Activity No. 7.1	
SORTING ALGORITHMS: BUBBLE, SELECTION, AND INSERTION SORT	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 10/16/2024
Section: CPE21S4	**Date Submitted:** 10/16/2024
Name(s): Reyes, Alexzander J.	**Instructor:** Prof. Maria Rizette Sayo

6. Output

Code + Console Screenshot

main.cpp

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main() {
    const int SIZE = 100;
    int arr[SIZE];

    // Seed the random number generator
    srand(static_cast<unsigned int>(time(0)));

    // Fill the array with random values
    for (int i = 0; i < SIZE; ++i) {
        arr[i] = rand() % 1000;  // Generate random numbers between 0 and 999
    }

    // Print the unsorted array
    cout << "Unsorted array: ";
    for (int i = 0; i < SIZE; ++i) {
        cout << arr[i] << " ";
    }

    cout << endl;

    return 0;
}
```

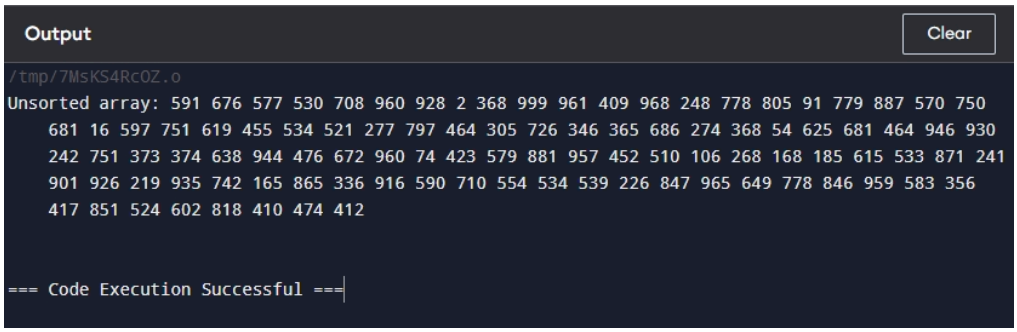
	Output: 
Observations	Rand() is used in the C++ code to generate an array of 100 random numbers between 0 and 999. Every time the program runs, the srand(time(0)) makes sure the numbers change. The numbers are printed in the order that they are generated, without any sorting.

Table 7-1. Array of Values for Sort Algorithm Testing

Code + Console Screenshot	Code:
	// SortingAlgorithms.h
	#ifndef SORTINGALGORITHMS_H
	#define SORTINGALGORITHMS_H
	
	class SortingAlgorithms {
	public:
	// Bubble Sort
	static void bubbleSort(int arr[], int size);
	
	// Insertion Sort
	static void insertionSort(int arr[], int size);
	
	// Selection Sort
	static void selectionSort(int arr[], int size);
	
	// Merge Sort
	static void mergeSort(int arr[], int left, int right);
	
	// Quick Sort
	static void quickSort(int arr[], int left, int right);
	
	private:
	// Helper function for Merge Sort to merge two subarrays
	static void merge(int arr[], int left, int mid, int right);
	
	// Helper function for Quick Sort to partition the array
	static int partition(int arr[], int left, int right);

```
};

#endif // SORTINGALGORITHMS_H
—————————————
// main.cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "SortingAlgorithms.h"  // Import the header file

using namespace std;

template <typename T>
void bubbleSort(T arr[], size_t arrSize) {
    for (size_t i = 0; i < arrSize - 1; i++) {
        for (size_t j = 0; j < arrSize - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}

int main() {
    const int SIZE = 100;
    int arr[SIZE];

    srand(static_cast<unsigned int>(time(0)));

    for (int i = 0; i < SIZE; ++i) {
        arr[i] = rand() % 1000;  // Generate random numbers between 0 and 999
    }

    cout << "Original array: ";
    for (int i = 0; i < SIZE; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

    bubbleSort(arr, SIZE);

