Activity No. 7	
Sorting Algorithms	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** October 16, 2024
Section: CPE21S4	**Date Submitted:** October 17, 2024
Name(s): Bona, Andrei Nycole So	**Instructor:** Prof. Maria Rizette Sayo

6. Output

Code + Console Screenshot

```
// Array of Values for Sort Algorithm Testing
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
   const int max_size = 100;

   // generate random values
   int arr[max_size];
   srand(time(0));
   for (int i = 0; i < max_size; i++)
         {
      arr[i] = rand() % 100;
   }

   // show your datasets content
   cout << "Unsorted array: " <<endl;
   for (int i = 0; i < max_size; i++)
         {
      cout << arr[i] << " ";
   }
   return 0;
}
```

```
input
Unsorted array:
65 89 3 13 32 52 17 45 98 28 45 74 12 72 70 99 14 25 48 89 66 78 75 32 6 82 80 85
 1 83 54 18 72 57 32 57 61 1 54 60 29 99 86 93 23 56 45 89 81 93 78 0 72 6 32 78
40 65 15 94 48 69 12 72 78 96 29 40 49 36 0 30 87 38 24 11 94 69 0 27 62 31 27 34
 37 12 64 29 77 80 23 77 1 88 49 80 36 31 72 86

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

Observations

The code generates an array of 100 random integers between 0 and 99 and prints the unsorted array. It uses srand(time(0)) to ensure different random values with each execution.

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

Code + Console Screenshot	**main.cpp**
	
	#include <iostream>
	#include <cstdlib>
	#include <ctime>
	#include "sort.h"
	using namespace std;
	
	int main()
	{
	const int max_size = 100;
	
	// generate random values
	int arr[max_size];
	srand(time(0));
	for (int i = 0; i < max_size; i++)
	{
	arr[i] = rand() % 100;
	}
	
	// show your array content
	cout << "Unsorted array: " <<endl;
	for (int i = 0; i < max_size; i++)
	{
	cout << arr[i] << " ";
	}
	
	size_t arrSize = sizeof(arr) / sizeof(arr[0]);
	
	cout <<endl;
	cout <<endl;
	
	bubbleSort(arr, arrSize);
	cout << "Sorted array: " << endl;
	for (size_t i = 0; i < arrSize; i++) {
	cout << arr[i] << " ";
	}
	cout << endl;
	return 0;
	}
		sort.h
		
		// Bubble Sort Technique
		#include <iostream>
		#include <cstdlib>
		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]
		swap(arr[j], arr[i]);
		}
		//[End of Inner for loop]
		}
		//[End if Outer for loop]
		}
		//Step 4: Exit
		}

```
Unsorted array:
26 64 65 58 34 23 54 18 74 82 49 22 21 37 22 80 60 61 99 13 79 65 49 93 68 11 60
67 83 24 55 10 40 21 20 74 96 74 44 23 56 93 45 77 82 20 57 43 81 9 8 13 74 9 58
42 20 71 62 4 47 69 14 88 42 86 62 39 61 59 62 17 4 59 47 87 79 56 82 61 65 90 74
 91 99 84 34 72 55 48 76 3 17 42 91 60 28 5 99 89

Sorted array:
99 99 99 96 93 93 91 91 90 89 88 87 86 84 83 82 82 82 81 80 79 79 77 76 74 74 74
74 74 72 71 69 68 67 65 65 65 64 62 62 62 61 61 61 60 60 60 59 59 58 58 57 56 56
55 55 54 49 49 48 47 47 45 44 43 42 42 42 40 39 37 34 34 28 26 24 23 23 22 22 21
21 20 20 20 18 17 17 14 13 13 11 10 9 9 8 5 4 4 3


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

Observations	The main.cpp file handles the array generation, printing, and calling the bubbleSort function from sort.h. The bubbleSort function sorts the array in descending order by swapping elements using two nested loops.

