Question 1:

You're working on a project where you're tasked with developing a feature for an e-commerce platform that allows users to sort products based on their prices. As part of this feature, you decide to implement a sorting algorithm to arrange the products in ascending order of price.

The project manager suggests using the bubble sort algorithm for simplicity and ease of implementation. They want to ensure that the sorting process is efficient, especially considering that the platform may have thousands of products.

Input Format:

- The user is prompted to input the size of the array (n), where n is a positive integer.
- Subsequently, the user inputs the elements of the array.

Output Format:

- The program display the original array before sorting.
- The program display the sorted array after applying the bubble sort algorithm.

Title for Question 1: Bubble Sort Implementation

Solution:

```
#include <algorithm>
#include <iostream>
#include <iterator>
#include <vector>
// Function to perform bubble sort algorithm
template <typename RandomAccessIterator>
void bubble_sort(RandomAccessIterator begin, RandomAccessIterator end) {
  bool swapped = true;
  while (begin != end-- && swapped) {
    swapped = false;


    for (auto i = begin; i != end; ++i) {
      if (*(i + 1) < *i) {
        std::iter_swap(i, i + 1);
        swapped = true;
      }
    }
  }
}


int main() {
  // Prompting the user to input the number of elements in the array
  //std::cout << "Enter the number of elements: ";
```

```
  int size;
  std::cin >> size;


  // Initializing a vector to store dynamic input elements
  std::vector<int> numbers(size);


  // Prompting the user to input the array elements
  //std::cout << "Enter the elements:\n";
  for (int i = 0; i < size; ++i) {
    std::cin >> numbers[i];
  }


  // Displaying the original numbers in the array
  std::cout << "Original numbers:\n";
  std::copy(numbers.begin(), numbers.end(), std::ostream_iterator<int>(st
  std::cout << "\n";


  // Sorting the array using bubble_sort function
  bubble_sort(numbers.begin(), numbers.end());


  // Displaying the sorted array after bubble sort
  std::cout << "Sorted array:\n";
  std::copy(numbers.begin(), numbers.end(), std::ostream_iterator<int>(st
  std::cout << "\n";


  return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	5 9 3 7 1 5	Original numbers: 9 3 7 1 5 Sorted array: 1 3 5 7 9
2	6 20 15 10 5 25 30	Original numbers: 20 15 10 5 25 30 Sorted array: 5 10 15 20 25 30
3	4 -2 8 -5 12	Original numbers: -2 8 -5 12 Sorted array: -5 -2 8 12
4	3 100 50 75	Original numbers: 100 50 75 Sorted array: 50 75 100
5	7 1 2 3 4 5 6 7	Original numbers: 1 2 3 4 5 6 7 Sorted array: 1 2 3 4 5 6 7
6	3 24 11 21	Original numbers: 24 11 21 Sorted array: 11 21 24

White List:

Black List:

Question 2:

Implement a program using the bubble sort algorithm to find the second largest element in an array. Prompt the user to input the array size and elements, and output the array both before and after sorting. Display the second largest element along with its index.

Input Format:

- Prompt the user to enter the size of the array (n).
- Prompt the user to enter the elements of the array.

Output Format:

- Display the original array before sorting.
- Display the array after sorting using the bubble sort algorithm.
- Display the second largest element along with its index.

Title for Question 2: Finding Second Largest Element

Solution:

```
#include <iostream>


// Function to perform bubble sort
void bubbleSort(int arr[], int size) {
    for (int i = 0; i < size - 1; ++i) {
        for (int j = 0; j < size - i - 1; ++j) {
            if (arr[j] > arr[j + 1]) {
                // Swap if the elements are in the wrong order
                std::swap(arr[j], arr[j + 1]);
            }
        }
    }
}


int main() {
    int size;


    // Input array size from the user
    //std::cout << "Enter the size of the array: ";
    std::cin >> size;


    int arr[size];


    // Input array elements from the user
    //std::cout << "Enter the elements of the array:\n";
    for (int i = 0; i < size; ++i) {
        //std::cout << "Element " << i + 1 << ": ";
        std::cin >> arr[i];
    }


    // Output the array before sorting
```

```
    std::cout << "Array before sorting: ";
    for (int i = 0; i < size; ++i) {
        std::cout << arr[i] << " ";
    }
    std::cout << "\n";


    // Perform bubble sort
    bubbleSort(arr, size);


    // Output the array after sorting
    std::cout << "Array after sorting: ";
    for (int i = 0; i < size; ++i) {
        std::cout << arr[i] << " ";
    }
    std::cout << "\n";


    // Output the second largest element and its index
    std::cout << "Second largest element: " << arr[size - 2] << "\n";
    std::cout << "Index of the second largest element: " << size - 2 << "


