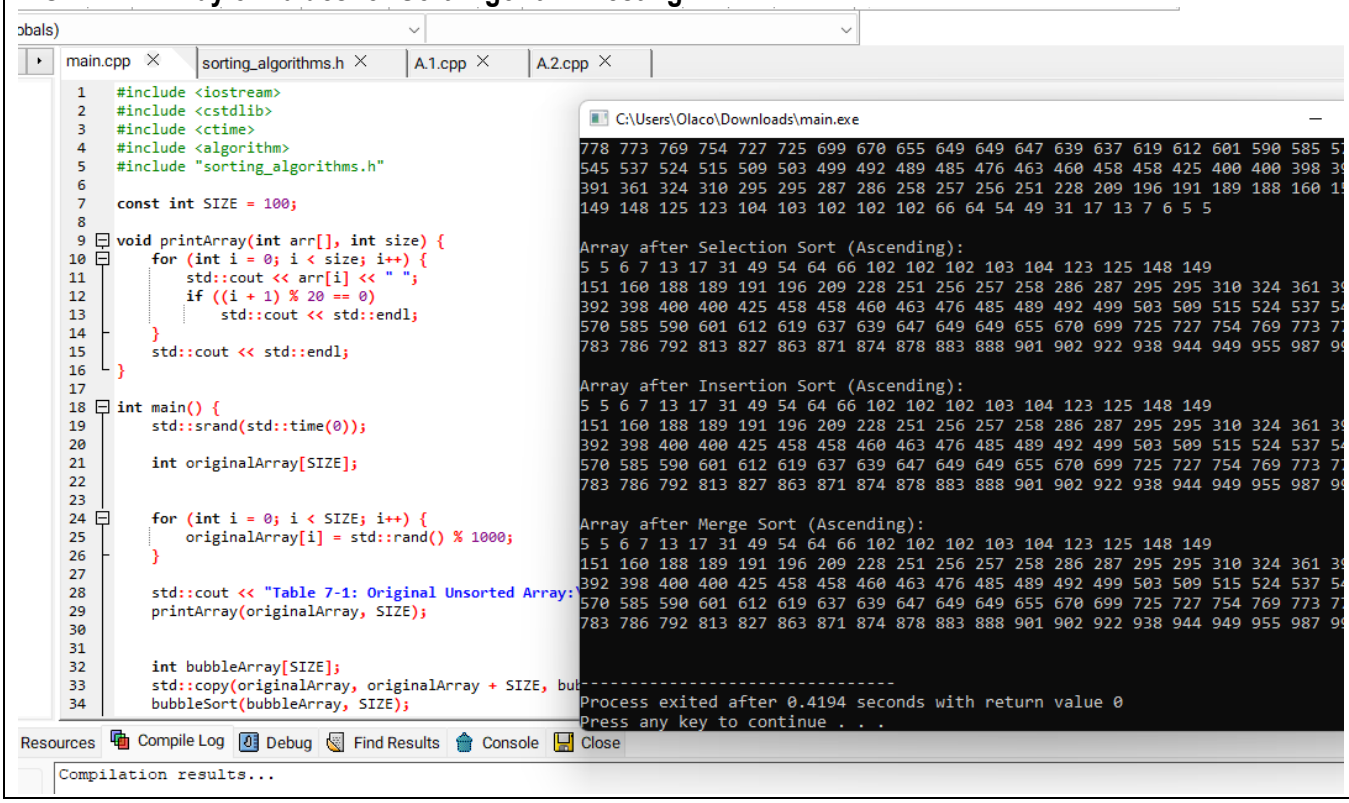
Hands-on Activity 7.1	
Sorting Algorithms Pt1	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 09/18/25
Section: CPE21S4	**Date Submitted:** 09/18/25
Name(s): Kerwin Jan B. Catungal	**Instructor:** Engr. Jimlord Quejado

