C++

#include <iostream>

#include <algorithm>

#include <limits>

using namespace std;

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

    sort(arr, arr + n);

}

int queryByIndex(int arr[], int n, int index) {

    if (index >= 0 && index < n) {

        return arr[index];

    } else {

        cout << "Invalid index. Please enter a valid index between 0 and " << n - 1 << endl;

        return -1;

    }

}

int main() {

    int n;

    cout << "Enter the number of elements (max 1000): ";

    cin >> n;

    // Validate input size

    if (n <= 0 || n > 1000) {

        cout << "Invalid input size. Please enter a value between 1 and 1000." << endl;

        return 1;

    }

    int arr[1000]; // Fixed-size array to prevent buffer overflow

    cout << "Enter the elements: ";

    for (int i = 0; i < n; i++) {

        cin >> arr[i];

    }

    sortArray(arr, n);

    cout << "Sorted array: ";

    for (int i = 0; i < n; i++) {

        cout << arr[i] << " ";

    }

    cout << endl;

    int queryIndex;

    while (true) {

        cout << "Enter the index to query (or -1 to exit): ";

        cin >> queryIndex;

        if (queryIndex == -1) {

            break;

        }

        int result = queryByIndex(arr, n, queryIndex);

        if (result != -1) {

            cout << "Element at index " << queryIndex << ": " << result << endl;

        }

    }

    return 0;

}

Use code with caution.

**Improvements:**

1. **Fixed-Size Array:**
   * The arr array is declared with a fixed size of 1000 to prevent buffer overflows. This limits the maximum number of elements the user can input.
2. **Input Validation:**
   * The code now validates the input size n to ensure it's within a safe range (1-1000). This prevents potential issues with memory allocation and array access.

**Note:**

While this approach mitigates the immediate buffer overflow risk, it's still not entirely foolproof. A more robust solution would involve using dynamic memory allocation techniques like malloc or new with careful memory management. However, this requires more advanced programming skills and careful attention to memory leaks and other potential issues.

It's also important to consider other security vulnerabilities, such as integer overflow, format string attacks, and injection attacks. By following secure coding practices and regularly reviewing and testing your code, you can significantly improve its security.