    cout << "\nSorted array (Bubble Sort): ";
    for (int i = 0; i < SIZE; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
—————————————
```

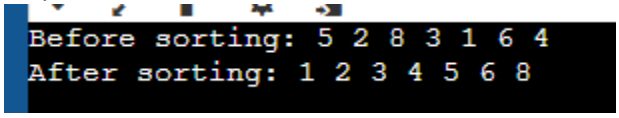
	Output: Original array: 386 262 778 674 137 120 978 94 255 467 422 859 972 956 652 528 347 34 899 996 938 680 775 948 428 380 514 105 891 827 612 278 89 743 304 227 863 634 673 118 101 96 330 74 404 982 954 752 16 205 748 307 237 524 255 665 256 121 122 499 948 734 129 390 829 434 617 693 420 642 163 522 738 493 948 143 475 254 895 844 459 995 151 696 519 758 713 127 879 835 627 828 921 756 218 751 542 187 796 963 Sorted array (Bubble Sort): 16 34 74 89 94 96 101 105 118 120 121 122 127 129 137 143 151 163 187 205 218 227 237 254 255 255 256 262 278 304 307 330 347 380 386 390 404 420 422 428 434 459 467 475 493 499 514 519 522 524 528 542 612 617 627 634 642 652 665 673 674 680 693 696 713 734 738 743 748 751 752 756 758 775 778 796 827 828 829 835 844 859 863 879 891 895 899 921 938 948 948 948 954 956 963 972 978 982 995 996
Observations	The SortingAlgorithms.h header file is well-structured, contains function declarations, includes guards to avoid multiple inclusions, and employs templates for flexibility. Sorting methods such as Bubble Sort are implemented in the.cpp file with clarity and efficiency by utilizing common library functions. The main function prints the original and sorted arrays and creates random integers to test the sorting.

Table 7-2. Bubble Sort Technique

Code + Console Screenshot	Code:
	// SortingAlgorithms.h
	#ifndef SORTINGALGORITHMS_H
	#define SORTINGALGORITHMS_H
	
	template <typename T>
	void selectionSort(T arr[], const int N);
	
	template <typename T>
	int Routine_Smallest(T A[], int K, const int arrSize);
	
	#endif // SORTINGALGORITHMS_H
	—----------
	// selectionSort.cpp
	#include "SortingAlgorithms.h"
	
	template <typename T>
	void selectionSort(T arr[], const int N) {
	  int POS, temp, pass = 0;
	  for(int i = 0; i < N; i++) {
	  POS = Routine_Smallest(arr, i, N);
	  temp = arr[i];
	  arr[i] = arr[POS];
	  arr[POS] = temp;
	  pass++;
	  }
	}
	
	template <typename T>
	int Routine_Smallest(T A[], int K, const int arrSize) {
	  int position, j;

```
    T smallestElem = A[K];
    position = K;
    for(int J = K + 1; J < arrSize; J++) {
        if(A[J] < smallestElem) {
            smallestElem = A[J];
            position = J;
        }
    }
    return position;
}

// Explicitly instantiate the template functions for int type
template void selectionSort<int>(int arr[], const int N);
template int Routine_Smallest<int>(int A[], int K, const int arrSize);
——------------------------------
// main.cpp
#include <iostream>
#include "SortingAlgorithms.h"

int main() {
    int arr[] = {5, 2, 8, 3, 1, 6, 4};
    const int N = sizeof(arr) / sizeof(arr[0]);

    std::cout << "Before sorting: ";
    for(int i = 0; i < N; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    selectionSort(arr, N);

    std::cout << "After sorting: ";
    for(int i = 0; i < N; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Output:

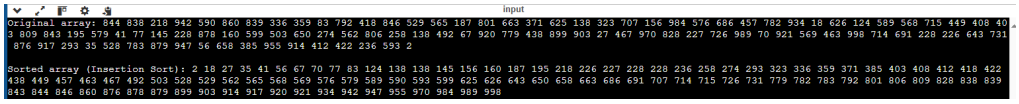


Observations	The SortingAlgorithms.h file does not contain the implementations of the selectionSort and Routine_Smallest functions, but it declares them as templates that enable them to operate on a variety of data types. These implementations are provided by the sortingAlgorithms.cpp file, which also makes use of templates and contains the header for function declarations. An integer array is sorted using the selectionSort method in

	the main.cpp file.

