Activity No. 7

SORTING ALGORITHM: BUBBLE, SELECTION, AND INSERTION SORT

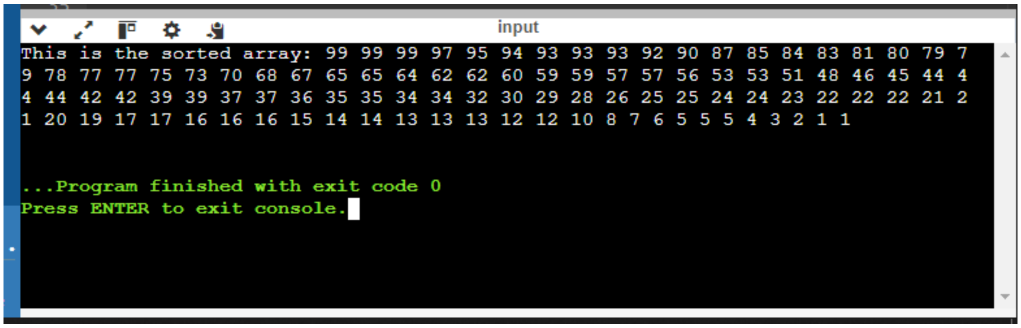
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 10/17/2024**
Section: CPE21S4	**Date Submitted: 10/17/2024**
Name(s): Santos Ma. Kassandra Nicole D,	**Instructor: Ma'am Rizette Sayo**

6. Output

Code +Console Screenshot

C/C++

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

vector<int> generateRandomArray(int size) {
    vector<int> array(size);
    srand(time(0));
    for(int i = 0; i < size; ++i) {
        array[i] = rand() % 100 + 1; // Random numbers between
1 and 100
    }
    return array;
}

void printArray(const vector<int>& array) {
    for(int num : array) {
        cout << num << " ";
    }
    cout << endl;
}

int main() {
    vector<int> array = generateRandomArray(100);
    cout << "Here is the unsorted array: ";
    printArray(array);
    return 0;
}
```



```
This is the sorted array: 99 99 99 97 95 94 93 93 93 92 90 87 85 84 83 81 80 79 7
9 78 77 77 75 73 70 68 67 65 65 64 62 62 60 59 59 57 57 56 53 53 51 48 46 45 44 4
4 44 42 42 39 39 37 37 36 35 35 34 34 32 30 29 28 26 25 25 24 24 23 22 22 22 21 2
1 20 19 17 17 16 16 16 15 14 14 13 13 13 12 12 10 8 7 6 5 5 5 4 3 2 1 1


...Program finished with exit code 0
Press ENTER to exit console.
```

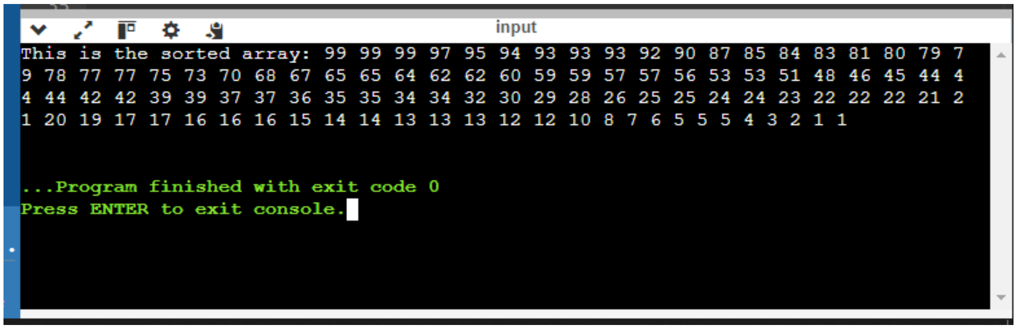
Observation	The shows a set of array ranging from 1 to 100

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

Code +Console Screenshot

C/C++

```cpp
#include <iostream>
#include <cstdlib> // For rand() and srand()
#include <ctime>   // For time()
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]) {
                std::swap(arr[j], arr[i]);
            }