Table 7-2. Bubble Sort Technique

Code + Console Screenshot

main.cpp

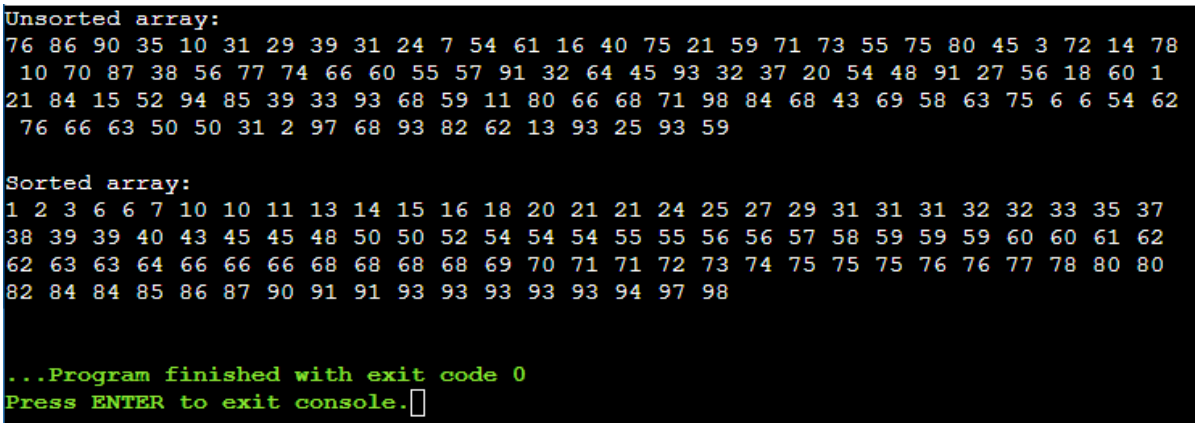
```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "sort.h"
using namespace std;

int main()
{
   const int max_size = 100;

   // generate random values
   int arr[max_size];
   srand(time(0));
   for (int i = 0; i < max_size; i++)
         {
      arr[i] = rand() % 100;
   }

   // show your array content
   cout << "Unsorted array: " <<endl;
   for (int i = 0; i < max_size; i++)
         {
      cout << arr[i] << " ";
   }

   size_t arrSize = sizeof(arr) / sizeof(arr[0]);

   cout <<endl;
   cout <<endl;
```

sort.h

```
// Selection Sort Algorithm
#include <iostream>
#include <cstdlib>
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;
```

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



Observations	In main.cpp, the random array is created and displayed, while the selectionSort function in sort.h sorts the array by repeatedly finding and swapping the smallest element. The sorted array is then printed.

Table 7-3. Selection Sort Algorithm

Code + Console Screenshot	**main.cpp**
	
	#include <iostream>
	#include <cstdlib>
	#include <ctime>
	#include "sort.h"
	using namespace std;
	
	int main()
	{
	const int max_size = 100;
		sort.h
		
		// Insertion Sort Algorithm
		#include <iostream>
		#include <cstdlib>
		using namespace std;
		
		//General Algorithm
		//Insertion Sort
		template <typename T>
		void insertionSort(T arr[], const int N)
		{

```
    // generate random values
    int arr[max_size];
    srand(time(0));
    for (int i = 0; i < max_size; i++)
            {
        arr[i] = rand() % 100;
    }

    // show your array content
    cout << "Unsorted array: " <<endl;
    for (int i = 0; i < max_size; i++)
            {
        cout << arr[i] << " ";
    }

    size_t arrSize = sizeof(arr) / sizeof(arr[0]);

    cout <<endl;
    cout <<endl;

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

```
    int K = 0, J, temp;
    //Step 1: Repeat Steps 2 to 5 for K = 1 to N-1
    while(K < N)
    {
        //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]
        while((J >= 0 && temp < arr[J])
        {
            //set A[J + 1] = A[J]
            arr[J+1] = arr[J];
            //set J = J – 1
            J--;
            //[end of inner loop]
        }
        //Step 5: set A[J + 1] = temp
        arr[J+1] = temp;
        //[end of loop]
        K++;
    }
    //Step 6: exit
}
```

