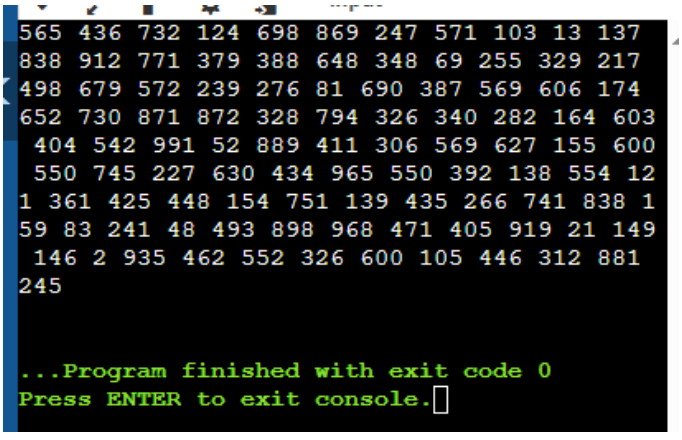
Activity No. 7	
Hands-on Activity 7.2 Sorting Algorithms	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** October 21, 2024
Section: CPE21S4	**Date Submitted:** October 23, 2024
Name(s): Don Eleazar T. Fernandez	**Instructor:** Maria Rizette Sayo

6. Output

Code + Console Screenshot	#include <iostream>
	#include <ctime>
	using namespace std;
	
	int main() {
	  const int size = 100;
	  int array[size];
	  srand(static_cast<unsigned int>(time(0)));
	
	  for (int i = 0; i < size; ++i) {
	  array[i] = rand() % 1000 + 1;
	  }
	
	  for (int i = 0; i < size; ++i) {
	  cout << array[i] << " ";
	  }
	  cout << endl;
	
	  return 0;
	}
	 

Observations	The program has performed as intended, which is to create a one hundred random elements of an array.

Table 8 - 1. Array of Values for Sort Algorithm Testing

Code + Console Screenshot	#include <iostream>

```
#include <ctime>
using namespace std;

void shellSort(int array[], int size) {
    for (int interval = size / 2; interval > 0; interval /= 2) {
        for (int i = interval; i < size; i++) {
            int temp = array[i];
            int j;
            for (j = i; j >= interval && array[j - interval] > temp; j
-= interval) {
                array[j] = array[j - interval];
            }
            array[j] = temp;
        }
    }
}

int main() {
    const int size = 100;
    int array[size];
    srand(static_cast<unsigned int>(time(0)));

    for (int i = 0; i < size; ++i) {
        array[i] = rand() % 1000 + 1;
    }
    cout << "Unsorted Array: ";
    for (int i = 0; i < size; ++i) {
        cout << array[i] << " ";
    }
    cout << "\n\n";

    shellSort(array, size);
    cout << "Sorted Array: ";
    for (int i = 0; i < size; ++i) {
        cout << array[i] << " ";
    }
    cout << endl;

    return 0;
}
```

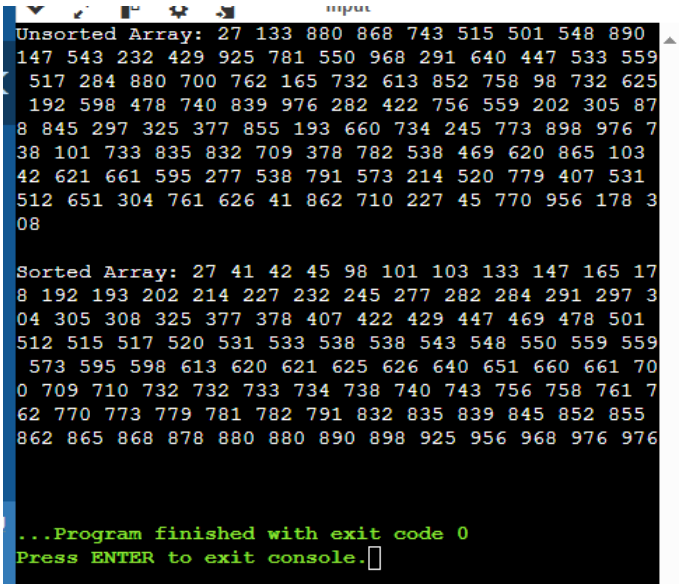
	
Observations	The program sorts the declared array with the Shell Sort method. It first displays the unsorted elements and then displays them sorted.

