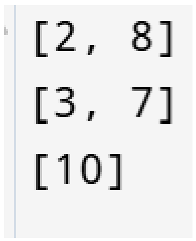
Extra assign 3

ARRAY SUBSET WITH SUM

1)

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

const int MAX_SIZE = 100;  // A maximum limit for the array size, adjust as needed

// Function to print the current subset
void printSubset(int current[], int size) {
   cout << "[";
   for (int i = 0; i < size; ++i) {
      cout << current[i];
      if (i != size - 1) cout << ", ";
   }
   cout << "]\n";
}

// Function to find subsets that sum to the target using a fixed-size array
void findSubsets(int arr[], int n, int index, int target, int current[], int currSize) {
   if (target == 0) {
      // If we found a valid subset, print it
      printSubset(current, currSize);
      return;
   }

   if (index >= n || target < 0) return;  // Base case

   // Include current element
   current[currSize] = arr[index];
   findSubsets(arr, n, index + 1, target - arr[index], current, currSize + 1);

   // Exclude current element
   findSubsets(arr, n, index + 1, target, current, currSize);
}

int main() {
   int arr[] = {2, 3, 7, 8, 10};  // Array of elements
```

```
    int target = 10;
    int n = sizeof(arr) / sizeof(arr[0]);  // Array length
    int current[MAX_SIZE];  // Array to store the current subset
    findSubsets(arr, n, 0, target, current, 0);
    return 0;
}
```

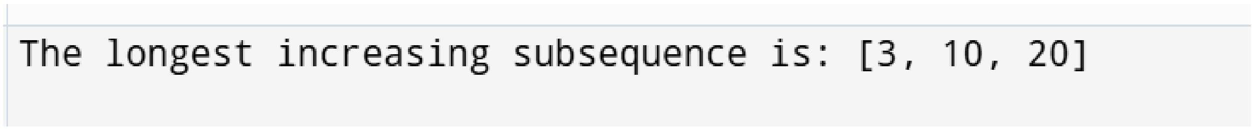


```
[2, 8]
[3, 7]
[10]
```

2)

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

// Function to find the length of Longest Increasing Subsequence (LIS)
void LIS(int arr[], int n) {
    if (n == 0) {
        cout << "The array is empty.\n";
        return;
    }

    int dp[n];  // dp[i] will store the length of the LIS ending at index i
    int previous[n];  // Array to track the previous index in the LIS

    // Initialize dp and previous arrays
    for (int i = 0; i < n; ++i) {
        dp[i] = 1;  // Each element is an LIS of length 1 by itself
        previous[i] = -1;  // No previous element initially
    }

    // Compute dp[] values in a bottom-up manner
    for (int i = 1; i < n; ++i) {
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j] && dp[i] < dp[j] + 1) {
                dp[i] = dp[j] + 1;
                previous[i] = j;
            }
        }
    }

    // Find the index of the maximum value in dp[] which represents the end of the LIS
    int maxLength = 0;
    int maxIndex = -1;
    for (int i = 0; i < n; ++i) {
        if (dp[i] > maxLength) {
            maxLength = dp[i];
            maxIndex = i;
        }
    }

    // Reconstruct the LIS by tracing the previous[] array
    int lis[maxLength];
    int index = maxLength - 1;
    while (maxIndex != -1) {
        lis[index--] = arr[maxIndex];
        maxIndex = previous[maxIndex];
    }

    // Output the LIS
    cout << "The longest increasing subsequence is: [";
    for (int i = 0; i < maxLength; ++i) {
        cout << lis[i] << (i == maxLength - 1 ? "" : ", ");
    }
    cout << "]\n";
}

int main() {
    int arr[] = {3, 10, 2, 1, 20};
    int n = sizeof(arr) / sizeof(arr[0]);

    LIS(arr, n);

    return 0;
```

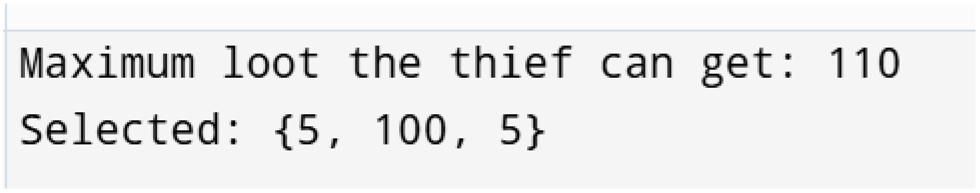
```
}
```



```
The longest increasing subsequence is: [3, 10, 20]
```