```
Unsorted array:
98 88 77 5 93 35 8 24 80 52 98 68 33 65 52 19 83 26 85 8 81 78 44 73 26 67 76 18
1 88 40 0 76 17 57 21 4 18 45 84 22 44 4 55 61 57 74 45 35 11 53 16 41 49 42 67 6
8 18 86 70 58 26 22 86 95 79 7 99 49 5 36 72 1 40 27 62 49 54 59 85 65 65 1 7 66
43 74 35 13 60 57 71 38 79 9 33 10 17 85 60

Sorted array:
0 1 1 1 4 4 5 5 7 7 8 8 9 10 11 13 16 17 17 18 18 18 19 21 22 22 24 26 26 26 27 3
3 33 35 35 35 36 38 40 40 41 42 43 44 44 45 45 49 49 49 52 52 53 54 55 57 57 57 5
8 59 60 60 61 62 65 65 65 66 67 67 68 68 70 71 72 73 74 74 76 76 77 78 79 79 80 8
1 83 84 85 85 85 86 86 88 88 93 95 98 98 99


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

Observations	The insertionSort function, defined in sort.h, sorts the array by comparing and inserting elements into their correct positions. After sorting, the sorted array is printed. This method is efficient for small datasets and ensures the array is sorted in ascending order.

Table 7-4. Insertion Sort Algorithm

7. Supplementary Activity

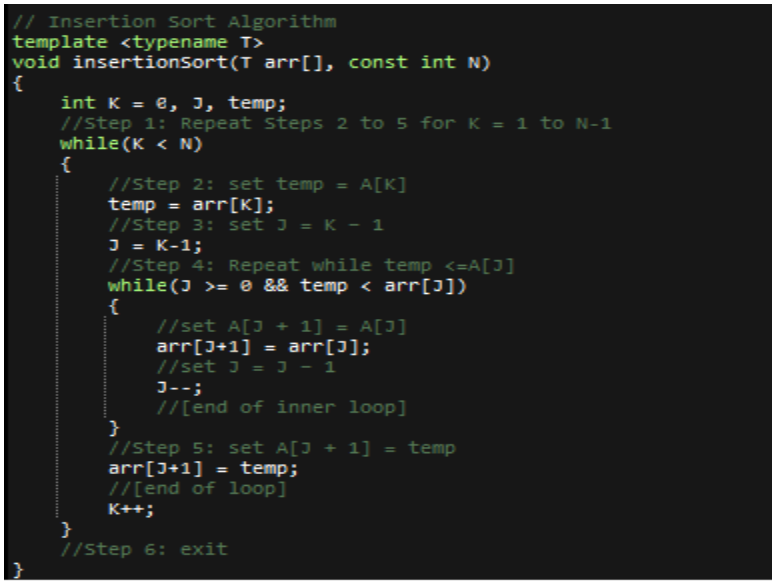
Deliverables:

- **Pseudocode of Algorithm**

```
START

FUNCTION main
   // Constants
   SET arrSize = 101
   SET candidates = 5

   // Arrays to store votes and count
   DECLARE votes[arrSize]
   DECLARE voteCount[6] = {0}  // Count for candidates 1 to 5

   // Seed for random number generation
   CALL srand(time(0))

   // Generate random votes from 1 to 5
   FOR i FROM 0 TO arrSize - 1
      SET votes[i] = random number between 1 and 5  // Randomly select a candidate
   END FOR

   // Display the unsorted random votes
   PRINT "Unsorted array:"
   FOR i FROM 0 TO arrSize - 1
      PRINT votes[i]
   END FOR
   PRINT NEWLINE

   // Sort the votes
   CALL insertionSort(votes, arrSize)

   // Count the votes
   FOR i FROM 0 TO arrSize - 1
      INCREMENT voteCount[votes[i]]  // Count the votes for each candidate
   END FOR

   // Display the sorted random votes
   PRINT "Sorted array:"
   FOR i FROM 0 TO arrSize - 1
      PRINT votes[i]
   END FOR
   PRINT NEWLINE

   // Determine the winning candidate
   SET maxVotes = 0
   SET winningCandidate = 0

   FOR i FROM 1 TO candidates
      PRINT "Candidate ", i, ": ", voteCount[i], " votes"
```

