Hands-on Activity 7.1	
Sorting Algorithms Pt1	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 18/9/25**
Section: CPE21S4	**Date Submitted: 18/9/25**
Name(s): Francis Nikko I. Andoy	**Instructor:** Jimlord Quejado

6. Output

BUBBLE SORT:

Sample Code:

MAIN:

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "searchingalgo.h"
const int size = 100;
int arr[size];

int main(){

    //BubbleSort
    std::srand(std::time(0));
    for(int i=0; i < size; i++){
        arr[i] = rand() % 200 + 1;
    }

    std::cout<<"Random Unsorted Array: " << std::endl;

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

    std::cout <<"End of the Unsorted Array" << std::endl;



    bubbleSort(arr, size);
    std::cout<<"Sorted Array using Bubble Sort: "<<std::endl;
    for(int i = 0; i < size; i++){
        std::cout<<arr[i] << "->";
    }
    std::cout << "End of the Sorted Array";
    //BubbleSort
```

HEADER:

```cpp
template<typename T>
void bubbleSort(T arr[], size_t arrSize){
    for(size_t i = 0; i < arrSize; i++){

        for(size_t j = i+1; j < arrSize; j++){

            if(arr[j] < arr[i]){
                std::swap(arr[j], arr[i]);
            }
        }
    }
}
```

Observation:

From what I've observed we used the header algorithm, it has a key to replace the data type, which is the size_t. We used it to define the arrSize parameter. We are used to for loops since we have to iterate after the first iteration, iterate till the array is sorted. This is inefficient because of how slow it is, iterating over and over again is the worst sorting. Unlike other sorting algorithms, which iterate just once. But it is fast when the array is already sorted and it is also efficient when the array is just small, but when it comes to a larger set of data, this algorithm is useless. The big O notation for this algorithm is, which is O(n^2)

As for the random generator for the array we used the header, `cstdlib.` Then we used the const to set it just to 100, it won't just no matter what do.

OUTPUT:

```
Random Unsorted Array:
34->142->144->21->24->20->190->26->49->167->45->143->69->86->112->147->9->48->145->68->133->186->78->7
25->129->57->69->19->70->140->46->82->148->132->167->121->35->105->156->182->146->35->9->101->21->23->
7->62->57->98->91->189->187->9->191->161->17->2->108->End of the Unsorted Array
Sorted Array using Bubble Sort:
2->4->8->8->8->9->9->9->10->10->10->11->16->17->17->19->20->21->21->23->23->24->25->26->29->30->31->31
>78->82->86->91->98->99->101->105->105->108->112->121->121->121->122->124->125->126->129->131->132->13
79->182->186->187->188->189->190->190->190->191->196->End of the Sorted Array
```

Table 7-1. Array of Values for Sort Algorithm Testing and Table 7-2. Bubble Sort Technique

---

SELECTION SORT

CODE:

MAIN:

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "searchingalgo.h"
const int size = 100;
int arr[size];

int main(){


    std::srand(std::time(0));
    for(int i=0; i < size; i++){
        arr[i] = rand() % 200 + 1;
    }

    std::cout<<"Random Unsorted Array: " << std::endl;

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

    std::cout <<"End of the Unsorted Array" << std::endl;

    //SelectioSort
    selectionSort(arr, size);
    std::cout<< "Sorted Array using Selecction Sort: " << std::endl;
    for(int i = 0; i<size; i++){
        std::cout<<arr[i] << "->";
    }
    std::cout << "End of the Sorted Array";


    return 0;
}
```

OBSERVATION:

I've observed that we first created a function prototype at the beginning of the header, then proceeded to make the actual selection sort. By using the template we can make our code generic. We then can select what type of data we will use like in the example code. Unlike bubble sort, where we used 2 for loop, here we just used 1 since this algorithm will just iterate all the elements just once. Additionally, the comparison of this algorithm is just small. This is more efficient than the bubble when it comes to a large data set like this array. Where the size is 100. The big O notation for this algorithm is just like bubble, which is O(n^2)

HEADER:

```h
template <typename T> int Routine_Smallest(T A[], int K, const int arrSize);
template <typename T>
int Routine_Smallest(T A[], int K, const int arrSize){
    int position, j;
    T smallestElem = A[K];
    position = K;
    for(int J=K+1; J < arrSize; J++){
        if(A[J] < smallestElem){
        smallestElem = A[J];
        position = J;
        }

    }
    return position;
}


template <typename T>
void selectionSort(T arr[], const int N){
    int POS, temp, pass=0;

    for(int i = 0; i < N; i++){
    POS = Routine_Smallest(arr, i, N);
    temp = arr[i];
    arr[i] = arr[POS];
    arr[POS] = temp;
    pass++;
    }

}
```