3) MAXIMIZE WITHOUT CONSECUTIVE

```
#include <iostream>
#include <algorithm>  // For max()
using namespace std;

int maxLoot(int hval[], int n) {
    if (n == 0) return 0;
    if (n == 1) return hval[0];

    int dp[n];  // dp[i] stores the maximum loot till house i

    // Initialize the first two houses
    dp[0] = hval[0];  // Maximum loot when there is only one house
    dp[1] = max(hval[0], hval[1]);  // Maximum loot when there are two houses

    // Calculate the maximum loot for each house from 2 to n-1
    for (int i = 2; i < n; ++i) {
        dp[i] = max(dp[i-1], hval[i] + dp[i-2]);
    }

    return dp[n-1];  // Maximum loot from all houses
}

void printLoot(int hval[], int n) {
    int dp[n];  // dp[i] stores the maximum loot till house i

    dp[0] = hval[0];
    dp[1] = max(hval[0], hval[1]);

    for (int i = 2; i < n; ++i) {
        dp[i] = max(dp[i-1], hval[i] + dp[i-2]);
    }
    // Reconstruct the selected houses (loot)
    int i = n - 1;
    cout << "Selected: {";
    while (i >= 0) {
        if (i == 0 || dp[i] != dp[i-1]) {
            cout << hval[i] << (i == 0 ? "" : ", ");
            i -= 2;  // Skip the next house as it's looted
        } else {
            i--;  // Move to the previous house if not looting the current one
        }
    }
    cout << "}\n";
}

int main() {
    int hval[] = {5, 5, 10, 100, 10, 5};
    int n = sizeof(hval) / sizeof(hval[0]);

    cout << "Maximum loot the thief can get: " << maxLoot(hval, n) << endl;
    printLoot(hval, n);

    return 0;
}
```

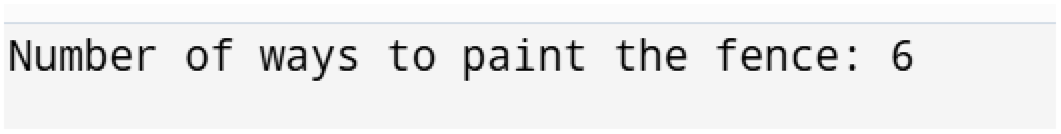


```
Maximum loot the thief can get: 110
Selected: {5, 100, 5}
```

WAYS TO PAINT SO NO MORE THAN 2 CONSECUTIVE R SAME

4)

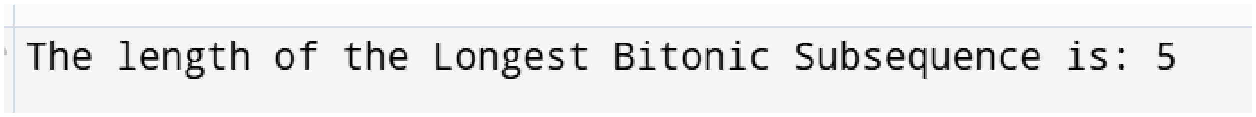
```
#include <iostream>
using namespace std;

int countWays(int n, int k) {
    if (n == 0) return 0;
    if (n == 1) return k;

    int dp[n + 1];
    dp[0] = 0;
    dp[1] = k;
    dp[2] = k * k;

    for (int i = 3; i <= n; ++i) {
        dp[i] = (k - 1) * (dp[i - 1] + dp[i - 2]);
    }

    return dp[n];
}

int main() {
    int n = 3, k = 2;
    cout << "Number of ways to paint the fence: " << countWays(n, k) << endl;
    return 0;
}
```



```
Number of ways to paint the fence: 6
```

5)

```
#include <iostream>
#include <algorithm>  // For max()

using namespace std;

int longestBitonicSubsequence(int arr[], int n) {
    if (n == 0) return 0;

    int inc[n], dec[n];
    fill(inc, inc + n, 1);
    fill(dec, dec + n, 1);

    // Calculate LIS (increasing subsequence)
    for (int i = 1; i < n; ++i) {
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                inc[i] = max(inc[i], inc[j] + 1);
            }
        }
    }

    // Calculate LDS (decreasing subsequence)
    for (int i = n - 2; i >= 0; --i) {
        for (int j = n - 1; j > i; --j) {
            if (arr[i] > arr[j]) {
                dec[i] = max(dec[i], dec[j] + 1);
            }
        }
    }

    // Find the maximum value of inc[i] + dec[i] - 1
    int maxLBS = 0;
    for (int i = 0; i < n; ++i) {
        maxLBS = max(maxLBS, inc[i] + dec[i] - 1);
    }

    return maxLBS;
}

int main() {
    int arr[] = {12, 11, 40, 5, 3, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "The length of the Longest Bitonic Subsequence is: " <<
longestBitonicSubsequence(arr, n) << endl;
    return 0;
}
```



```
The length of the Longest Bitonic Subsequence is: 5
```