Activity No. 7	
SORTING ALGORITHMS: BUBBLE, SELECTION, AND INSERTION SORT	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: OCTOBER 16, 2024**
Section: CPE21S1	**Date Submitted: OCTOBER 16, 2024**
Name(s): GASPAR, AARON ROWEN O.	**Instructor: MS. MARIA RIZETTE SAYO**

6. Output

main.cpp

Share Run

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

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

    srand(time(0));

    for (int i = 0; i < size; ++i) {
        arr[i] = rand() % 100;
    }

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

    return 0;
}

```

```
/tmp/p2RZvnnvIL.o
53 90 9 17 23 0 86 29 54 70 68 9 5 39 61 68 26 52 92 13 72 63 42 11 54 11 71 4 0 72 10 5 15 71 22 38 24 8 67 78
    30 87 39 88 78 1 56 57 53 0 22 77 63 64 88 17 27 11 74 27 84 36 84 99 8 6 89 32 66 8 62 96 48 53 84 26 6 92
    83 11 44 57 88 7 21 29 77 0 40 3 79 76 39 63 27 99 69 16 83 87
```

Observation:
I use rand() for this code to randomize the selection of 100 numbers. I observed that every run of this code the sets of numbers are not the same from the previous one. We will notice a variety of results in the following code.

main.cpp

Share Run

```cpp
#include <iostream>
 using namespace std;
template <typename T>
void bubbleSort(T arr[], size_t arrSize){
//Step 1: For i = 0 to N-1 repeat Step 2
for(int i = 0; i < arrSize; i++){
//Step 2: For J = i + 1 to N – I repeat
for(int j = i+1; j < arrSize; j++){
//Step 3: if A[J] > A[i]
if(arr[j]>arr[i]){
//Swap A[J] and A[i]
std::swap(arr[j], arr[i]);
}
//[End of Inner for loop]
}
//[End if Outer for loop]
}
//Step 4: Exit
}

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

    srand(time(0));

    for (int i = 0; i < size; ++i) {
        arr[i] = rand() % 100;
    }

    bubbleSort(arr, size);

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

    return 0;
}
```

Output Clear

```
/tmp/ig9puTEZo6.o
98 97 95 94 92 91 91 91 91 90 89 88 87 87 87 87 86 86 86 82 79 77 77 77 75 75 74 71 70 68 65 65 65 65 64 63 63
    60 59 59 58 58 57 56 56 54 50 50 50 48 47 45 44 43 42 40 39 37 36 36 34 33 32 31 29 29 28 27 26 23 21 21 21
    20 20 20 20 17 16 16 16 13 12 11 11 11 11 10 10 10 9 8 8 8 8 7 4 3 1 1

=== Code Execution Successful ===
```

Observation:
I observed that when the function bubbleSort() is called, the output of the array was arrange from the biggest number up to the smaller ones.

main.cpp

Share Run

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

template <typename T>
int Routine_Smallest(T A[], int K, const int arrSize){
int position, j;
//Step 1: [initialize] set smallestElem = A[K]
T smallestElem = A[K];
//Step 2: [initialize] set POS = K
position = K;
//Step 3: for J = K+1 to N -1,repeat
for(int J=K+1; J < arrSize; J++){
if(A[J] < smallestElem){
smallestElem = A[J];
position = J;
}
}
//Step 4: return POS
return position;
}

template <typename T>
void selectionSort(T arr[], const int N){
int POS, temp, pass=0;
//Step 1: Repeat Steps 2 and 3 for K = 1 to N-1
for(int i = 0; i < N; i++){
//Step 2: Call routine smallest(A, K, N,POS)
POS = Routine_Smallest(arr, i, N);
temp = arr[i];
//Step 3: Swap A[K] with A [POS]
arr[i] = arr[POS];
arr[POS] = temp;
//Count
pass++;
}
//[End of loop]
//Step 4: EXIT
}
```

```
39
40  int main() {
41      const int size = 100;
42      int arr[size];
43
44      srand(time(0));
45
46      for (int i = 0; i < size; ++i) {
47          arr[i] = rand() % 100;
48      }
49
50      selectionSort(arr, size);
51
52      for (int i = 0; i < size; ++i) {
53          cout << arr[i] << " ";
54      }
55
56      return 0;
57  }
```

Output Clear

```
/tmp/YOdmAdXIRU.o
0 2 2 3 3 4 6 6 7 9 10 10 11 12 15 16 19 19 19 21 22 22 23 24 25 25 26 26 27 29 31 31 32
    34 35 36 36 37 38 39 41 41 42 42 43 45 47 48 49 49 51 52 53 53 55 55 57 58 59 61 62
    63 64 64 65 65 66 66 67 67 67 68 71 74 74 75 75 76 77 77 79 80 82 82 83 83 83 83 84
    84 86 86 87 90 91 91 92 94 98 99

=== Code Execution Successful ===
```