6. Output

ILO A: 7-1. Array of Values for Sort Algorithm Testing



```cpp
#ifndef SORTING_ALGORITHMS_H
#define SORTING_ALGORITHMS_H

#include <algorithm>


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


template <typename T>
int findSmallestPosition(T arr[], int start, int size) {
    int pos = start;
    for (int i = start + 1; i < size; i++) {
        if (arr[i] < arr[pos]) {
            pos = i;
        }
    }
    return pos;
}


template <typename T>
void selectionSort(T arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        int smallestPos = findSmallestPosition(arr, i, size);
```

Resources | Compile Log | Debug | Find Results | Console | Close

Header:

Here as tasked I created a random number generator program that creates a hundred random numbers. After that I copied the numbers into different sets and used bubble, selection, insertion, and merge sort to analyze how each one arranges the same data. I also created a header file that would allow me to directly call all the sorting codes and thus, I could easily observe the differences in the algorithms.

A.1. Bubble Sort Technique:

main.cpp | sorting_algorithms.h | A.1.cpp | A.2.cpp

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <algorithm>

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]) {
                std::swap(arr[j], arr[j + 1]);
            }
        }
    }
}


template <typename T>
void printArray(T arr[], size_t arrSize) {
    for (size_t i = 0; i < arrSize; i++) {
        std::cout << arr[i] << " ";
        if ((i + 1) % 10 == 0) std::cout << std::endl;
    }
    std::cout << std::endl;
}

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


    std::srand(static_cast<unsigned int>(std::time(nullptr)));
```

C:\Users\Olaco\Downloads\A.1.exe

```
Array before Bubble Sort:
52 13 7 65 14 9 8 17 8 63
62 34 89 50 81 30 78 85 78 48
35 57 89 53 36 7 45 94 69 86
69 79 9 85 56 31 47 12 35 44
10 23 26 62 71 28 68 75 60 87
54 86 99 26 71 12 84 6 33 30
51 78 35 84 85 50 2 97 69 81
12 57 38 27 51 19 47 26 58 31
89 13 95 63 64 6 93 20 97 40
70 7 59 44 76 32 73 89 3 15

Array after Bubble Sort (ascending order):
2 3 6 6 7 7 7 8 8 9
9 10 12 12 12 13 13 14 15 17
19 20 23 26 26 26 27 28 30 30
31 31 32 33 34 35 35 35 36 38
40 44 44 45 47 47 48 50 50 51
51 52 53 54 56 57 57 58 59 60
62 62 63 63 64 65 68 69 69 69
70 71 71 73 75 76 78 78 78 79
81 81 84 84 85 85 85 86 86 87
89 89 89 89 93 94 95 97 97 99


--------------------------------
Process exited after 0.3445 seconds with return value 0
Press any key to continue . . .
```

sources | Compile Log | Debug | Find Results | Console | Close

```
Compilation results...
--------
```

```cpp
#ifndef SORTING_H
#define SORTING_H

#include <iostream>
#include <algorithm>


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]) {
                std::swap(arr[j], arr[j + 1]);
            }
        }
    }
}


template <typename T>
void printArray(T arr[], size_t arrSize) {
    for (size_t i = 0; i < arrSize; i++) {
        std::cout << arr[i] << " ";
        if ((i + 1) % 10 == 0) std::cout << std::endl;
    }
    std::cout << std::endl;
}

#endif
```

I added a dedicated header file with the bubble sort and printArray functions. So here I generated 100 random numbers, and I initially displayed them in an unsorted manner, after which I invoked bubble sort to sort them in ascending order and displayed the output.

A.2. Selection Sort Algorithm:

main.cpp | sorting_algorithms.h | A.1.cpp | A.2.cpp | A.3.cpp | SUPILOB.cpp

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

int main() {
    int arr[5] = {20, 10, 50, 30, 40};
    int n = 5;

    for(int i = 0; i < n - 1; i++) {
        int minIndex = i;

        for(int j = i + 1; j < n; j++) {
            if(arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        swap(arr[i], arr[minIndex]);
    }

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

```
C:\Users\Olaco\Downloads\A.2.exe
10 20 30 40 50
--------------------------------
Process exited after 0.3836 seconds with return value 0
Press any key to continue . . .
```

```cpp
#ifndef SELECTION_SORT_H
#define SELECTION_SORT_H

#include <algorithm>

template <typename T>
void selectionSort(T arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        std::swap(arr[i], arr[minIndex]);
    }
}

#endif
```

So the data points here were in a disorganized manner and needed order to be set in a structured form. So I have used selection sort. Steps to carry out this action are trying to find the lowest number in the group, then, moving that number to the front. I also moving the selection sort function into a header file to make my program more structured. I can also access this function from the main program with more ease.

A.3. Insertion Sort Algorithm:

main.cpp | sorting_algorithms.h | A.1.cpp | A.2.cpp | A.3.cpp | SUPILOB.cpp

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

int main() {
    int arr[5] = {20, 10, 50, 30, 40};
    int n = 5;

    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--;
        }
        arr[j + 1] = key;
    }

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

```
C:\Users\Olaco\Downloads\A.3.exe
10 20 30 40 50
--------------------------------
Process exited after 0.3767 seconds with return value 0
Press any key to continue . . .
```

```cpp
#ifndef INSERTION_SORT_H
#define INSERTION_SORT_H

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--;
        }
        arr[j + 1] = key;
    }
}

#endif
```

I have used the insertion sort method as tasked and for arranging the numbers for this program. The algorithm takes each element and puts the element in the correct index and kept the order in the array sorted. The insertion sort function was kept in a header file also so that my code is organized and I can simply invoke it in the main program.

7. Supplementary Activity

main.cpp | sorting_algorithms.h | A.1.cpp | A.2.cpp | A.3.cpp | SUPILOB.cpp

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

int main() {
    int A[100];
    int n = 100;
    int i, j, temp;

    srand(time(0));


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


    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (A[j] > A[j + 1]) {
                temp = A[j];
                A[j] = A[j + 1];
                A[j + 1] = temp;
            }
        }
    }


    int c1 = 0, c2 = 0, c3 = 0, c4 = 0, c5 = 0;
    for (i = 0; i < n; i++) {
        if (A[i] == 1) c1++;
        else if (A[i] == 2) c2++;
        else if (A[i] == 3) c3++;
```

C:\Users\Olaco\Downloads\SUPILOB.exe

```
Sorted Array:
1 1 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 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 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5

Manual Count:
Gab: 17
Dal: 19
Jason: 22
Gelo: 16
Lester: 26

Winner is Candidate 5 with 26 votes!

--------------------------------
Process exited after 0.3455 seconds with return value 0
Press any key to continue . . .
```

Resources | Compile Log | Debug | Find Results | Console | Close

```
Compilation results...
```

Output Console Showing Sorted Array	Manual Count	Count Result of Algorithm
Sorted Array: 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 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 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	Manual Count: Gab: 15 Dal: 18 Jason: 20 Gelo: 23 Lester: 24	Gab: 15 Dal: 18 Jason: 20 Gelo: 23 Lester: 24 Winner is Candidate 5 with 24 votes!

I generated 100 random votes with values from 1 to 5 and each one representing a candidate I choose to use the bubble sort to arrange the votes in ascending order so it's easier to count. Thereafter I created manual tallies for each candidate and finally the program compares the counts and declares the winner.

8. Conclusion

So after doing this activity I managed to at least learn and practice bubble sort, selection sort, and insertion sort.I learned how each of them works in processing and ordering numbers and also attempted to use header files in an effort to systematize my code. I incorporated this in a voting program in which the casts were organized and counted in order to establish a winner. I did receive some assistance while doing this because coding, to me, is still somewhat difficult. But I know this as a part of learning and I still need to work more in order to sharpen my skills in programming.

9. Assessment Rubric