Table 7-3. Selection Sort Algorithm

Code + Console Screenshot	Code:
	// SortingAlgorithms.h
	#ifndef SORTINGALGORITHMS_H
	#define SORTINGALGORITHMS_H
	
	// Function declarations for sorting algorithms
	template <typename T>
	void insertionSort(T arr[], const int N);
	
	#endif // SORTINGALGORITHMS_H
	—----------
	// insertionSort.cpp
	#include "SortingAlgorithms.h"
	
	template <typename T>
	void insertionSort(T arr[], const int N) {
	  int K = 1;
	  while (K < N) {
	  T temp = arr[K];
	  int J = K - 1;
	
	  while (J >= 0 && temp < arr[J]) {
	  arr[J + 1] = arr[J];
	  J--;
	  }
	  arr[J + 1] = temp;
	  K++;
	  }
	}
	
	// Explicit instantiation for int type
	template void insertionSort<int>(int arr[], const int N);
	—----------
	//main.cpp
	#include <iostream>
	#include <cstdlib>
	#include <ctime>
	#include "SortingAlgorithms.h" // Import the header file
	
	using namespace std;
	
	int main() {
	  const int SIZE = 100;
	  int arr[SIZE];

	srand(static_cast<unsigned int>(time(0))); // Seed the random number generator
	
	for (int i = 0; i < SIZE; ++i) {
	arr[i] = rand() % 1000; // Generate random numbers between 0 and 999
	}
	
	cout << "Original array: ";
	for (int i = 0; i < SIZE; ++i) {
	cout << arr[i] << " ";
	}
	cout << endl;
	
	insertionSort(arr, SIZE); // Sort the array using Insertion Sort
	
	cout << "\nSorted array (Insertion Sort): ";
	for (int i = 0; i < SIZE; ++i) {
	cout << arr[i] << " ";
	}
	cout << endl;
	
	return 0;
	}
	—----------
	Output:
	 

Observations	The insertionSort template function is declared in the SortingAlgorithms.h file, which also uses include guards to stop multiple includes. This function is implemented and an integer version is expressly created in the insertionSort.cpp code. An array of random numbers is created in main.cpp, and after sorting, both the sorted and unsorted arrays are shown. For clarity, the code is divided into distinct files, and it generates distinct random numbers each time it executes by using srand(time(0)).

Table 7-4. Insertion Sort Algorithm

7. Supplementary Activity

INPUT:

main.cpp

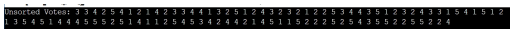
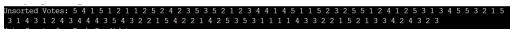
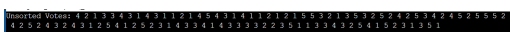
```cpp
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <string>

using namespace std;

// Define candidate names
const string candidates[] = {
    "Bo Dalton Capistrano",
    "Cornelius Raymon Agustin",
    "Deja Jayla Bañaga",
    "Lalla Brielle Yabut",
    "Franklin Relano Castro"
};

void voteCount(const vector<int>& votes) {
    const int maxCandidate = 5;
    vector<int> count(maxCandidate + 1, 0); // Count array for candidates 1 to 5

    // Count votes
    for (int vote : votes) {
        count[vote]++;
    }

    // Output results
    cout << "Vote Counts for Each Candidate:" << endl;
    for (int candidate = 1; candidate <= maxCandidate; candidate++) {
        cout << "Candidate " << candidate << ": " << candidates[candidate - 1] << " - " << count[candidate] << " votes" << endl;
    }

    // Determine the winning candidate
    int maxVotes = 0;
    int winningCandidate = -1;
    for (int candidate = 1; candidate <= maxCandidate; candidate++) {
        if (count[candidate] > maxVotes) {
            maxVotes = count[candidate];
            winningCandidate = candidate;
        }
    }

    cout << "Winning Candidate: " << "Candidate " << winningCandidate << ": " << candidates[winningCandidate - 1] << " with " << maxVotes << " votes." << endl;
}

int main() {
    srand(static_cast<unsigned int>(time(0))); // Seed for random number generation
    vector<int> votes(100); // Array to hold votes

    // Generate random votes between 1 and 5
    for (int i = 0; i < 100; i++) {
        votes[i] = rand() % 5 + 1; // Random number from 1 to 5
    }

    // Print unsorted votes
    cout << "Unsorted Votes: ";
    for (int vote : votes) {
        cout << vote << " ";
    }
    cout << endl;

    // Count votes
    voteCount(votes);

    return 0;
}
```