Observation:
For this one, as for the conditions, the output of the array is arranged from smallest to biggest value of the number.

main.cpp

Share Run

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

template <typename T>
void insertionSort(T arr[], const int N) {
int K, J;
T temp;
// Step 1: Repeat Steps 2 to 5 for K = 1 to N-1
for (K = 1; K < N; K++) {
// Step 2: set temp = A[K]
temp = arr[K];
// Step 3: set J = K - 1
J = K - 1;
// Step 4: Repeat while temp <= A[J] and J >= 0
while (J >= 0 && temp < arr[J]) {
// set A[J + 1] = A[J]
arr[J + 1] = arr[J];
// set J = J - 1
J--;
}
// Step 5: set A[J + 1] = temp
 arr[J + 1] = temp;
}
// Step 6: exit
}
```

```cpp
int main() {
    const int size = 100;
    int arr[size];

    // Seed the random number generator
    srand(time(0));

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

    // Sort the array using insertion sort
    insertionSort(arr, size);

    // Output the sorted array
    for (int i = 0; i < size; ++i) {
        cout << arr[i] << " ";
    }

    return 0;
}
```

Output Clear

```
/tmp/SHchgDYiM9.o
0 0 0 1 3 4 5 7 7 8 8 8 9 9 9 11 12 13 15 15 16 16 16 16 18 18 18 18 19 22 24 27 28 28
   29 30 32 33 33 37 38 38 38 41 43 43 43 46 47 47 47 50 53 53 54 54 55 55 59 61 62 62
   64 65 65 66 67 68 69 69 69 69 70 71 74 75 77 77 78 78 79 79 80 82 83 83 84 84 85 86
   87 88 88 94 94 95 96 98 99 99

