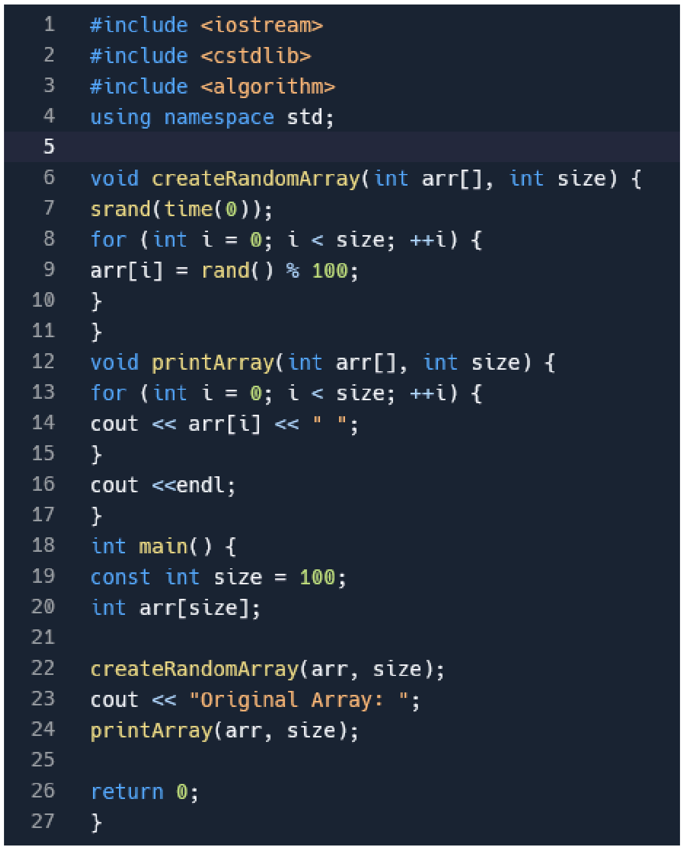
Activity No. <7.1>	
<Sorting Algorithms>	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 16/10/2024**
Section: CPE21S1	**Date Submitted: 16/10/2024**
Name(s): Sanchez, Justin Bjorn	**Instructor: Mrs. Maria Rizette Sayo**

6. Output

Code + Console Screenshot

Console:



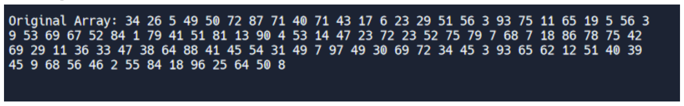
```cpp
#include <iostream>
#include <cstdlib>
#include <algorithm>
using namespace std;

void createRandomArray(int arr[], int size) {
srand(time(0));
for (int i = 0; i < size; ++i) {
arr[i] = rand() % 100;
}
}
void printArray(int arr[], int size) {
for (int i = 0; i < size; ++i) {
cout << arr[i] << " ";
}
cout <<endl;
}
int main() {
const int size = 100;
int arr[size];

createRandomArray(arr, size);
cout << "Original Array: ";
printArray(arr, size);

return 0;
}
```

Output:

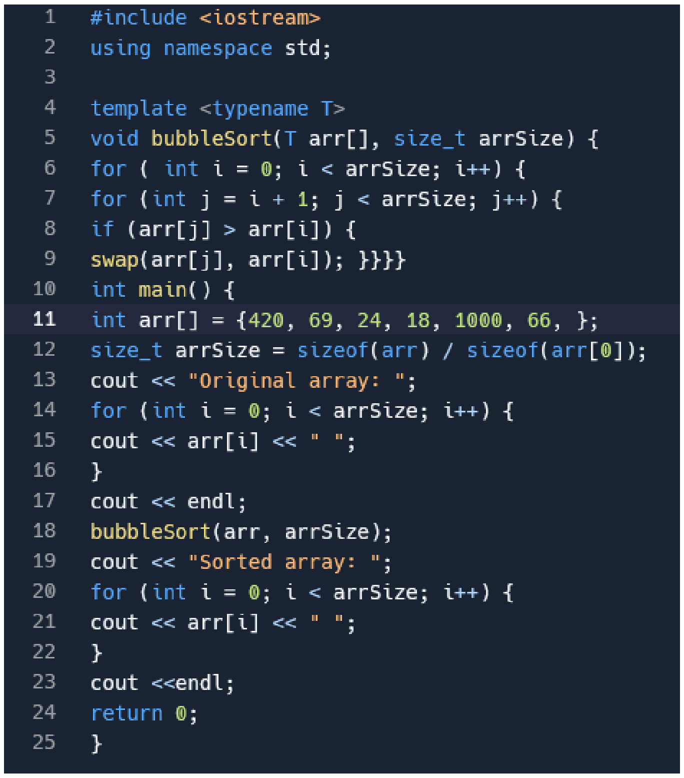
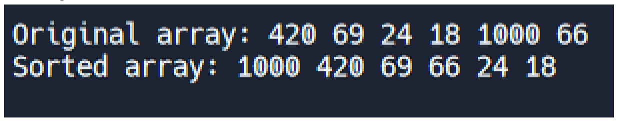


```
Original Array: 34 26 5 49 50 72 87 71 40 71 43 17 6 23 29 51 56 3 93 75 11 65 19 5 56 3
9 53 69 67 52 84 1 79 41 51 81 13 90 4 53 14 47 23 72 23 52 75 79 7 68 7 18 86 78 75 42
69 29 11 36 33 47 38 64 88 41 45 54 31 49 7 97 49 30 69 72 34 45 3 93 65 62 12 51 40 39
45 9 68 56 46 2 55 84 18 96 25 64 50 8
```

Observations

My observation on this is that the code generated 100 random elements in the array.

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

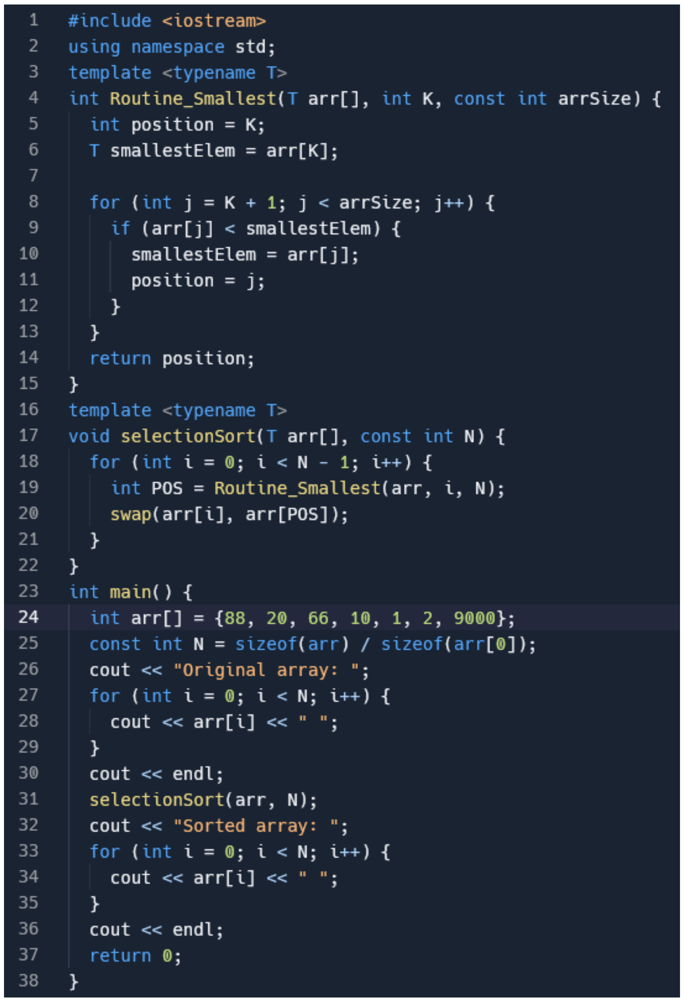
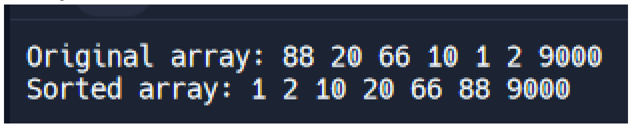
Code + Console Screenshot	Console:  Output: 
Observations	The bubble sort technique is repeatedly iterating through the list, contrasting neighboring items, and switching them if they are not in the correct order. The bubble sort method iterates across the list, contrasting neighboring items, and switching them if the sequence is incorrect.

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

template <typename T>
void bubbleSort(T arr[], size_t arrSize) {
for ( int i = 0; i < arrSize; i++) {
for (int j = i + 1; j < arrSize; j++) {
if (arr[j] > arr[i]) {
swap(arr[j], arr[i]); }}}}
int main() {
int arr[] = {420, 69, 24, 18, 1000, 66, };
size_t arrSize = sizeof(arr) / sizeof(arr[0]);
cout << "Original array: ";
for (int i = 0; i < arrSize; i++) {
cout << arr[i] << " ";
}
cout << endl;
bubbleSort(arr, arrSize);
cout << "Sorted array: ";
for (int i = 0; i < arrSize; i++) {
cout << arr[i] << " ";
}
cout <<endl;
return 0;
}
```

```
Original array: 420 69 24 18 1000 66
Sorted array: 1000 420 69 66 24 18
```

Table 7-2. Bubble Sort Technique

Code + Console Screenshot	Console:  Output: 

```cpp
#include <iostream>
using namespace std;
template <typename T>
int Routine_Smallest(T arr[], int K, const int arrSize) {
  int position = K;
  T smallestElem = arr[K];

  for (int j = K + 1; j < arrSize; j++) {
    if (arr[j] < smallestElem) {
      smallestElem = arr[j];
      position = j;
    }
  }
  return position;
}
template <typename T>
void selectionSort(T arr[], const int N) {
  for (int i = 0; i < N - 1; i++) {
    int POS = Routine_Smallest(arr, i, N);
    swap(arr[i], arr[POS]);
  }
}
int main() {
  int arr[] = {88, 20, 66, 10, 1, 2, 9000};
  const int N = sizeof(arr) / sizeof(arr[0]);
  cout << "Original array: ";
  for (int i = 0; i < N; i++) {
    cout << arr[i] << " ";
  }
  cout << endl;
  selectionSort(arr, N);
  cout << "Sorted array: ";
  for (int i = 0; i < N; i++) {
    cout << arr[i] << " ";
  }
  cout << endl;
  return 0;
}
```

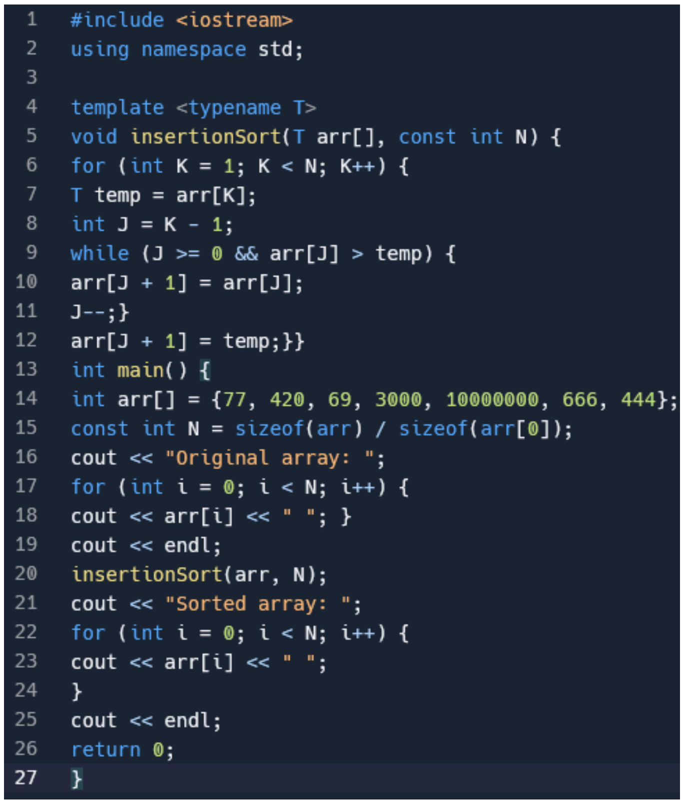
```
Original array: 88 20 66 10 1 2 9000
Sorted array: 1 2 10 20 66 88 9000
```

Observations	My observation on Routine_Smallest is that it's like the Bubble sort Technique but the opposite since it swaps the numbers again and again til it makes the correct order of array according to smallest value to biggest value.

Table 7-3. Selection Sort Algorithm

Code + Console Screenshot

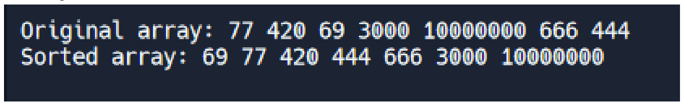
Console:



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

template <typename T>
void insertionSort(T arr[], const int N) {
for (int K = 1; K < N; K++) {
T temp = arr[K];
int J = K - 1;
while (J >= 0 && arr[J] > temp) {
arr[J + 1] = arr[J];
J--;}
arr[J + 1] = temp;}}
int main() {
int arr[] = {77, 420, 69, 3000, 10000000, 666, 444};
const int N = sizeof(arr) / sizeof(arr[0]);
cout << "Original array: ";
for (int i = 0; i < N; i++) {
cout << arr[i] << " "; }
cout << endl;
insertionSort(arr, N);
cout << "Sorted array: ";
for (int i = 0; i < N; i++) {
cout << arr[i] << " ";
}
cout << endl;
return 0;
}
```