        }

    }

}

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


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


    for (size_t i = 0; i < arrSize; i++) {
        arr[i] = rand() % 100 + 1; // Random values between 1
and 99
    }

    // Sort the array
    bubbleSort(arr, arrSize);
    cout << "This is the sorted array: ";
    for (size_t i = 0; i < arrSize; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```



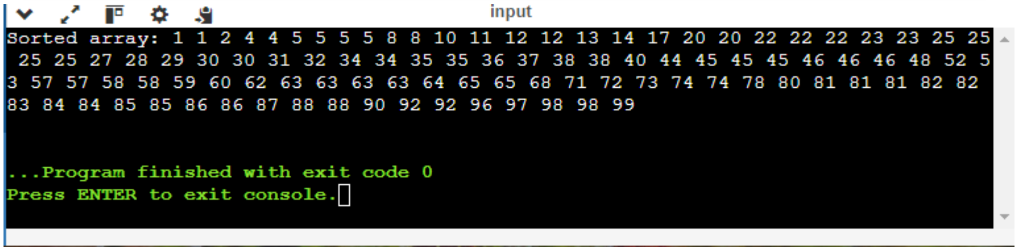
```
This is the sorted array: 99 99 99 97 95 94 93 93 93 92 90 87 85 84 83 81 80 79 7
9 78 77 77 75 73 70 68 67 65 65 64 62 62 60 59 59 57 57 56 53 53 51 48 46 45 44 4
4 44 42 42 39 39 37 37 36 35 35 34 34 32 30 29 28 26 25 25 24 24 23 22 22 22 21 2
1 20 19 17 17 16 16 16 15 14 14 13 13 13 12 12 10 8 7 6 5 5 5 4 3 2 1 1


...Program finished with exit code 0
Press ENTER to exit console.
```

Observation	the output shows that the displayed array is arranged from the biggest value to the smallest value

Table 7-2. Bubble Sort Technique

Code +Console Screenshot

```
C/C++
#include <iostream>
using namespace std;

// Selection Sort (Array, Size of Array)
void selection_sort(int arr[], int n) {
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n - 1; i++) {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i + 1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;

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

int main() {
    const size_t arrSize = 100;
    int arr[arrSize];
    srand(static_cast<unsigned>(time(0)));


    for (size_t i = 0; i < arrSize; i++) {
        arr[i] = rand() % 100 ; // Random values between 0 and
999
    }


    selection_sort(arr, arrSize);


    cout << "Sorted array: ";
    for (size_t i = 0; i < arrSize; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```



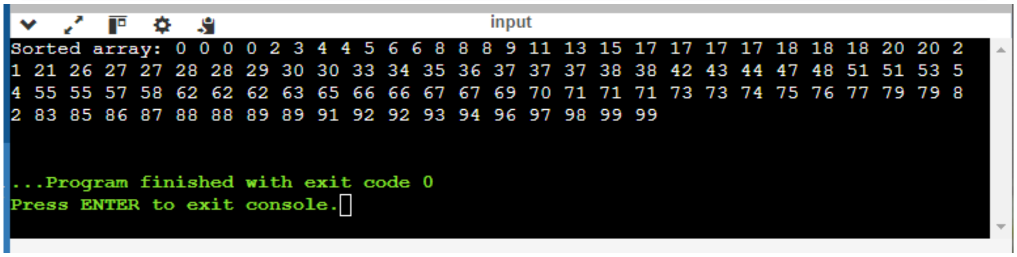
```
Sorted array: 1 1 2 4 4 5 5 5 5 8 8 10 11 12 12 13 14 17 20 20 22 22 22 23 23 25 25
 25 25 27 28 29 30 30 31 32 34 34 35 35 36 37 38 38 40 44 45 45 45 46 46 46 48 52 5
3 57 57 58 58 59 60 62 63 63 63 63 64 65 65 68 71 72 73 74 74 78 80 81 81 81 82 82
83 84 84 85 85 86 86 87 88 88 90 92 92 96 97 98 98 99


...Program finished with exit code 0
Press ENTER to exit console.
```

Observation	compared to the previous output, this output displayed a sorted array that arranges from the smallest value to the largest value

Table 7-3. Selection Sort Algorithm

Code +Console Screenshot