    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	5 10 5 8 3 12	Array before sorting: 10 5 8 3 12 Array after sorting: 3 5 8 10 12 Second largest element: 10 Index of the second largest element: 3
2	4 7 2 11 6	Array before sorting: 7 2 11 6 Array after sorting: 2 6 7 11 Second largest element: 7 Index of the second largest element: 2
3	3 15 9 4	Array before sorting: 15 9 4 Array after sorting: 4 9 15 Second largest element: 9 Index of the second largest element: 1
4	6 1 14 8 6 11 3	Array before sorting: 1 14 8 6 11 3 Array after sorting: 1 3 6 8 11 14 Second largest element: 11 Index of the second largest element: 4
5	2 9 5	Array before sorting: 9 5 Array after sorting: 5 9 Second largest element: 5 Index of the second largest element: 0
6	7 4 12 8 6 15 3 10	Array before sorting: 4 12 8 6 15 3 10 Array after sorting: 3 4 6 8 10 12 15 Second largest element: 12 Index of the second largest element: 5

White List:

Black List:

Question 3:

Create a program that employs the insertion sort algorithm to sort an array of integers in ascending order. Prompt the user to input the number of elements and the array itself. After sorting, output the sorted array. Discuss the implementation of insertion sort and its efficiency compared to other sorting algorithms.

Input Format:

- The program takes a sequence of integers as input, separated by spaces.

Output Format:

- The program outputs the sorted sequence of numbers after performing the insertion sort algorithm.

Title for Question 3: Simple Insertion Sort

Solution:

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


// Function to perform insertion sort algorithm
void insertionSort(std::vector<int>& arr) {
    int n = arr.size();


    for (int i = 1; i < n; ++i) {
        int key = arr[i];
        int j = i - 1;


        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            --j;
        }


        arr[j + 1] = key;
    }
}


int main() {
    //std::cout << "Enter the number of elements: ";
    int n;
    std::cin >> n;


    //std::cout << "Enter the elements:\n";
    std::vector<int> arr(n);
    for (int i = 0; i < n; ++i) {
        std::cin >> arr[i];
    }


    // Sorting using insertion sort
```

```
    insertionSort(arr);


    // Displaying the sorted array
    std::cout << "Sorted array:\n";
    for (int i = 0; i < n; ++i) {
        std::cout << arr[i] << " ";
    }
    std::cout << "\n";


    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	5 12 5 8 2 10	Sorted array: 2 5 8 10 12
2	7 -3 0 7 1 -5 9 4	Sorted array: -5 -3 0 1 4 7 9
3	3 100 50 200	Sorted array: 50 100 200
4	1 999	Sorted array: 999
5	4 10 10 10 10	Sorted array: 10 10 10 10
6	9 8 7 6 5 4	Sorted array: 0 0 0 0 4 5 6 7 8

White List:

Black List:

Question 4:

Combine insertion sort with binary search to efficiently insert elements into a sorted array. Create a program that demonstrates this hybrid sorting approach, allowing the user to input elements and displaying the sorted array.

Input Format:

- The Program prompts the user to enter an integer n representing the number of elements in the array.
- Then user needs to enter second line contains n space-separated integers representing the elements of the array.

Output Format:

- The output consists of a single line displaying the sorted array after applying the hybrid sorting approach.

Title for Question 4: Combining Insertion Sort and Binary Search

Solution:

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


// Function to perform binary search to find the correct position for ins
int binarySearch(const std::vector<int>& arr, int key, int low, int high)
    while (low <= high) {
        int mid = low + (high - low) / 2;


        if (arr[mid] == key)
            return mid;
        else if (arr[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return low;
}


// Function to perform insertion sort with binary search
void insertionSortWithBinarySearch(std::vector<int>& arr) {
    for (int i = 1; i < arr.size(); ++i) {
        int key = arr[i];
        int insertIndex = binarySearch(arr, key, 0, i - 1);


        // Shift elements to make space for the key
        for (int j = i; j > insertIndex; --j) {
            arr[j] = arr[j - 1];
        }


        // Insert the key at the correct position
        arr[insertIndex] = key;
    }
}


int main() {
    std::vector<int> arr;


    // Input elements from the user
    int size, element;
    //std::cout << "Enter the size of the array: ";
    std::cin >> size;
    //std::cout << "Enter " << size << " elements: ";
    for (int i = 0; i < size; ++i) {
        std::cin >> element;
        arr.push_back(element);
    }


    // Perform insertion sort with binary search
    insertionSortWithBinarySearch(arr);
```

```
    // Display the sorted array
    std::cout << "Sorted Array: ";
    for (int num : arr) {
        std::cout << num << " ";
    }
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	5 15 8 23 10 5	Sorted Array: 5 8 10 15 23
2	7 -5 20 0 12 8 15 -3	Sorted Array: -5 -3 0 8 12 15 20
3	3 100 50 75	Sorted Array: 50 75 100
4	4 -10 0 10 -5	Sorted Array: -10 -5 0 10
5	6 9 3 15 7 2 11	Sorted Array: 2 3 7 9 11 15
6	8 1000 500 750 200 150 1200 900 600	Sorted Array: 150 200 500 600 750 900 1000 1200

White List:

Black List:

Question 5:

Imagine you are developing a system for a retail store to manage inventory. The store manager wants a program to sort the products in the inventory based on their prices using the Selection Sort algorithm. Customers often ask for products sorted by price to make browsing easier. Implement a program that allow the manager to enter the prices of products, and then it will display the products sorted by price. This will help the store organize its inventory more efficiently and provide a better shopping experience for customers.

