Hands-on Activity 7.1	
Sorting Algorithms Pt1	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 09/18/2025
Section: CPE21S4	**Date Submitted:** 09/18/2025
Name(s): Cabrera, Gabriel A.	**Instructor:** Engr. Jimlord Quejado

6. Output

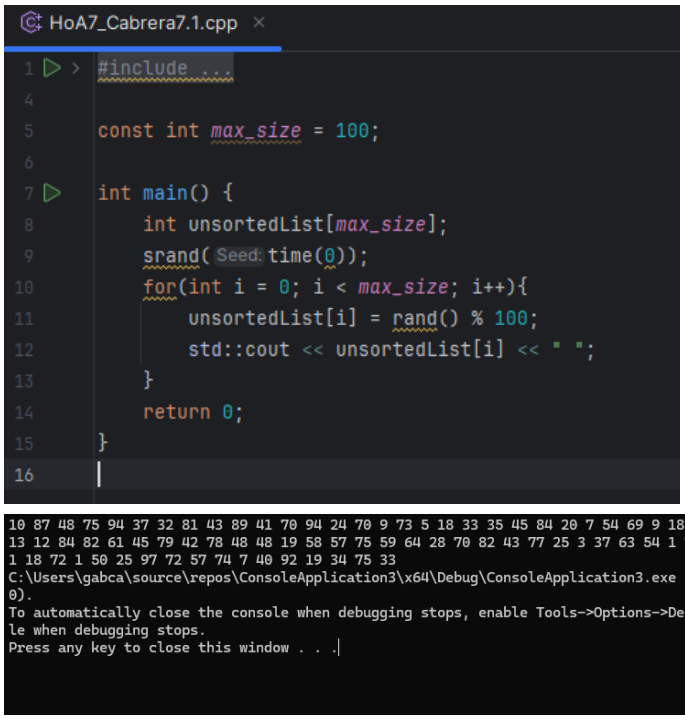
Code + Console Screenshot	
Observation	Standard listing of random numbers using the rand function and limiting the numbers generated to a maximum of 99 by utilizing the %100 operation.

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

Code + Console Screenshot

HoA7_Cabrera7.2.cpp

```cpp
#include <iostream>
#include <cstdlib>
#include <time.h>
#include "bubbleSort.h"

const int max_size = 100;

void randomArr(int arr[]);
void displayArr(int arr[]);

int main() {
    int unsortedList[max_size];
    srand( Seed: time(0));
    for(int i = 0; i < max_size; i++){
        unsortedList[i] = rand() % 100;
        std::cout << unsortedList[i] << " ";
    }

    randomArr(unsortedList);
    std::cout << "\n\nBUBBLE SORT" << std::endl;
    bubbleSort(unsortedList, max_size);
    displayArr(unsortedList);

    return 0;
}

void randomArr(int arr[]){
    int unsortedList[max_size];
    srand( Seed: time(0));
    for(int i = 0; i < max_size; i++){
        unsortedList[i] = rand() % 100;
    }
}

void displayArr(int arr[]){
    for(int i = 0; i < max_size; i++){
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}
```

HoA7_Cabrera7.2.cpp bubbleSort.h

```
#ifndef BUBBLESORT_H
#define BUBBLESORT_H
template <typename T>
void bubbleSort(T arr[], const int arrSize){
    T temp;
    for(int i = 0; i < arrSize; i++){
        for(int j = i+1; j < arrSize; j++){
            if(arr[j] < arr[i]){
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}
#endif #ifndef BUBBLESORT_H
```