```
C/C++
#include <iostream>
using namespace std;

// Insertion Sort (Array, Size of Array)
template <typename T>
void insertionSort(T arr[], int n) {
    for (int i = 1; i < n; i++) {
        T key = arr[i];
        int j = i - 1;

        // Move elements of arr[0..i-1], that are greater than
key,
        // to one position ahead of their current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

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


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

    // Fill the array with random values
    for (size_t i = 0; i < arrSize; i++) {
        arr[i] = rand() % 100;
    }

    // Sort the array
    insertionSort(arr, arrSize);

    // Print the sorted array
    cout << "Sorted array: ";
    for (size_t i = 0; i < arrSize; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```



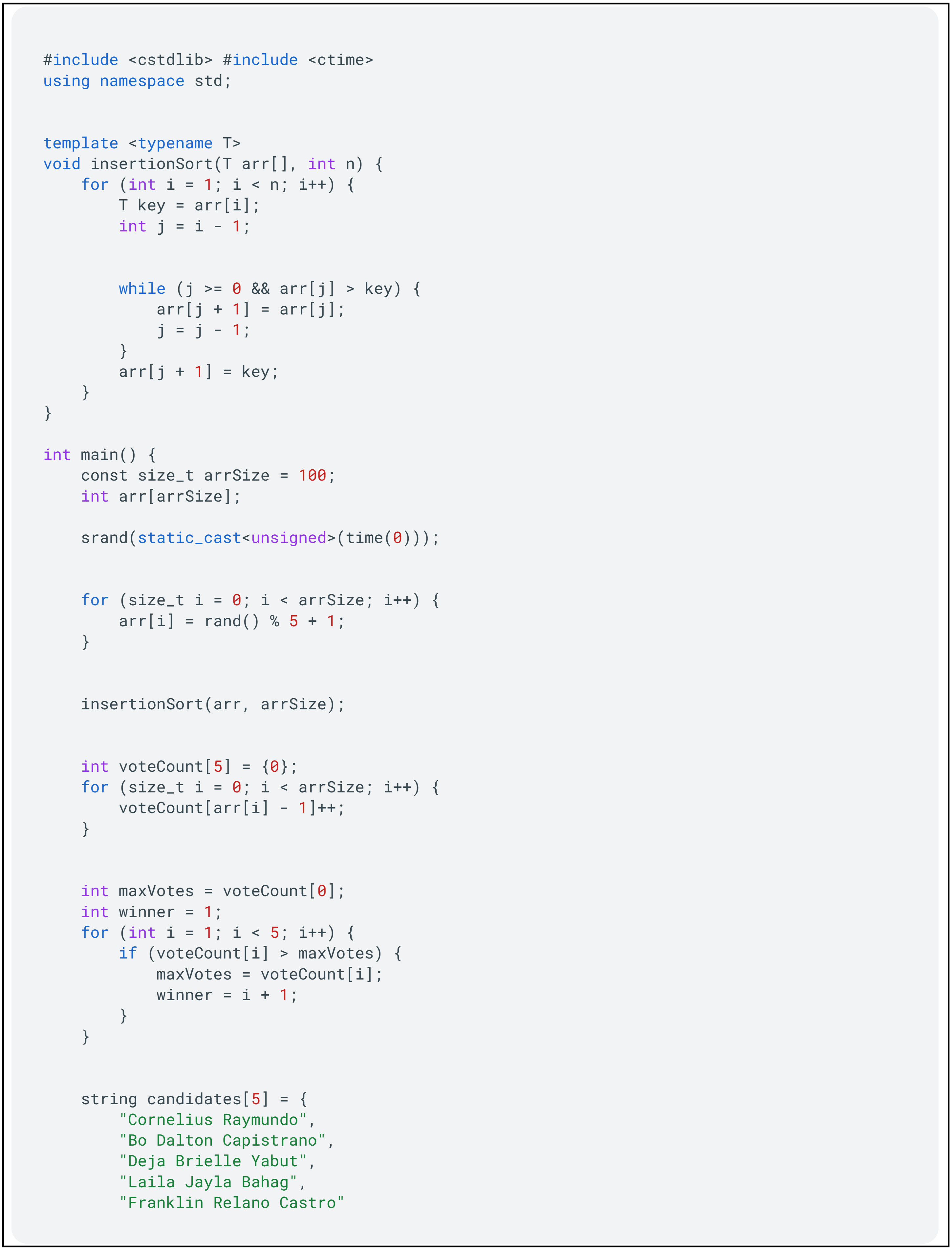
```
Sorted array: 0 0 0 0 2 3 4 4 5 6 6 8 8 8 9 11 13 15 17 17 17 17 18 18 18 20 20 2
1 21 26 27 27 28 28 29 30 30 33 34 35 36 37 37 37 38 38 42 43 44 47 48 51 51 53 5
4 55 55 57 58 62 62 62 63 65 66 66 67 67 69 70 71 71 71 73 73 74 75 76 77 79 79 8
2 83 85 86 87 88 88 89 89 91 92 92 93 94 96 97 98 99 99


...Program finished with exit code 0
Press ENTER to exit console.
```