OUTPUT:

```
Random Unsorted Array:
177->28->33->106->118->61->126->75->54->173->133->152->157->9->85->45->106->26->187->170->47->89->189
9->110->120->143->127->52->15->8->54->93->153->110->141->31->80->60->16->36->143->199->116->20->118->
2->175->73->41->190->95->93->112->58->17->64->185->62->66->52->66->109->13->24->195->194->165->137->1
Sorted Array using Selecction Sort:
8->9->10->11->12->13->15->15->16->17->20->24->26->28->31->31->33->36->41->45->47->47->50->52->52->54-
80->85->85->89->91->93->93->93->95->98->100->106->106->107->109->110->110->112->112->116->118->118->1
53->157->157->165->168->169->170->173->175->175->176->177->181->182->185->187->189->190->193->194->19
```

Table 7-3. Selection Sort Algorithm

INSERTION SORT:

CODE:

MAIN:

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "searchingalgo.h"
const int size = 100;
int arr[size];

int main(){


    std::srand(std::time(0));
    for(int i=0; i < size; i++){
        arr[i] = rand() % 200 + 1;
    }

    std::cout<<"Random Unsorted Array: " << std::endl;

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

    std::cout <<"End of the Unsorted Array" << std::endl;

    //Insertaion Sort
    insertionSort(arr, size);
    std::cout<< "Sorted Array using Insertion Sort: " << std::endl;
    for(int i = 0; i<size; i++){
        std::cout<<arr[i] << "->";
    }
    std::cout << "End of the Sorted Array";


    return 0;
}
```

HEADER:

```cpp
template <typename T>
void insertionSort(T arr[], const int N){
int K = 0, J, temp;

    while(K < N){
    temp = arr[K];
    J = K-1;
        while(temp <= arr[J]){
            arr[J+1] = arr[J];
            J--;
        }
    arr[J+1] = temp;
    K++;
    }
}
```

OUTPUT:

```
Random Unsorted Array:
122->55->30->101->181->14->129->157->67->192->140->1->126->73->2->115->50->29->97->171->98->192->180-
->77->57->190->191->99->200->112->172->4->149->163->142->152->83->52->103->184->200->3->143->87->160-
->167->155->20->144->81->18->144->117->119->55->135->158->120->155->142->End of the Unsorted Array
Sorted Array using Insertion Sort:
1->2->3->4->7->12->14->18->19->20->21->22->24->28->29->30->31->37->43->44->46->50->52->55->55->57->67
17->119->120->120->122->123->126->129->129->131->132->135->139->139->140->141->142->142->143->144->14
5->179->180->181->182->183->184->190->191->192->192->195->198->200->200->End of the Sorted Array
```

OBSERVATION:

I've observed, this algorithm is just like selection. Because it will just iterate the array in one go. If the comparison falls to True then it will swap it. Sounds like bubble sort but it is not, the big difference is that the bubble sort will iterate the array 1 by 1, meaning 1 iterate 1 sorted value. That is how bubble sort works. But for this algorithm, it will swap and swap the values on their right space in just 1 iteration. The big notation of this is just like the other 2 where it is O(n^2)

Table 7-4. Insertion Sort Algorithm

7. Supplementary Activity

CODE:
MAIN:

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "supp_sorting.h"

const int size = 100;
int arr[size];

void distance(){
    std::cout<<
"SORTING ALGORITHM-------------------------------------------------------
---------SORTING ALGORITHM
\n";
}

int main() {
    distance();
    std::cout<< "\n";
    randomNum(arr, size);
    std::cout << "\n\n";

    distance();
    std::cout << "\n\n";

    std::cout << "Bubble sort\n";
    bubbleSort(arr, size);
    std::cout << "Sorted Array using Bubble sort: ";
    for(int i = 0; i < size; i++){
        std::cout << arr[i] << " - ";
    }
    std::cout << "End of the Array\n\n";

    countVotes(arr, size);

    distance();
    return 0;
}
```

ANALYSIS:

For my analysis here, there's not much logic here. Since I just called the functions that I created in the header file. The most important thing here are the library that I used, we can't use or write on this without it. The function for the cstdlib is for generating a set of numbers, randomly. This will not exceed a size of 100, because we declared a variable that will hold the constant size of the elements that I'll use. The ctime library wasn't used, since it will change the random numbers when we run the program and this will make it hard for me. The variable declared here, which is the size, to put it to the array we set it to another variable to overwrite the initial declaration. Then make it to an array by putting it to a bracket [], by doing all this we now have an array that has 100 elements and will take a value that is integer. Although, it is actual just a 100 since the starting point of an array is 0, which we can deduce as n-1. Then I created a function to separate and make it organized and easy to see, once we run the actual code. At this point I'm just calling the functions inside the header file, nothing much. But the bubble sort which has a 2 parameter for the array and the size which is the 100. same for the countVotes.