Output:



```
Original array: 77 420 69 3000 10000000 666 444
Sorted array: 69 77 420 444 666 3000 10000000
```

Observations

My observation on this is that the insertionSort goes through every element on the original array and then puts it in the right ascending order.

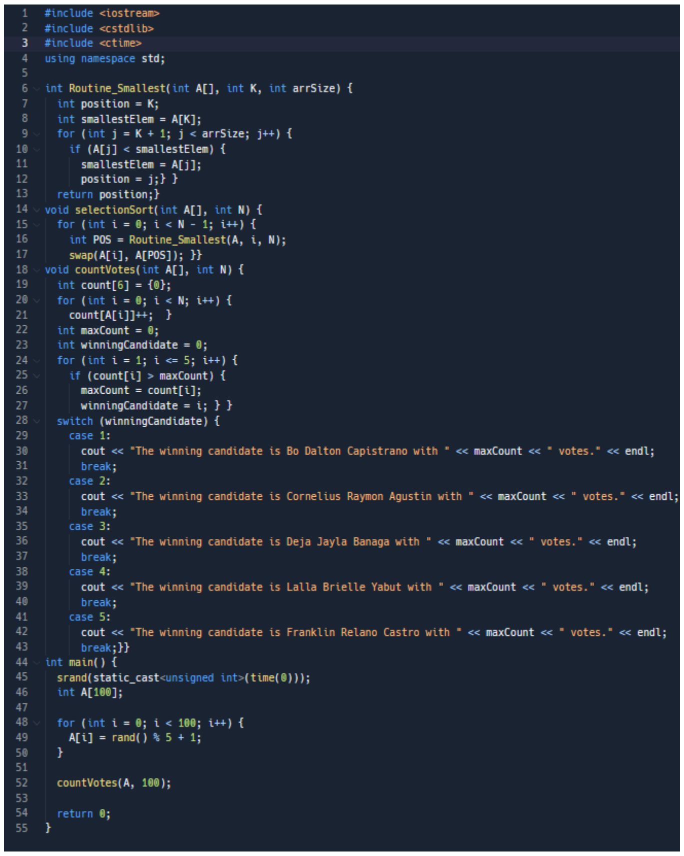
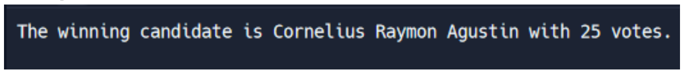
Table 7-4. Insertion Sort Algorithm

