

TASK 1

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

```
struct Product {
    int id;
    std::string name;
    int quantity;
    // Add more fields as needed (e.g., price, description)
};
```

```
std::vector<Product> inventory;
```

```
void addProduct() {
```

```
    Product product;
```

```
    std::cout << "Enter product details:\n";
```

```
    std::cout << "ID: ";
```

```
    std::cin >> product.id;
```

```
    // Ensure unique ID:
```

```
    if (std::find_if(inventory.begin(), inventory.end(),
```

```
        [&product](const Product& p) { return p.id == product.id; }) != inventory.end()) {
```

```
        std::cout << "Error: Product with ID " << product.id << " already exists.\n";
```

```
        return;
```

```
    }
```

```
    std::cout << "Name: ";
```

```
    std::cin >> product.name;
```

```
    std::cout << "Quantity: ";
```

```
    std::cin >> product.quantity;
```

```

    inventory.push_back(product);
    std::cout << "Product added successfully!\n";
}

void removeProduct() {
    int idToRemove;
    std::cout << "Enter product ID to remove: ";
    std::cin >> idToRemove;

    auto it = std::find_if(inventory.begin(), inventory.end(),
        [idToRemove](const Product& p) { return p.id == idToRemove; });

    if (it != inventory.end()) {
        inventory.erase(it);
        std::cout << "Product removed successfully!\n";
    } else {
        std::cout << "Product not found.\n";
    }
}

// Additional functions for a complete system:
void displayInventory() {
    // Implement logic to display the inventory items
}

void searchProduct() {
    // Implement logic to search for products by ID or name
}

// ... (other functions as needed)

int main() {
    int choice;
    do {

```

```

    std::cout << "\nInventory Management System\n";
    std::cout << "1. Add Product\n";
    std::cout << "2. Remove Product\n";
    std::cout << "3. Display Inventory\n";
    std::cout << "4. Search Product\n";
    // Add more options as needed
    std::cout << "0. Exit\n";
    std::cout << "Enter your choice: ";
    std::cin >> choice;

    switch (choice) {
        case 1:
            addProduct();
            break;
        case 2:
            removeProduct();
            break;
        case 3:
            displayInventory();
            break;
        case 4:
            searchProduct();
            break;
        // Handle other choices
        case 0:
            std::cout << "Exiting...\n";
            break;
        default:
            std::cout << "Invalid choice.\n";
    }
    } while (choice != 0);

    return 0;
}

```

OUTPUT

```
Inventory Management System
1. Add Product
2. Remove Product
3. Display Inventory
4. Search Product
0. Exit
Enter your choice: 1
Enter product details:
ID: 2
Name: SHYAN
Quantity: 2
Product added successfully!
```

TASK 2

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

```
int main() {
    // Create a vector of 100,000 integers in descending order
    std::vector<int> numbers(100000);
    for (int i = 0; i < 100000; ++i) {
        numbers[i] = 100000 - i;
    }

    // Measure the execution time of Bubble Sort
    auto startBubbleSort = std::chrono::high_resolution_clock::now();
    for (int i = 0; i < numbers.size() - 1; ++i) {
        for (int j = 0; j < numbers.size() - i - 1; ++j) {
            if (numbers[j] > numbers[j + 1]) {
                std::swap(numbers[j], numbers[j + 1]);
            }
        }
    }
}
```

```

    }
    auto endBubbleSort = std::chrono::high_resolution_clock::now();
    auto durationBubbleSort = std::chrono::duration_cast<std::chrono::milliseconds>(endBubbleSort -
startBubbleSort);

    // Measure the execution time of STL sort algorithm
    auto startSTLSort = std::chrono::high_resolution_clock::now();
    std::sort(numbers.begin(), numbers.end());
    auto endSTLSort = std::chrono::high_resolution_clock::now();
    auto durationSTLSort = std::chrono::duration_cast<std::chrono::milliseconds>(endSTLSort -
startSTLSort);

    // Print the execution times
    std::cout << "Bubble Sort Execution Time: " << durationBubbleSort.count() << " milliseconds" <<
std::endl;
    std::cout << "STL Sort Execution Time: " << durationSTLSort.count() << " milliseconds" << std::endl;

    return 0;
}

```

OUTPUT

```

Bubble Sort Execution Time: 12296 milliseconds
STL Sort Execution Time: 2 milliseconds

```

CMD System Info

Command Prompt

```
Hotfix(s):              7 Hotfix(s) Installed.
                        [01]: KB5032005
                        [02]: KB5031988
                        [03]: KB5011048
                        [04]: KB5015684
                        [05]: KB5033372
                        [06]: KB5014032
                        [07]: KB5032907

Network Card(s):        3 NIC(s) Installed.
                        [01]: Intel(R) Dual Band Wireless-AC 8265
                             Connection Name: Wi-Fi
                             DHCP Enabled:   Yes
                             DHCP Server:    192.168.18.1
                             IP address(es)
                                [01]: 192.168.18.140
                                [02]: fe80::7c21:7e25:7eba:8014
                                [03]: 2407:d000:f:950b:1c32:7fcb:4f98:9ebe
                                [04]: 2407:d000:f:950b:f6:6bab:8d8d:f937
                        [02]: Realtek PCIe GbE Family Controller
                             Connection Name: Ethernet
                             Status:          Media disconnected
                        [03]: Bluetooth Device (Personal Area Network)
                             Connection Name: Bluetooth Network Connection
                             Status:          Media disconnected

Hyper-V Requirements:  VM Monitor Mode Extensions: Yes
                       Virtualization Enabled In Firmware: Yes
                       Second Level Address Translation: Yes
                       Data Execution Prevention Available: Yes

C:\Users\Shayan Awan>
```