#ifndef BUBBLESORT_H

Microsoft Visual Studio Debu

```
31 95 77 54 89 42 41 25 26 72 60 76 98 98 35 91 94 91 79 36 25 48 33 39 64 70 42 31 66 6 54 47 90 92 19 8 19 43 72 89 11
 68 64 90 55 29 65 86 51 64 64 95 59 35 22 23 73 78 92 80 20 43 83 95 32 6 80 58 94 34 6 51 27 70 91 32 88 87 52 97 72 3
7 27 7 86 0 59 30 80 49 83 41 49 5 54 68 26 93 87 82

BUBBLE SORT
0 5 6 6 6 7 8 11 19 19 20 22 23 25 25 26 26 27 27 29 30 31 31 32 32 33 34 35 35 36 37 39 41 41 42 42 43 43 47 48 49 49 5
1 51 52 54 54 54 55 58 59 59 60 64 64 64 64 65 66 68 68 70 70 72 72 72 73 76 77 78 79 80 80 80 82 83 83 86 86 87 87 88 8
9 89 90 90 91 91 91 92 92 93 94 94 95 95 95 97 98 98
```

Observation	After initiating the random numbers from the first code the bubble sort function is called. The bubble sorting code uses two nested for loops to accomplish this: the outer loop iterates through the array, while the inner loop compares each element to the one following it. If an element is found to be smaller than the one it is being compared to, the two are swapped using a temporary variable. The template ensures this function can be used to sort arrays of various data types, which is integer for this example.

Table 7-2. Bubble Sort Technique

Code + Console Screenshot

HoA7_Cabrera7.3.cpp | selectionSort.h

```cpp
#include <iostream>
#include <cstdlib>
#include <time.h>
#include "selectionSort.h"

const int max_size = 100;

void randomArr(int arr[]);
void displayArr(int arr[]);

int main() {
    int unsortedList[max_size];
    srand( Seed: time(0));
    for(int i = 0; i < max_size; i++){
        unsortedList[i] = rand() % 100;
        std::cout << unsortedList[i] << " ";
    }

    randomArr(unsortedList);
    std::cout << "\nSELECTION SORT\n" << std::endl;
    selectionSort(unsortedList, max_size);
    displayArr(unsortedList);
    return 0;
}

void randomArr(int arr[]){
    int unsortedList[max_size];
    srand( Seed: time(0));
    for(int i = 0; i < max_size; i++){
        unsortedList[i] = rand() % 100;
    }
}

void displayArr(int arr[]){
    for(int i = 0; i < max_size; i++){
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}

```

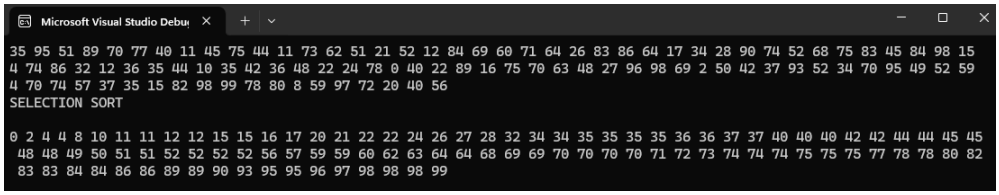
HoA7_Cabrera7.3.cpp | selectionSort.h

```cpp
#ifndef SELECTIONSORT_H
#define SELECTIONSORT_H
template <typename T> int routineSmallest(T subArr[], int K, const int arrSize);

template <typename T>
void selectionSort(T arr[], const int arrSize){
    int POS, pass = 0;
    T temp;
    for(int i = 0; i < arrSize; i++){
        POS = routineSmallest(arr, i, arrSize);
        temp = arr[i];
        arr[i] = arr[POS];
        arr[POS] = temp;

        pass++;
    }
}

template <typename T>
int routineSmallest(T subArr[], int K, const int arrSize){
    int position = K; T smallestElement = subArr[K];
    for(int J = K+1; J < arrSize; J++){
        if(subArr[J] < smallestElement){
            smallestElement = subArr[J];
            position = J;
        }
    }
    return position;
}
#endif
```



Observation	Similar to the previous programs, this one initializes an unsorted array first before calling the selectionSort function. The selectionSort function's main loop iterates through the array from i=0 to arrSize. In each iteration, it calls routineSmallest to find the index of the smallest element in the remaining unsorted part of the array. Once the smallest element is found, it's swapped with the element at the current position i. The pass++ statement is a counter for how many passes the algorithm has made.

Table 7-3. Selection Sort Algorithm

Code + Console Screenshot

HoA7_Cabrera7.4.cpp

```cpp
#include <iostream>
#include <cstdlib>
#include <time.h>
#include "insertionSort.h"

const int max_size = 100;

void randomArr(int arr[]);
void displayArr(int arr[]);

int main() {
    int unsortedList[max_size];
    srand( Seed: time(0));
    for(int i = 0; i < max_size; i++){
        unsortedList[i] = rand() % 100;
        std::cout << unsortedList[i] << " ";
    }

    randomArr(unsortedList);
    std::cout << "\nINSERTION SORT\n" << std::endl;
    insertionSort(unsortedList, max_size);
    displayArr(unsortedList);

    return 0;
}

void randomArr(int arr[]){
    int unsortedList[max_size];
    srand( Seed: time(0));
    for(int i = 0; i < max_size; i++){
        unsortedList[i] = rand() % 100;
    }
}

void displayArr(int arr[]){
    for(int i = 0; i < max_size; i++){
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}
```

HoA7_Cabrera7.4.cpp insertionSort.h

```cpp
#ifndef INSERTIONSORT_H
#define INSERTIONSORT_H
template <typename T>
void insertionSort(T arr[], const int arrSize){
    int K = 1, J;
    T temp;
    while(K < arrSize){
        temp = arr[K];
        J = K-1;
        while(temp <= arr[J]){
            arr[J+1] = arr[J];
            J--;
            if(J < 0) break;
        }
        arr[J+1] = temp;
        K++;
    }
}
#endif #ifndef INSERTIONSORT_H
```

Microsoft Visual Studio Debu

```
29 38 2 69 16 57 84 38 37 41 85 12 77 3 55 98 50 51 81 32 80 8 59 14 2 36 2 70 62 76 81 6 99 82 52 42 64 15 64 21 44 44
67 6 92 64 97 61 28 76 43 35 13 97 3 78 82 4 97 1 86 52 64 43 98 81 87 38 48 33 81 53 63 47 40 58 27 17 6 58 11 18 85 70
 97 89 24 3 86 33 13 20 47 73 38 38 43 34 35 32
INSERTION SORT

1 2 2 2 3 3 3 4 6 6 6 8 11 12 13 13 14 15 16 17 18 20 21 24 27 28 29 32 32 33 33 34 35 35 36 37 38 38 38 38 40 41 42
43 43 43 44 44 47 47 48 50 51 52 52 53 55 57 58 58 59 61 62 63 64 64 64 64 67 69 70 70 73 76 76 77 78 80 81 81 81 81 82
82 84 85 85 86 86 87 89 92 97 97 97 97 98 98 99
```

Observation	Like the previous programs, this one initializes a random array first then calls the insertionSort function. The code uses a main while loop to iterate through the array, taking one element at a time and inserting it into the correct position within the already sorted part of the array. Inside the loop, a temporary variable is used to hold the current element being sorted. A second nested while loop is used to shift all the elements in the sorted portion that are greater than the current element to the right, making room for the insertion. This process continues until the entire array is sorted.

Table 7-4. Insertion Sort Algorithm

7. Supplementary Activity

HoA7_CabreraSupplementary.cpp × sortingAlgo.h

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

#include "sortingAlgo.h"

int main() {
    std::vector<int> votes( Count: 101);

    std::cout << "Votes casted:\n";
    srand(static_cast<unsigned int>(time(0)));

    for (int i = 0; i < 101; ++i) {
        votes[i] = rand() % 5 + 1;
        std::cout << votes[i] << " ";
    }
    std::cout << std::endl << std::endl;

    insertionSort( [&] votes);

    std::cout << "Votes sorted:\n";
    for (int i = 0; i < 101; ++i) {
        std::cout << votes[i] << " ";
    }
    std::cout << std::endl << std::endl;