Table 8 - 2. Shell Sort Technique

Code + Console Screenshot	#include <iostream>
	#include <ctime>
	#include <vector>
	using namespace std;
	
	void merge(int arr[], int left, int middle, int right) {
	  int n1 = middle - left + 1;
	  int n2 = right - middle;
	  vector<int> L(n1), R(n2);
	  for (int i = 0; i < n1; i++)
	  L[i] = arr[left + i];
	  for (int j = 0; j < n2; j++)
	  R[j] = arr[middle + 1 + j];
	
	  int i = 0, j = 0, k = left;
	  while (i < n1 && j < n2) {
	  if (L[i] <= R[j]) {
	  arr[k] = L[i];
	  i++;
	  } else {
	  arr[k] = R[j];
	  j++;
	  }
	  k++;
	  }
	  while (i < n1) {
	  arr[k] = L[i];
	  i++;

```
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void merge_sort(int arr[], int left, int right) {
    if (left < right) {
        int middle = left + (right - left) / 2;
        merge_sort(arr, left, middle);
        merge_sort(arr, middle + 1, right);
        merge(arr, left, middle, right);
    }
}

int main() {
    const int size = 100;
    int array[size];
    srand(static_cast<unsigned int>(time(0)));

    for (int i = 0; i < size; ++i) {
        array[i] = rand() % 1000 + 1;
    }

    cout << "Original array: ";
    for (int i = 0; i < size; ++i) {
        cout << array[i] << " ";
    }
    cout << "\n\n";

    merge_sort(array, 0, size - 1);

    cout << "Sorted array: ";
    for (int i = 0; i < size; ++i) {
        cout << array[i] << " ";
    }
    cout << endl;

    return 0;
}
```

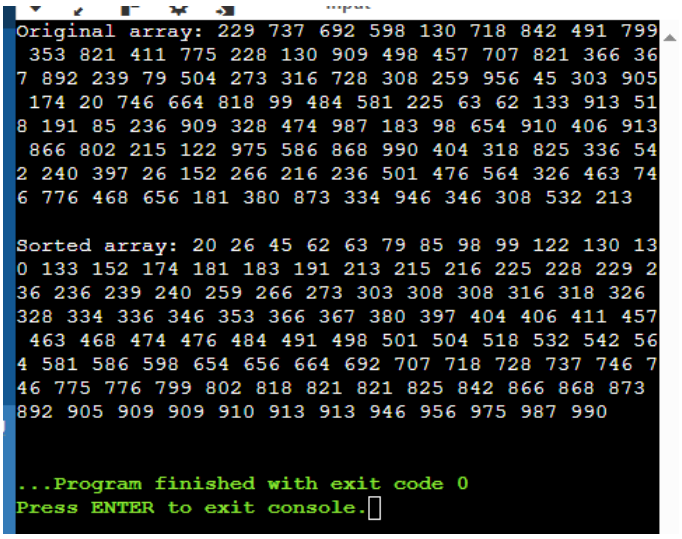
	
Observations	The program sorts the declared array with the Merge Sort method. It first displays the unsorted elements and then displays them sorted.

