11 | Minimum number of coins required: 3 |
| 2 | 2 3 7 8 | It's not possible to make the change for the given amount. |
| 3 | 3 2 5 7 15 | Minimum number of coins required: 3 |
| 4 | 4 2 5 10 20 37 | Minimum number of coins required: 4 |
| 5 | 4 1 2 4 8 11 | Minimum number of coins required: 3 |
| 6 | 2 4 6 5 | It's not possible to make the change for the given amount. |

**White List:**

**Black List:**

---

**Question 2:**

You are developing a cashier system for a futuristic arcade that uses a unique currency system with three types of coins: "Token," "Credit," and "Bit." The values of these coins are as follows: Token (T) = 7 units, Credit (C) = 5 units, and Bit (B) = 3 units. Your task is to implement a function that prints all possible combinations of these coins that make up a given amount of units. For example, if a customer wants change for 14 units, the function should output all the different combinations of tokens, credits, and bits that add up to 14 units. You would design and implement this function.

**Input Format:**

- The program expects the user to enter two integers on the first line separated by space:
- The first integer (n) represents the target amount for which combinations are to be found.
- The second integer (numCoins) represents the number of coin denominations.

- On the second line, the user should provide numCoins integers separated by space, representing the coin denominations.

**Output Format:**

- For each valid combination that makes up the given amount, the program prints a line starting with "Combination: " followed by the coins used, separated by commas.
- The coin combinations are printed in lexicographical order.

**Title for Question 2:** Print All Possible Combinations of Coin Change

**Solution:**

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

void printCombinations(std::vector<int>& coins, int amount, std::vector<i
    if (amount == 0) {
        // Print the combination
        std::cout << "Combination: ";
        for (int i = 0; i < combination.size(); ++i) {
            std::cout << combination[i];
            if (i < combination.size() - 1)
                std::cout << ", ";
        }
        std::cout << std::endl;
        return;
    }

    for (int i = index; i < coins.size(); ++i) {
        if (coins[i] <= amount) {
            combination.push_back(coins[i]);
            printCombinations(coins, amount - coins[i], combination, i);
            combination.pop_back(); // Backtrack
        }
    }
}

int main() {
    int n, numCoins, coin;
    std::cin >> n;
    std::cin >> numCoins;
    std::vector<int> coins;
    for (int i = 0; i < numCoins; ++i) {
        std::cin >> coin;
        coins.push_back(coin);
    }
    std::vector<int> combination;
    printCombinations(coins, n, combination, 0);
    return 0;
}
```

**TestCases:**

| S.No | Inputs | Outputs |
|---|---|---|
| 1 | 11 3 1 2 5 | Combination: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 Combination: 1, 1, 1, 1, 1, 1, 1, 1, 1, 2 Combination: 1, 1, 1, 1, 1, 1, 1, 2, 2 Combination: 1, 1, 1, 1, 1, 1, 5 Combination: 1, 1, 1, 1, 1, 2, 2, 2 Combination: 1, 1, 1, 1, 2, 5 Combination: 1, 1, 1, 2, 2, 2, 2 Combination: 1, 1, 2, 2, 5 Combination: 1, 2, 2, 2, 2, 2 Combination: 1, 5, 5 Combination: 2, 2, 2, 5 |
| 2 | 8 2 2 3 | Combination: 2, 2, 2, 2 Combination: 2, 3, 3 |
| 3 | 5 3 1 2 3 | Combination: 1, 1, 1, 1, 1 Combination: 1, 1, 1, 2 Combination: 1, 1, 3 Combination: 1, 2, 2 Combination: 2, 3 |
| 4 | 7 3 2 3 5 | Combination: 2, 2, 3 Combination: 2, 5 |
| 5 | 15 2 1 7 | Combination: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 Combination: 1, 1, 1, 1, 1, 1, 1, 1, 7 Combination: 1, 7, 7 |
| 6 | 9 2 3 4 | Combination: 3, 3, 3 |

**White List:**

**Black List:**

---

**Question 3:**

Imagine you are managing a vending machine that dispenses snacks and beverages. The machine accepts coins of various denominations. You have two arrays representing the available coins in the machine: one array for regular coins and another for special edition coins. Additionally, there are two amounts representing the cost of a snack and a beverage. Your task is to determine the maximum sum of coins that can be obtained from both regular and special edition coins to make the exact change for both the snack and the beverage. Consider that you can use any combination of coins from both arrays to achieve the maximum sum.

**Input Format:**

- The first line contains an integer n1 representing the size of the first array of coins.
- The second line contains n1 space-separated integers representing the elements of the first array of coins.
- The third line contains an integer n2 representing the size of the second array of coins.
- The fourth line contains n2 space-separated integers representing the elements of the second array of coins.
- The fifth line contains an integer amount1 representing the first amount.
- The sixth line contains an integer amount2 representing the second amount.

**Output Format:**

- The program outputs a single line containing the maximum sum of coins that can be obtained to make the change.

**Title for Question 3:** Maximum Sum Combination

**Solution:**