Input Format:

- Prompt the user to enter a length of the array
- Prompt the user to enter the element in the array

Output Format:

- Display the original entered array element

Title for Question 5: Selection Sort Implementation for Integer Arrays

Solution:

```
#include <iostream>


void selectionSort(int arr[], int size) {
    for (int i = 0; i < size - 1; ++i) {
        int minIndex = i;
        for (int j = i + 1; j < size; ++j) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        std::swap(arr[i], arr[minIndex]);
    }
}


void displayArray(const int arr[], int size) {
    for (int i = 0; i < size; ++i) {
        std::cout << arr[i] << " ";
    }
    std::cout << "\n";
}


int main() {
    int size;


    //std::cout << "Enter the size of the array: ";
    std::cin >> size;


    int arr[size];


    //std::cout << "Enter the array elements: ";
    for (int i = 0; i < size; ++i) {
        std::cin >> arr[i];
    }


    std::cout << "Original array: ";
    displayArray(arr, size);


    selectionSort(arr, size);


    std::cout << "Sorted array: ";
    displayArray(arr, size);


    return 0;
}
```

TestCases:

S.No	Inputs	Outputs

1	5 7 2 9 1 5	Original array: 7 2 9 1 5 Sorted array: 1 2 5 7 9
2	8 15 4 8 12 6 1 10 7	Original array: 15 4 8 12 6 1 10 7 Sorted array: 1 4 6 7 8 10 12 15
3	3 10 5 2	Original array: 10 5 2 Sorted array: 2 5 10
4	6 3 6 1 9 4 7	Original array: 3 6 1 9 4 7 Sorted array: 1 3 4 6 7 9
5	4 8 3 2 6	Original array: 8 3 2 6 Sorted array: 2 3 6 8
6	7 5 10 2 8 7 1 9	Original array: 5 10 2 8 7 1 9 Sorted array: 1 2 5 7 8 9 10

White List:

Black List:

Question 6:

To implement a program using the selection sort algorithm, begin by prompting the user to input the size of the array dynamically. Subsequently, allocate memory for the array and request the user to enter the elements. Implement the selection sort algorithm and create a function for the swap operation. Display both the original and sorted arrays.

Input Format:

- Prompt the user to enter the size of the array (n).
- Dynamically allocate memory for an array of size n.
- Request the user to input n elements of the array.

Output Format:

- Display the original array before sorting.
- Output each step of the selection sort process, showing the array after each iteration.
- Display the final sorted array.
- Ensure clear and concise formatting for better readability.

Title for Question 6: Selection Sort

Solution:

```
#include <iostream>


void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}


void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; ++i) {
        // Find the minimum element in the unsorted part of the array
```

```
        int minIndex = i;
        for (int j = i + 1; j < n; ++j) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }


        // Swap the found minimum element with the first element
        swap(arr[i], arr[minIndex]);
    }
}


void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        std::cout << arr[i] << " ";
    std::cout << std::endl;
}
int main() {
    int n;
    // Get the size of the array from the user
    //std::cout << "Enter the number of elements: ";
    std::cin >> n;
    int *arr = new int[n];  // Dynamically allocate an array of size 'n'
    // Get array elements from the user
    //std::cout << "Enter the elements of the array:" << std::endl;
    for (int i = 0; i < n; ++i) {
       //std::cout << "Element " << i + 1 << ": ";
        std::cin >> arr[i];
    }
    std::cout << "Original array: ";
    printArray(arr, n);
    selectionSort(arr, n);
    std::cout << "Sorted array: ";
    printArray(arr, n);
    // Free the dynamically allocated memory
    delete[] arr;
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	5 2 9 1 5 6	Original array: 2 9 1 5 6 Sorted array: 1 2 5 6 9
2	3 1 3 5	Original array: 1 3 5 Sorted array: 1 3 5
3	6 10 8 6 4 2 9	Original array: 10 8 6 4 2 9 Sorted array: 2 4 6 8 9 10
4	9 3 5 2 5 2 3 7 1 4	Original array: 3 5 2 5 2 3 7 1 4 Sorted array: 1 2 2 3 3 4 5 5 7
5	4 1000 999 10000 1	Original array: 1000 999 10000 1 Sorted array: 1 999 1000 10000
6	2 285 381	Original array: 285 381 Sorted array: 285 381

White List:

Black List:
