CECOS UNIVERSITY

DATA STRUCTURE AND ALGORITHUMS

LAB TASK 3

Submitted by:
Misbah Ullah Jan
CU-4273-2023

Submitted to: HIJAB DURRANI

Date: 11/6/2025

TASK 1:

1. Modify the Bubble Sort so it stops early if no swaps occur during a pass (which means the list is already sorted).

Bubble Sort Algorithm:

1. Initiate the primary (outer) loop, which iterates from index i = 0 up to n-2, where n represents the total number of elements in the array. This loop governs the total number of sorting passes.
2. Within the scope of the outer loop, declare and initialize a boolean flag, swapped, to false. The purpose of this variable is to record the occurrence of any swaps within the current iteration (pass).
3. Initiate the secondary (inner) loop, iterating from index j = 0 up to n-i-2. This loop is responsible for executing the pairwise comparisons and subsequent transpositions for the active pass.
4. Within the inner loop's body, perform a comparison between the adjacent elements located at arr[j] and arr[j+1].
5. Evaluate the condition arr[j] > arr[j+1]. If this condition holds true (for an ascending sort): a. Execute a transposition of the elements arr[j] and arr[j+1]. b. Update the swapped flag to true, thereby indicating that at least one transposition has occurred during this pass.
6. Upon the completion of the inner loop (signaling the end of a single pass):
7. Examine the current state of the swapped boolean variable.
8. If the swapped variable remains false, this indicates that no elements were transposed during the completed pass. This state confirms that the array is fully sorted.
9. Consequently, if swapped is false, execute a break statement to exit the outer loop, thus terminating the sorting procedure prematurely.
10. Conversely, if swapped is true, the algorithm proceeds to the next iteration of the outer loop to commence the subsequent pass.
11. The algorithm concludes either upon the natural completion of the outer loop or through the aforementioned early termination condition.

Main.cpp + 443uxrve8

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

using namespace std;

void bubbleSortOptimized(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {

        bool swapped = false;

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

            if (arr[j] > arr[j + 1]) {

                swap(arr[j], arr[j + 1]);

                swapped = true;
            }
        }

        if (swapped == false) {

            break;
        }
    }
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}


int main() {
    int arr[] = {5, 1, 4, 2, 8};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    printArray(arr, n);

    bubbleSortOptimized(arr, n);

    cout << "Sorted array (Optimized Bubble Sort): ";
    printArray(arr, n);

    int sorted_arr[] = {1, 2, 3, 4, 5};
    int n_sorted = 5;

    cout << "\nAlready sorted array (original): ";
    printArray(sorted_arr, n_sorted);

    bubbleSortOptimized(sorted_arr, n_sorted);

    cout << "Already sorted array (after sort): ";
    printArray(sorted_arr, n_sorted);


    return 0;
}
```

Output:

```
Original array: 5 1 4 2 8
Sorted array (Optimized Bubble Sort): 1 2 4 5 8

Already sorted array (original): 1 2 3 4 5
Already sorted array (after sort): 1 2 3 4 5
```

TASK NO 2:

Modify your Selection Sort algorithm to sort an array in descending order instead of ascending.

Selection Sort Algorithm (Descending Order):

1. Start the outer loop that iterates from `i = 0` to `n-2` (where `n` is the number of elements). This loop marks the beginning of the unsorted subarray.
2. Inside the outer loop, assume the first element of the unsorted portion (at index `i`) is the largest. Store its index in a variable: `maxIndex = i`.
3. Start the inner loop that iterates from `j = i + 1` to `n-1`. This loop will scan the rest of the unsorted subarray to find the actual largest element.
4. Inside the inner loop, compare the element at the current index `j` with the element at `maxIndex`: `arr[j]` vs. `arr[maxIndex]`.
5. If `arr[j] > arr[maxIndex]` (for descending order), it means we have found a new largest element. Update `maxIndex` to `j`.
6. After the inner loop finishes, `maxIndex` will hold the index of the largest element in the unsorted portion of the array (from index `i` to `n-1`).
7. Swap the element at the beginning of the unsorted portion (index `i`) with the element at `maxIndex`.
8. The outer loop continues, and the sorted portion of the array grows from the left, while the unsorted portion shrinks.
9. The algorithm terminates when the outer loop finishes, and the array is fully sorted in descending order.

Main.cpp 443uxrve8

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

using namespace std;

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

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


        int maxIndex = i;


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

            if (arr[j] > arr[maxIndex]) {

                maxIndex = j;
            }
        }
        if (maxIndex != i) {
            swap(arr[i], arr[maxIndex]);
        }
    }
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int arr[] = {5, 1, 4, 2, 8};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    printArray(arr, n);

    selectionSortDescending(arr, n);

    cout << "Sorted array (Selection Sort Descending): ";
    printArray(arr, n);

    return 0;
}
```