7. Supplementary Activity

Pseudocode:

```
Procedure CountVotes(votes)
  // Initialize a dictionary to store the vote counts
  vote_counts = {}
  // Loop through each vote
  for each vote in votes:
        // If the vote is already in the dictionary, increment its count
        if vote in vote_counts:
        vote_counts[vote] = vote_counts[vote] + 1
        // Otherwise, add the vote to the dictionary with a count of 1
        else:
        vote_counts[vote] = 1
```

	// Find the winning candidate
	winning_candidate = the key in vote_counts with the highest value
	winning_votes = the value associated with winning_candidate in vote_counts
	
	// Print the result
	print("The winning candidate is Candidate " + winning_candidate + " with " + winning_votes + " votes.")
	End Procedure

Console screenshot + Output	Console Screenshot:  Output: 

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

int Routine_Smallest(int A[], int K, int arrSize) {
  int position = K;
  int smallestElem = A[K];
  for (int j = K + 1; j < arrSize; j++) {
    if (A[j] < smallestElem) {
      smallestElem = A[j];
      position = j;} }
  return position;}
void selectionSort(int A[], int N) {
  for (int i = 0; i < N - 1; i++) {
    int POS = Routine_Smallest(A, i, N);
    swap(A[i], A[POS]); }}
void countVotes(int A[], int N) {
  int count[6] = {0};
  for (int i = 0; i < N; i++) {
    count[A[i]]++;  }
  int maxCount = 0;
  int winningCandidate = 0;
  for (int i = 1; i <= 5; i++) {
    if (count[i] > maxCount) {
      maxCount = count[i];
      winningCandidate = i; } }
  switch (winningCandidate) {
    case 1:
      cout << "The winning candidate is Bo Dalton Capistrano with " << maxCount << " votes." << endl;
      break;
    case 2:
      cout << "The winning candidate is Cornelius Raymon Agustin with " << maxCount << " votes." << endl;
      break;
    case 3:
      cout << "The winning candidate is Deja Jayla Banaga with " << maxCount << " votes." << endl;
      break;
    case 4:
      cout << "The winning candidate is Lalla Brielle Yabut with " << maxCount << " votes." << endl;
      break;
    case 5:
      cout << "The winning candidate is Franklin Relano Castro with " << maxCount << " votes." << endl;
      break;}}
int main() {
  srand(static_cast<unsigned int>(time(0)));
  int A[100];

  for (int i = 0; i < 100; i++) {
    A[i] = rand() % 5 + 1;
  }

  countVotes(A, 100);

  return 0;
}
```

```
The winning candidate is Cornelius Raymon Agustin with 25 votes.
```

Question: Justify why you chose to use this sorting algorithm.

- I chose this algorithm because for me it is much easier to find or identify which has the most quantity by using Routine_Smallest/Selection sort since it is simple yet effective also because it goes through the right order set for the array.

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

- It is effective because it is simple to understand and easy to use by counting the votes quickly by going through it 1 by 1 then putting the correct order.

Output Console Showing Sorted Array	Manual Count	Count Result of Algorithm
Sorted Votes: 1 1 2 2 3 4 5 5 5 5 ...	Candidate 1: 20 votes Candidate 2: 18 votes Candidate 3: 15 votes Candidate 4: 22 votes Candidate 5: 25 votes	The winning candidate is Cornelius Raymon Agustin with 25 votes.

8. Conclusion

To conclude this activity is that I learned how to arrange values in arrays by using different coding techniques and it made me learn more about arrays that it can be used to different stuff, also using this lab template, it guided me to learn by myself quickly and silently. The supplementary activity was not hard but tiring since there is too much to type. I think I did well in this activity because I finished it without problems.

9. Assessment Rubric