Pseudocode of Algorithm

*Untitled - Notepad

File Edit Format View Help

```
BEGIN
    // Define candidate names
    CANDIDATES = ["Bo Dalton Capistrano", "Cornelius Raymon Agustín", "Deja Jayla Bañaga", "Lalla Brielle Yabut", "Franklin Relano Castro"]

    // Initialize the number of votes
    NUM_VOTES = 100
    VOTES = ARRAY[NUM_VOTES]  // Array to hold the votes

    // Seed for random number generation (for simulation purposes)
    SEED_RANDOM()

    // Generate random votes between 1 and 5
    FOR i FROM 0 TO NUM_VOTES - 1 DO
        VOTES[i] = RANDOM_NUMBER(1, 5)  // Random number from 1 to 5
    END FOR

    // Print unsorted votes
    PRINT "Unsorted Votes: ", VOTES

    // Initialize count array for candidates
    COUNT = ARRAY[6] // Array to hold count for candidates 1 to 5 (index 0 is unused)

    // Count votes
    FOR EACH VOTE IN VOTES DO
        COUNT[VOTE] = COUNT[VOTE] + 1  // Increment the count for the corresponding candidate
    END FOR

    // Output results
    PRINT "Vote Counts for Each Candidate:"
    FOR candidate FROM 1 TO 5 DO
        PRINT "Candidate ", candidate, ": ", CANDIDATES[candidate - 1], " - ", COUNT[candidate], " votes"
    END FOR

    // Determine the winning candidate
    MAX_VOTES = 0
    WINNING_CANDIDATE = -1

    FOR candidate FROM 1 TO 5 DO
        IF COUNT[candidate] > MAX_VOTES THEN
            MAX_VOTES = COUNT[candidate]
            WINNING_CANDIDATE = candidate
        END IF
    END FOR

    // Print the winning candidate
    PRINT "Winning Candidate: Candidate ", WINNING_CANDIDATE, ": ", CANDIDATES[WINNING_CANDIDATE - 1], " with ", MAX_VOTES, " votes."
END
```

Output Console Showing Sorted Array	Manual Count	Count Result of Algorithm
	Vote Counts for Each Candidate: Candidate 1: Bo Dalton Capistrano - 17 votes Candidate 2: Cornelius Raymon Agustin - 24 votes Candidate 3: Deja Jayla Bañaga - 15 votes Candidate 4: Lalla Brielle Yabut - 22 votes Candidate 5: Franklin Relano Castro - 22 votes	Winning Candidate: Candidate 2: Cornelius Raymon Agustin with 24 votes.
	Vote Counts for Each Candidate: Candidate 1: Bo Dalton Capistrano - 22 votes Candidate 2: Cornelius Raymon Agustin - 22 votes Candidate 3: Deja Jayla Bañaga - 20 votes Candidate 4: Lalla Brielle Yabut - 18 votes Candidate 5: Franklin Relano Castro - 18 votes	Winning Candidate: Candidate 1: Bo Dalton Capistrano with 22 votes.
	Vote Counts for Each Candidate: Candidate 1: Bo Dalton Capistrano - 20 votes Candidate 2: Cornelius Raymon Agustin - 21 votes Candidate 3: Deja Jayla Bañaga - 24 votes Candidate 4: Lalla Brielle Yabut - 18 votes Candidate 5: Franklin Relano Castro - 17 votes	Winning Candidate: Candidate 3: Deja Jayla Bañaga with 24 votes.

Question: Was your developed vote counting algorithm effective? Why or why not?

Yes, the developed vote counting algorithm was effective because it efficiently counts votes using a counting sort approach, which operates in linear time (**O(n)**) for a small fixed range of candidates (1 to 5). Its straightforward implementation minimizes errors, and it provides clear outputs showing each candidate's vote count and the winning candidate. However, its effectiveness depends on accurate input data, and it is limited to a specific range of candidates, which could necessitate adjustments for larger datasets. Overall, it serves as a reliable method for counting votes efficiently.

8. Conclusion

Provide the following:

Summary of lessons learned

Sorting algorithms are essential for organizing data in many fields, including data analysis, e-commerce, and everyday tasks like managing emails and music playlists. Understanding these algorithms helps us process information more efficiently.

Analysis of the procedure

We studied various sorting algorithms and their effectiveness in different situations. For example, Quick Sort is great for large datasets, while Bubble Sort is simpler but works well for smaller ones. Each algorithm has its own strengths based on the data type and size.

Analysis of the supplementary activity

The supplementary activities showed how sorting algorithms are used in real life, like organizing medical records or e-commerce data. This helped me see the practical importance of these algorithms beyond just theory.

Concluding statement / Feedback: How well did you think you did in this activity? What are your areas for improvement?

I feel I did well in this activity by understanding both the theory and real-world applications of sorting algorithms. However, I want to improve my knowledge of more complex algorithms and their use in different technologies. My goal is to better match sorting methods to specific problems.

9. Assessment Rubric