=== Code Execution Successful ===
```

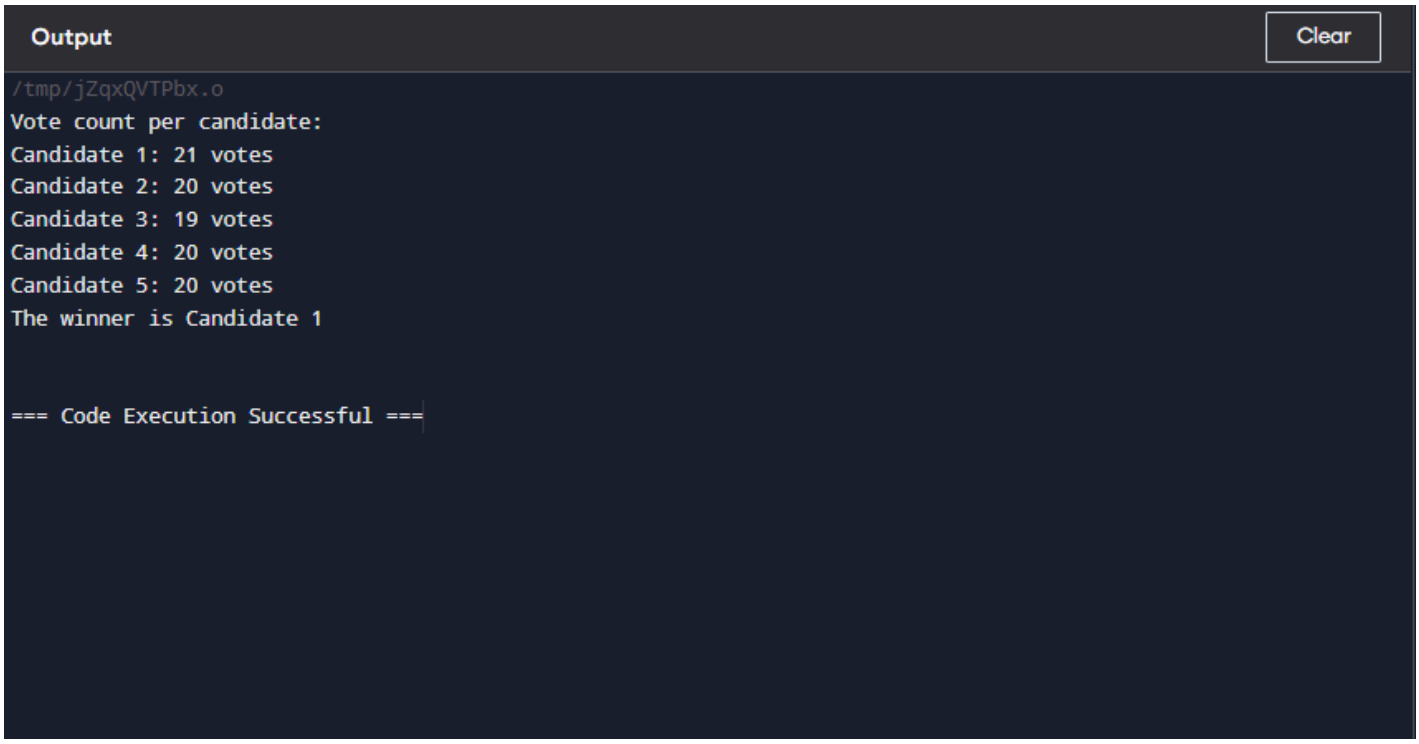
Observation:
The function outputs the array with a pattern based on your condition in the insertionSort() function.

7. Supplementary Activity

main.cpp

Share Run

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

void votesCounter(const int arr[], int size, int votes[]) {
    for (int i = 0; i < size; ++i) {
        votes[arr[i] - 1]++;
    }
}

int Winner(const int votes[], int numCandidates) {
    int maxVotes = votes[0];
    int winner = 0;
    for (int i = 1; i < numCandidates; ++i) {
        if (votes[i] > maxVotes) {
            maxVotes = votes[i];
            winner = i;
        }
    }
    return winner;
}

int main () {
    const int size = 100;
    int arr[size] = {4, 2, 3, 1, 5, 3, 2, 4, 1, 5, 2, 1, 4, 3, 5, 2, 4, 3, 1, 5, 4, 3, 2, 1, 5, 3, 4, 2
        , 1, 5, 1, 5, 2, 4, 3, 1, 4, 2, 5, 3, 1, 2, 5, 3, 4, 1, 5, 2, 3, 4, 2, 1, 5, 4, 2, 3, 1, 5, 4,
        3, 1, 5, 2, 4, 3, 1, 5, 2, 4, 3, 2, 1, 5, 3, 4, 2, 1, 5, 4, 2, 1, 5, 3, 4, 1, 2, 5, 3, 4, 1, 1,
        2, 3, 4, 5, 5, 4, 3, 2, 1};
    const int numCandidates = 5;
    int vote[numCandidates] = {0};


    votesCounter(arr, size, vote);

    int winner = Winner(vote, numCandidates);

    cout << "Vote count per candidate:" << endl;
    cout << "Candidate 1: " << vote[0] << " votes" << endl;
    cout << "Candidate 2: " << vote[1] << " votes" << endl;
    cout << "Candidate 3: " << vote[2] << " votes" << endl;
    cout << "Candidate 4: " << vote[3] << " votes" << endl;
    cout << "Candidate 5: " << vote[4] << " votes" << endl;
    cout << "The winner is Candidate " << winner + 1 << endl;

    return 0;
}
```



Pseudocode:

START
Initialize array arr[100] with given values
Initialize array votes[5] to {0, 0, 0, 0, 0}

Function votesCounter(arr, size, votes):
  For each element in arr:
    Increment the corresponding index in votes

Function Winner(votes, numCandidates):
  Set maxVotes to votes[0]
  Set winner to 0
  For each candidate from 1 to numCandidates - 1:
    If current candidate's votes > maxVotes:
      Set maxVotes to current candidate's votes
      Set winner to current candidate's index
  Return winner

Main:
  Initialize arr with 100 elements
  Initialize votes with 5 elements set to 0
  Call votesCounter(arr, size, votes)
  Set winner to the result of Winner(votes, numCandidates)
  Print votes for each candidate
  Print winner
STOP

8. Conclusion

Getting a handle on sorting algorithms like Bubble Sort, Selection Sort, and Insertion Sort in C++ opens up a great understanding of data manipulation fundamentals. Bubble Sort is simple to understand but not efficient for large datasets due to its quadratic time complexity. It's a good starting point for beginners. Selection Sort also has a quadratic time complexity, but does fewer swaps compared to Bubble Sort. It's useful when the cost of swapping is high. Insertion Sort is efficient for small datasets or nearly sorted

arrays with a linear time complexity in the best case, making it preferable in situations where the data is partially sorted. Each method has its strengths and weaknesses, which become more apparent as you delve into practical implementations. Learning these algorithms hones your problem-solving skills and deepens your grasp of basic programming concepts in C++, giving you tools to optimize and enhance performance in more complex tasks.

9. Assessment Rubric