Output:

```
Original array: 5 1 4 2 8
Sorted array (Selection Sort Descending): 8 5 4 2 1
```

TASK 3:

Write a program that:

1. Takes a list of 10 random numbers.

2. Sorts it using Bubble Sort and Selection Sort (both ascending).

3. Counts and prints the number of comparisons and swaps each algorithm performs.

Expected Output Example:

Original list: [9, 3, 7, 1, 5, 2, 6, 8, 4, 0]

Bubble Sort: Comparisons = 45, Swaps = 20

Selection Sort: Comparisons = 45, Swaps = 9

1. Modified Bubble Sort Function (with Counters)

Create a function bubbleSortCounter that takes an array, its size n, and references to comparisons and swaps counters.

1. Initialize comparisons and swaps to 0.
2. Start the outer loop i from 0 to n-2.
3. Start the inner loop j from 0 to n-i-2.
4. Inside the inner loop, **increment comparisons** (one comparison is about to happen).
5. Check if arr[j] > arr[j+1]: a. If true, swap(arr[j], arr[j+1]). b. If true, **increment swaps**.

2. Modified Selection Sort Function (with Counters)

Create a function selectionSortCounter that takes an array, its size n, and references to comparisons and swaps counters.

1. Initialize comparisons and swaps to 0.

2. Start the outer loop i from 0 to n-2.
3. Initialize minIndex = i.
4. Start the inner loop j from i+1 to n-1.
5. Inside the inner loop, **increment comparisons** (one comparison is about to happen).
6. Check if arr[j] < arr[minIndex]: a. If true, update minIndex = j.
7. After the inner loop finishes:
8. Check if minIndex != i: a. If true, swap(arr[i], arr[minIndex]). b. If true, **increment swaps**.

3. Main Program Logic

1. Define the original list of 10 numbers (e.g., originalList[10] = {9, 3, 7, 1, 5, 2, 6, 8, 4, 0}).
2. Create an identical copy of this list (e.g., listCopy[10]). This is crucial.
3. Declare counters for Bubble Sort: bubbleComparisons, bubbleSwaps.
4. Declare counters for Selection Sort: selectionComparisons, selectionSwaps.
5. Print the originalList.
6. Call bubbleSortCounter(originalList, 10, bubbleComparisons, bubbleSwaps).
7. Call selectionSortCounter(listCopy, 10, selectionComparisons, selectionSwaps).
8. Print the results for Bubble Sort (the values of bubbleComparisons and bubbleSwaps).
9. Print the results for Selection Sort (the values of selectionComparisons and selectionSwaps).

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

using namespace std;

void printArray(int arr[], int size) {
    cout << "[";
    for (int i = 0; i < size; i++) {
        cout << arr[i];
        if (i < size - 1) {
            cout << ", ";
        }
    }
    cout << "]" << endl;
}


void bubbleSortCounter(int arr[], int n, long long &comparisons, long long &swaps) {
    comparisons = 0;
    swaps = 0;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {

            comparisons++;
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);

            }
        }
    }
}


void selectionSortCounter(int arr[], int n, long long &comparisons, long long &swaps) {
    comparisons = 0;
    swaps = 0;
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {

            comparisons++;
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }

        if (minIndex != i) {
            swap(arr[i], arr[minIndex]);

            swaps++;
        }
    }
}
```

```
58
59  int main() {
60
61      int arr[] = {9, 3, 7, 1, 5, 2, 6, 8, 4, 0};
62      int n = 10;
63
64
65      int arrCopy[10];
66      for(int i = 0; i < n; i++) {
67          arrCopy[i] = arr[i];
68      }
69
70      cout << "Original list: ";
71      printArray(arr, n);
72      cout << endl;
73
74
75      long long bubbleComparisons = 0;
76      long long bubbleSwaps = 0;
77      bubbleSortCounter(arr, n, bubbleComparisons, bubbleSwaps);
78
79      cout << "Bubble Sort: Comparisons = " << bubbleComparisons << ", Swaps = " << bubbleS
80
81
82      long long selectionComparisons = 0;
83      long long selectionSwaps = 0;
84
85      selectionSortCounter(arrCopy, n, selectionComparisons, selectionSwaps);
86
87
88
89
90
91      return 0;
92  }
```

Output:

```
Original list: [9, 3, 7, 1, 5, 2, 6, 8, 4, 0]

Bubble Sort: Comparisons = 45, Swaps = 28
Selection Sort: Comparisons = 45, Swaps = 6
```

TASK NO 4:

4. Modify your sorting code (either Bubble or Selection) to:

- **Print the array after each pass of the outer loop.**
- **Show which elements were swapped in that pass.**

ALGORITHUM Selection Sort:

1. Start the outer loop that iterates from i = 0 to n-2. This loop represents each pass.
2. Inside the outer loop, assume the first element of the unsorted portion is the minimum: minIndex = i.
3. Start the inner loop j from i + 1 to n-1 to find the actual minimum element in the unsorted portion.
4. Inside the inner loop, compare arr[j] with arr[minIndex].
5. If arr[j] < arr[minIndex], update minIndex = j.
6. After the inner loop finishes, minIndex holds the index of the minimum element.
7. Store the values for logging: Before swapping, save the value at the current pass's start: valueAtI = arr[i]. Also, save the minimum value found: valueAtMin = arr[minIndex].
8. Perform the swap: Check if minIndex != i. a. If true, swap(arr[i], arr[minIndex]).
9. Print the pass results: a. Print the pass number (e.g., Pass i+1:). b. Print the entire array in its current state. c. If minIndex != i (meaning a swap *did* happen), print (swapped valueAtI and valueAtMin). d. If minIndex == i (no swap needed), you can optionally print (no swap).
10. The outer loop continues to the next pass.
11. The algorithm terminates when the outer loop is complete.

Main.cpp 443uxrve8

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

void printArrayFull(int arr[], int size) {
    cout << "[";
    for (int i = 0; i < size; i++) {
        cout << arr[i];
        if (i < size - 1) {
            cout << ", ";
        }
    }
    cout << "]";
}

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


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


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

        int valueAtI = arr[i];

        int valueAtMin = arr[minIndex];


        if (minIndex != i) {
            swap(arr[i], arr[minIndex]);
        }

        cout << "Pass " << (i + 1) << ": ";
        printArrayFull(arr, n);

        if (minIndex != i) {
            cout << " (swapped " << valueAtI << " and " << valueAtMin << ")" << endl;
        } else {
            cout << " (no swap)" << endl;
        }
    }
}


int main() {

    int arr[] = {5, 2, 3, 8, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original list: ";
    printArrayFull(arr, n);
```

```
    cout << "\n" << endl;

    selectionSortVerbose(arr, n);

    cout << "\nFinal Sorted list: ";
    printArrayFull(arr, n);
    cout << endl;

    return 0;
}
```

Output:

```
Original list: [5, 2, 3, 8, 6]

Pass 1: [2, 5, 3, 8, 6] (swapped 5 and 2)
Pass 2: [2, 3, 5, 8, 6] (swapped 5 and 3)
Pass 3: [2, 3, 5, 8, 6] (no swap)
Pass 4: [2, 3, 5, 6, 8] (swapped 8 and 6)

Final Sorted list: [2, 3, 5, 6, 8]
```

TASK NO 5:

Use Selection Sort to sort a list of strings alphabetically.

Main.cpp 443uxrve8

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

using namespace std;


void printStringVector(const vector<string>& arr) {
    cout << "[";
    for (size_t i = 0; i < arr.size(); ++i) {
        cout << "\"" << arr[i] << "\"";
        if (i < arr.size() - 1) {
            cout << ", ";
        }
    }
    cout << "]" << endl;
}


void selectionSortStrings(vector<string>& arr) {
    int n = arr.size();


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

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

            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }

        if (minIndex != i) {
            swap(arr[i], arr[minIndex]);
        }
    }
}

int main() {

    vector<string> arr = {"pear", "apple", "banana", "mango"};

    cout << "Original list: ";
    printStringVector(arr);

    selectionSortStrings(arr);

    cout << "Sorted list: ";
    printStringVector(arr);

    return 0;
}
```

Output:

```
Original list: [5, 2, 3, 8, 6]

Pass 1: [2, 5, 3, 8, 6] (swapped 5 and 2)
Pass 2: [2, 3, 5, 8, 6] (swapped 5 and 3)
Pass 3: [2, 3, 5, 8, 6] (no swap)
Pass 4: [2, 3, 5, 6, 8] (swapped 8 and 6)

Final Sorted list: [2, 3, 5, 6, 8]
```