    // Count the votes

    std::vector<int> voteCounts( Count: 5,  Val: 0);
    for (int vote : votes) {
        if (vote >= 1 && vote <= 5) {
            voteCounts[vote - 1]++;
        }
    }


    int maxVotes = -1;
    int winningCandidateIndex = -1;
    for (int i = 0; i < voteCounts.size(); ++i) {
```

main

```cpp
    int maxVotes = -1;
    int winningCandidateIndex = -1;
    for (int i = 0; i < voteCounts.size(); ++i) {
        if (voteCounts[i] > maxVotes) {
            maxVotes = voteCounts[i];
            winningCandidateIndex = i;
        }
    }

    std::cout << "Vote Results:" << std::endl;
    for (int i = 0; i < candidates.size(); ++i) {
        std::cout << "Candidate " << (i + 1) << " (" << candidates[i] << "): " << voteCounts[i] << " votes" << std::endl;
    }

    std::cout << "\nWinner:" << std::endl;
    if (winningCandidateIndex != -1) {
        std::cout << "Candidate " << (winningCandidateIndex + 1) << " (" << candidates[winningCandidateIndex] << ") with " << maxVotes << " votes." << std::endl;
    }
    else {
        std::cout << "No votes were counted." << std::endl;
    }

    return 0;
}
```

HoA7_CabreraSupplementary.cpp sortingAlgo.h

```cpp
#ifndef SORTINGALGO_H
#define SORTINGALGO_H
#include ...

void insertionSort(std::vector<int>& arr) {
    int n = arr.size();
    for (int i = 1; i < n; ++i) {
        int 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;
    }
}

const std::vector<std::string> candidates = {
    "Bo Dalton Capistrano",
    "Cornelius Raymon Agustin",
    "Deja Jayla Bañaga",
    "Lalla Brielle Yabut",
    "Franklin Relano Castro"
};

#endif #ifndef SORTINGALGO_H
```

Microsoft Visual Studio Debu

```
Votes casted:
1 1 4 1 2 5 2 4 3 3 4 2 1 2 5 5 2 5 4 3 2 3 5 1 1 1 1 2 5 2 5 5 5 1 2 1 2 3 1 4 4 2 3 5 3 2 3 4 3 4 5 2 3 2 2 5 1 4 4 4
3 5 3 4 1 2 5 4 2 5 5 5 5 2 3 3 3 4 3 2 4 2 3 4 3 1 4 4 2 3 1 3 3 2 4 3 4 4 1 4 3

Votes sorted:
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5

Vote Results:
Candidate 1 (Bo Dalton Capistrano): 16 votes
Candidate 2 (Cornelius Raymon Agustin): 22 votes
Candidate 3 (Deja Jayla Ba±aga): 23 votes
Candidate 4 (Lalla Brielle Yabut): 22 votes
Candidate 5 (Franklin Relano Castro): 18 votes

Winner:
Candidate 3 (Deja Jayla Ba±aga) with 23 votes.
```

Manual Counting:
Candidate 1 = 16 votes
Candidate 2 = 22 votes
Candidate 3 = 23 votes
Candidate 4 = 22 votes
Candidate 5 = 18 votes
Winner: Candidate 3

Question: Was your developed vote counting algorithm effective? Why or why not?
I used insertion sort since the vote counts were only up to 5, making it ideal because of the very small range and data set. Also, after generating numbers, the array will be nearly sorted since it's only 1 to 5. This is a simple implementation and is also memory efficient

8. Conclusion

This activity demonstrated the use of bubble sorting, selection sorting, and insertion sorting, and how they work and how they get implemented. Seeing how each was coded also made me understand how they work internally. I learned that each method has its own strengths and weaknesses, which is why the learning outcome of choosing the right algorithm for a given problem is so important. For instance, in the vote counting activity, knowing how to efficiently sort a large number of votes is critical for accurately determining a winner.

9. Assessment Rubric