```
#include <iostream>
#include <vector>
#include <algorithm>

int maxCoinsSum(const std::vector<int>& coins1, const std::vector<int>& c
    int n1 = coins1.size();
    int n2 = coins2.size();

    // Create a 2D vector to store the maximum sum at each subproblem
    std::vector<std::vector<int>> dp(n1 + 1, std::vector<int>(n2 + 1, 0))

    // Fill the dp array using bottom-up dynamic programming
    for (int i = 1; i <= n1; ++i) {
        for (int j = 1; j <= n2; ++j) {
            // Calculate the maximum sum considering the current coin fro
            dp[i][j] = std::max(dp[i - 1][j], dp[i][j - 1]);

            // Update the maximum sum if the current coin can be used for
            if (amount1 >= coins1[i - 1] && amount2 >= coins2[j - 1]) {
                dp[i][j] = std::max(dp[i][j], dp[i - 1][j - 1] + coins1[i
            }
        }
    }

    // Return the maximum sum of coins
    return dp[n1][n2];
}

int main() {
    int n1, n2;
    std::cin >> n1;
    std::vector<int> coins1(n1);
    for (int i = 0; i < n1; ++i) {
        std::cin >> coins1[i];
    }
    std::cin >> n2;
    std::vector<int> coins2(n2);
    for (int i = 0; i < n2; ++i) {
        std::cin >> coins2[i];
    }
    int amount1, amount2;
    std::cin >> amount1;
    std::cin >> amount2;
    int result = maxCoinsSum(coins1, coins2, amount1, amount2);
    std::cout << "Maximum sum of coins that can be obtained: " << result
    return 0;
}
```

**TestCases:**

| S.No | Inputs | Outputs |
|---|---|---|
| 1 | 3 2 4 1 2 1 3 5 4 | Maximum sum of coins that can be obtained: 10 |
| 2 | 3 1 2 3 3 2 3 4 6 8 | Maximum sum of coins that can be obtained: 15 |

| S.No | Inputs | Outputs |
|---|---|---|
| 3 | 4 3 5 7 9 3 1 2 3 10 5 | Maximum sum of coins that can be obtained: 27 |
| 4 | 4 1 2 3 4 2 2 4 6 8 | Maximum sum of coins that can be obtained: 13 |
| 5 | 3 1 1 1 3 1 1 1 2 2 | Maximum sum of coins that can be obtained: 6 |
| 6 | 2 2 2 2 1 1 2 2 | Maximum sum of coins that can be obtained: 6 |

**White List:**

**Black List:**

---

**Question 4:**

Imagine you are working on an e-commerce platform where each product is assigned a unique integer identifier. The product IDs are stored in an array, and you want to optimize the user experience by identifying the length of the longest increasing subsequence of product IDs. This subsequence would represent a sequence of products that are arranged in ascending order of their IDs. Your task is to write a function or algorithm to find the length of the longest increasing subsequence of product IDs.

**Input Format:**

- The first line contains an integer n, representing the size of the array.
- The second line contains n space-separated integers, representing the elements of the array.

**Output Format:**

- A single line containing an integer, representing the length of the longest increasing subsequence in the given array.

**Title for Question 4:** Longest Increasing Subsequence

**Solution:**

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

int longestIncreasingSubsequence(const std::vector<int>& nums) {
    int n = nums.size();
    if (n == 0) {
        return 0;
    }

    std::vector<int> dp(n, 1);

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

```
            if (nums[i] > nums[j] && dp[i] < dp[j] + 1) {
                dp[i] = dp[j] + 1;
            }
        }
    }

    int maxLength = 1;
    for (int i = 0; i < n; ++i) {
        if (dp[i] > maxLength) {
            maxLength = dp[i];
        }
    }