Observation	Compared to the previous code, this code now includes and recognizes 0 and it is still arranged from smalles to largest

Table 7-4. Insertion Sort Algorithm

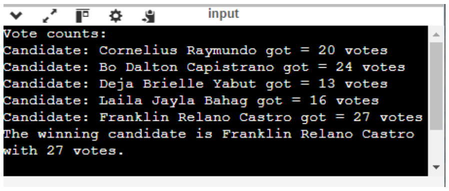
7. Supplementary Activity

```
C/C++
#include <iostream>
```



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


template <typename T>
void insertionSort(T arr[], int n) {
    for (int i = 1; i < n; i++) {
        T key = arr[i];
        int j = i - 1;


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

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

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


    for (size_t i = 0; i < arrSize; i++) {
        arr[i] = rand() % 5 + 1;
    }


    insertionSort(arr, arrSize);


    int voteCount[5] = {0};
    for (size_t i = 0; i < arrSize; i++) {
        voteCount[arr[i] - 1]++;
    }


    int maxVotes = voteCount[0];
    int winner = 1;
    for (int i = 1; i < 5; i++) {
        if (voteCount[i] > maxVotes) {
            maxVotes = voteCount[i];
            winner = i + 1;
        }
    }


    string candidates[5] = {
        "Cornelius Raymundo",
        "Bo Dalton Capistrano",
        "Deja Brielle Yabut",
        "Laila Jayla Bahag",
        "Franklin Relano Castro"
    };

    cout << "Vote counts:" << endl;
    for (int i = 0; i < 5; i++) {
        cout << "Candidate: " << candidates[i] << " got = " << voteCount[i] << " votes" <<
endl;
    }
    cout << "The winning candidate is " << candidates[winner - 1] << " with " << maxVotes
<< " votes." << endl;

    return 0;
}
```

Output Console showing sorted arra	manual count	count result of algotrithm


`Vote counts:`
`Candidate: Cornelius Raymundo got = 20 votes`
`Candidate: Bo Dalton Capistrano got = 24 votes`
`Candidate: Deja Brielle Yabut got = 13 votes`
`Candidate: Laila Jayla Bahag got = 16 votes`
`Candidate: Franklin Relano Castro got = 27 votes`
`The winning candidate is Franklin Relano Castro`
`with 27 votes.`
	11111111111111111111	20

	111111111111111111111111	24
	1111111111111	13
	1111111111111111	16
	111111111111111111111111111	27

The developed code-counting algorithm is effective because it shows the necessary information needed such as the identification of the candidate and the number of votes each candidate got

8. Conclusion

Sorting plays an important role in the world of algorithms. Efficiency sorting is powerful as it helps users to have an easier and less tedious approach when comparing data information, faster lookup, and more. In this activity we applied sorting in making a list of votes for each candidate, that activity proved how sorting plays a helpful role in identifying the winner accurately. Sorting not only organizes data, but it also simplifies other tasks such as searching, counting, and analyzing data information

9. Assessment Rubric