HEADER:

```cpp
#ifndef supp_sorting_h
#define supp_sorting_h
#include <algorithm>

template<typename T>
void randomNum(T arr[], size_t arrSize){
    for(int i = 0; i < arrSize; i++){
        arr[i] = (rand() % 5) + 1;
    }
    std::cout<<"Random Numbers generated: ";
    for(int i = 0; i <arrSize; i++){
        std::cout<< arr[i] << " - ";
    }
}

template<typename T>
void bubbleSort(T arr[], size_t arrSize){
    int swap = 0;
    int comparison = 0;

    //will iterate the elements of the array
    for(size_t i = 0; i < arrSize; i++){
        //will compare and swap the value
        for(size_t j = 0; j < arrSize; j++){
            comparison ++;
            if(arr[i] < arr[j]){
                std::swap(arr[i], arr[j]);
                swap++;
            }
        }
    }
    std::cout<<"Swap: " << swap << std::endl;
    std::cout<<"Comparison: " << comparison<<std::endl;
}

template<typename T>
void countVotes(T arr[], size_t arrSize) {
    int count[5] = {0};

    for (size_t i = 0; i < arrSize; i++) {
        if (arr[i] >= 1 && arr[i] <= 5)
            count[arr[i] - 1]++;
    }

    std::cout << "Manual Count:\n";
    for (int i = 0; i < 5; i++) {
        std::cout << "Candidate " << (i + 1) << ": " << count[i] << " votes\n";
    }


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

    std::cout << "Winning Candidate: " << winner << " with " << maxVotes << " votes\n";
}

#endif
```

ANALYSIS:

For my analysis here, which is the header, I first added the algorithm library to make it easier for me to implement. Then I created a template to take any data we want since that is how templating works. Then I created a function to generate a random set of numbers. It takes 2 parameters: the array, which is declared at the main function and the size_t this is like a data type. Since we declared the arrSize variable using this type, this is included at the cstdlib that is included at the main function. The 2 parameters at the functions makes sense, because we also call arr and the size at the make function, which means that we took the value of the variables at the main to make this happen. as for the rand function this is also included at the cstdlib. The logic behind the 5 + 1 is that since the candidate is 5 if we use 5 the actual number of it in the index is just 4 that is why I added a +1. then another for loop to display all the random generated numbers. Then I created a function for the bubble sort which takes parameters. The explanation is just like the randomNum since that is all the value that we needed to work. I created a 2 variable to count the swaps it will make and the comparison then set it to 0 this will increment whenever the program swaps the value or compares them.
I created a for loop that will iterate the elements of the array, this i will run and run till all the elements are sorted. then another for loop for comparing and swapping the std::swap is one of the functions of the algorithm which made it easier for me since I don't have to manually do the logic. Then the countVotes function to decide who the winner of the voting. I first created a variable that has a size of 5 and give it an element 0. for the first loop I created this because I want to map all the candidates to make it easier for me. then the manual count to give the candidates their voites.

OUTPUT:

```
SORTING ALGORITHM----------------------------------------------------------------SORTING ALGORITHM


Bubble sort
Swap: 190
Comparison: 10000
Sorted Array using Bubble sort: 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 - 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 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - End of the Array

Manual Count:
Candidate 1: 14 votes
Candidate 2: 25 votes
Candidate 3: 20 votes
Candidate 4: 21 votes
Candidate 5: 20 votes
Winning Candidate: 2 with 25 votes
SORTING ALGORITHM----------------------------------------------------------------SORTING ALGORITHM
PS C:\Users\nikko\Desktop\c++>
```

PSEUDO
START
1Generate array A[0...99] with random values from 1 to 5.
Sort array using Bubble Sort.
Initialize count[5] = {0, 0, 0, 0, 0}.
For each element in A:
count[element - 1] += 1
Find index of max value in count[].
Output winning candidate and vote counts.
END

Candidate 1: 14 votes
Candidate 2: 25 votes
Candidate 3: 20 votes
Candidate 4: 21 votes
Candidate 5: 20 votes

8. Conclusion

To conclude, to make it easier for me I needed to research more about the library of c++. In this HOA I learned so much because of the complex logic and the functions of some useful library like the algorithm, this made it easier for me since I didn't have to use a temporary variable to store the value that will be compared. The declaration of the arrSize which has a type of size_t also helped here since this will count the actual index of the array. I used the bubble sort because I saw that the swaps are smaller compared to the other 2, which made sense because 1 it is easier to implement and to use. This is the most basic out of all the algorithms that are tasked to use. Since, sorting an array of 100 elements, each being a vote from 1 to 5. That's a small enough dataset where Bubble Sort's inefficiency doesn't really hurt performance. The comparison doesn't count no matter how many there were, the swap is the most important and it is smaller compared to the 2. Plus, it allows us to easily track comparison and swap for detailed analysis.

9. Assessment Rubric