    return maxLength;
}

int main() {
    int n;
    std::cin >> n;
    std::vector<int> nums(n);
    for (int i = 0; i < n; ++i) {
        std::cin >> nums[i];
    }
    int result = longestIncreasingSubsequence(nums);
    std::cout << "Length of the longest increasing subsequence: " << resu
    return 0;
}
```

**TestCases:**

| S.No | Inputs | Outputs |
|---|---|---|
| 1 | 7 1 2 3 4 5 6 7 | Length of the longest increasing subsequence: 7 |
| 2 | 6 3 10 2 1 20 30 | Length of the longest increasing subsequence: 4 |
| 3 | 5 10 9 2 5 3 | Length of the longest increasing subsequence: 2 |
| 4 | 8 5 6 7 8 9 1 2 3 | Length of the longest increasing subsequence: 5 |
| 5 | 9 10 22 9 33 21 50 41 60 80 | Length of the longest increasing subsequence: 6 |
| 6 | 10 3 5 10 8 12 15 7 8 11 13 | Length of the longest increasing subsequence: 6 |

**White List:**

**Black List:**

---

**Question 5:**

Imagine you are developing a music playlist application where each song is represented by an integer, indicating its popularity. The users want to create playlists with the longest increasing subsequences of popular songs. Given an array of integers representing the popularity of songs, your task is to implement a function to find the count of all possible longest increasing subsequences for a playlist.

**Input Format:**

- The first line contains an integer n (1 <= n <= 100), representing the number of elements in the array.
- The second line contains n space-separated integers, representing the elements of the array.
-

**Output Format:**

- The program will output a single line.
- It will print "The count of all possible longest increasing subsequences is: " followed by an integer, which represents the count of all possible longest increasing subsequences in the given array.

**Title for Question 5:** Longest Increasing Subsequence (Count of LIS)

**Solution:**

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

int countLIS(vector<int>& nums) {
    int n = nums.size();
    if (n <= 1) {
        return n;
    }

    vector<int> lengths(n, 1);
    vector<int> counts(n, 1);

    int max_length = 1;

    for (int i = 1; i < n; ++i) {
        for (int j = 0; j < i; ++j) {
            if (nums[i] > nums[j]) {
                if (lengths[j] + 1 > lengths[i]) {
                    lengths[i] = lengths[j] + 1;
                    counts[i] = counts[j];
                } else if (lengths[j] + 1 == lengths[i]) {
                    counts[i] += counts[j];
                }
            }
        }
        max_length = max(max_length, lengths[i]);
    }

    int result = 0;
    for (int i = 0; i < n; ++i) {
        if (lengths[i] == max_length) {
            result += counts[i];
        }
    }

    return result;
}
```

```
int main() {
    int n;
    cin >> n;
    vector<int> nums(n);
    for (int i = 0; i < n; ++i) {
        cin >> nums[i];
    }
    int result = countLIS(nums);
    cout << "The count of all possible longest increasing subsequences is
    return 0;
}
```

**TestCases:**

| S.No | Inputs | Outputs |
|---|---|---|
| 1 | 5 1 3 5 4 7 | The count of all possible longest increasing subsequences is: 2 |
| 2 | 8 10 22 9 33 21 50 41 60 | The count of all possible longest increasing subsequences is: 2 |
| 3 | 6 3 10 2 1 20 4 | The count of all possible longest increasing subsequences is: 1 |
| 4 | 7 3 2 4 2 6 7 8 | The count of all possible longest increasing subsequences is: 2 |
| 5 | 4 10 9 2 5 | The count of all possible longest increasing subsequences is: 1 |
| 6 | 4 4 3 2 1 | The count of all possible longest increasing subsequences is: 4 |

**White List:**

**Black List:**

---

**Question 6:**

Imagine you are working on a project that involves analyzing data sequences, specifically focusing on bitonic subsequences. In a given dataset representing the values of a variable over time, you want to find the length of the longest bitonic subsequence. This subsequence should exhibit a pattern where it initially increases, reaches a peak, and then starts decreasing.

**Input Format:**

- The first line contains an integer n, representing the size of the array.
- The second line contains n space-separated integers, representing the elements of the array.

**Output Format:**

- A single line containing an integer, representing the length of the longest bitonic subsequence in the given array.

**Title for Question 6:** Longest Bitonic Subsequence

**Solution:**

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

int bitonicLength(vector<int>& arr) {
    int n = arr.size();

    // LIS array to store the length of the longest increasing subsequenc
    vector<int> lis(n, 1);

    // Compute LIS from left to right
    for (int i = 1; i < n; ++i) {
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j] && lis[i] < lis[j] + 1) {
                lis[i] = lis[j] + 1;
            }
        }
    }

    // LDS array to store the length of the longest decreasing subsequenc
    vector<int> lds(n, 1);

    // Compute LDS from right to left
    for (int i = n - 2; i >= 0; --i) {
        for (int j = n - 1; j > i; --j) {
            if (arr[i] > arr[j] && lds[i] < lds[j] + 1) {
                lds[i] = lds[j] + 1;
            }
        }
    }

    int maxBitonicLength = 1;

    // Find the maximum length of bitonic subsequence
    for (int i = 0; i < n; ++i) {
        maxBitonicLength = max(maxBitonicLength, lis[i] + lds[i] - 1);
    }

    return maxBitonicLength;
}

int main() {
    int n;
    cin >> n;
    vector<int> arr(n);
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }
    int result = bitonicLength(arr);
    cout << "Length of the longest bitonic subsequence: " << result << en
    return 0;
}
```

**TestCases:**

| S.No | Inputs | Outputs |
|---|---|---|

| | | |
|---|---|---|
| 1 | 5 1 2 3 4 5 | Length of the longest bitonic subsequence: 5 |
| 2 | 8 5 4 3 2 1 2 3 4 | Length of the longest bitonic subsequence: 5 |
| 3 | 10 1 2 3 4 5 4 3 2 1 6 | Length of the longest bitonic subsequence: 9 |
| 4 | 6 3 5 2 6 4 8 | Length of the longest bitonic subsequence: 4 |
| 5 | 9 10 22 9 33 21 50 41 60 80 | Length of the longest bitonic subsequence: 6 |
| 6 | 7 1 2 3 4 5 6 7 | Length of the longest bitonic subsequence: 7 |

**White List:**

**Black List:**

---