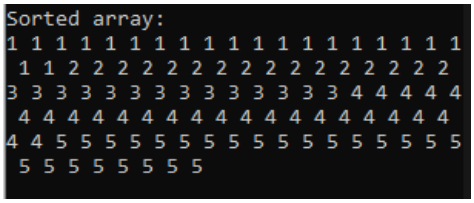
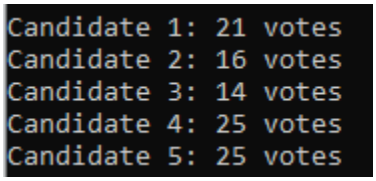
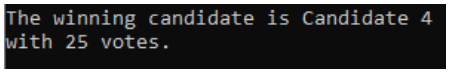
```
        IF voteCount[i] > maxVotes THEN
            SET maxVotes = voteCount[i]
            SET winningCandidate = i
        END IF
    END FOR

    // Output the winning candidate
    PRINT "The winning candidate is Candidate ", winningCandidate, " with ", maxVotes, " votes."

END FUNCTION

// Function to perform selection sort
FUNCTION selectionSort(arr[], N)
    FOR i FROM 0 TO N - 1
        SET POS = Routine_Smallest(arr, i, N)  // Find the position of the smallest element
        SWAP arr[i] WITH arr[POS]  // Swap the smallest element with the first unsorted element
    END FOR
END FUNCTION

// Function to find the position of the smallest element
FUNCTION Routine_Smallest(A[], K, arrSize)
    SET smallestElem = A[K]
    SET position = K
    FOR J FROM K + 1 TO arrSize - 1
        IF A[J] < smallestElem THEN
            SET smallestElem = A[J]
            SET position = J
        END IF
    END FOR
    RETURN position
END FUNCTION

END
```

• Screenshot of Algorithm Code



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

• **Output Testing**

Output Console Showing Sorted Array	Manual Count	Count Result of Algorithm
		


```
Unsorted array:
5 3 2 4 1 3 2 4 3 5 4 4 4 4 5 1 2 4 5 1 2 4 2 4 4 3 1 1 4 3 5 1 2 5 4 1 5 2 1 5 5 1 1
 3 5 4 2 1 3 1 5 3 5 5 5 4 5 4 4 5 4 2 4 4 5 2 1 5 2 1 5 4 2 4 5 1 5 1 3 2 1 4 5 1 2
1 3 5 3 4 5 2 1 3 5 1 3 3 4 2 4

Sorted array:
1 1 1 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 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 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

Candidate 1: 21 votes
Candidate 2: 16 votes
Candidate 3: 14 votes
Candidate 4: 25 votes
Candidate 5: 25 votes
The winning candidate is Candidate 4 with 25 votes.

--------------------------------
Process exited after 0.02066 seconds with return value 0
Press any key to continue . . .
```

Question: Was your developed vote counting algorithm effective? Why or why not?

The vote counting algorithm using insertion sort is effective for small datasets because it is simple to implement and performs well with nearly sorted data, offering a time complexity in the best case. It generates random votes, counts them, and identifies the winning candidate clearly and efficiently.

8. Conclusion

In this laboratory, I learned about sorting algorithms, particularly how bubble, selection, and insertion sort works. I gained hands-on experience in generating random data, displaying arrays, and implementing sorting techniques. The procedure involved creating an array of random integers and using the three techniques for sorting to arrange them in order, with the code organized into main.cpp for generating and displaying the array and sort.h for the sorting algorithm.

The activity helped me understand the logic and efficiency of different sorting methods, highlighting how algorithm choice impacts performance based on dataset size. Overall, I believe I did well by successfully implementing the three algorithms. However, I see areas for improvement, such as learning more about complex sorting algorithms, adding error handling, and optimizing for larger datasets. This experience reinforced my coding skills and enhanced my understanding of algorithm design.

9. Assessment Rubric