Table 8 - 3. Merge Sort Algorithm

Code + Console Screenshot	#include <iostream>
	#include <ctime>
	using namespace std;
	
	int partition(int arr[], int low, int high) {
	  int pivot = arr[high];
	  int i = low - 1;
	
	  for (int j = low; j < high; j++) {
	  if (arr[j] < pivot) {
	  i++;
	  swap(arr[i], arr[j]);
	  }
	  }
	  swap(arr[i + 1], arr[high]);
	  return i + 1;
	}
	
	void quicksort(int A[], int low, int high) {
	  if (low < high) {
	  int pivot = partition(A, low, high);
	  quicksort(A, low, pivot - 1);
	  quicksort(A, pivot + 1, high);
	  }
	}
	
	int main() {
	  const int size = 100;
	  int array[size];

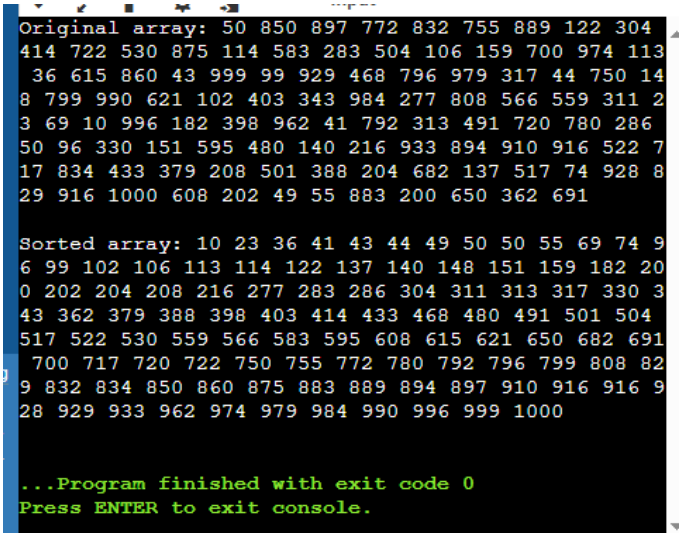
```
    srand(static_cast<unsigned int>(time(0)));

    for (int i = 0; i < size; ++i) {
        array[i] = rand() % 1000 + 1;
    }
    cout << "Original array: ";
    for (int i = 0; i < size; ++i) {
        cout << array[i] << " ";
    }
    cout << "\n\n";

    quicksort(array, 0, size - 1);

    cout << "Sorted array: ";
    for (int i = 0; i < size; ++i) {
        cout << array[i] << " ";
    }
    cout << endl;

    return 0;
}
```



Observations	The program sorts the declared array with the Quick Sort method. It first displays the unsorted elements and then displays them sorted.

Table 8 - 4. Quick Sort Algorithm

7. Supplementary Activity

Problem 1: Can we sort the left sub list and right sub list from the partition method in quick sort using other sorting algorithms? Demonstrate an example.

- Yes, the left and right sub list from the partition method in quick sort can be sorted with other sort method, such as the shell method. An example of that is the image below, where it is applied in the shell sort.

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

void shellSort(int arr[], int size) {
    for (int gap = size / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < size; i++) {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {
                arr[j] = arr[j - gap];
            }
            arr[j] = temp;
        }
    }
}

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivotIndex = partition(arr, low, high);
        int leftSize = pivotIndex - low;
        int rightSize = high - pivotIndex;
        int left[leftSize];
        int right[rightSize];

        for (int i = 0; i < leftSize; i++) {
            left[i] = arr[low + i];
        }
        for (int i = 0; i < rightSize; i++) {
            right[i] = arr[pivotIndex + 1 + i];
        }

        shellSort(left, leftSize);
        shellSort(right, rightSize);
        for (int i = 0; i < leftSize; i++) {
            arr[low + i] = left[i];
        }
        arr[pivotIndex] = arr[pivotIndex];
        for (int i = 0; i < rightSize; i++) {
            arr[pivotIndex + 1 + i] = right[i];
        }
    }
}

int main() {
    int array[] = {5, 4, 3, 2, 1, 6, 7, 8, 9, 10};
    int size = sizeof(array) / sizeof(array[0]);

    cout << "Original Array: ";
    for (int num : array) {
        cout << num << " ";
    }
    cout << endl;

    quickSort(array, 0, size - 1);

    cout << "Sorted Array: ";
    for (int num : array) {
        cout << num << " ";
    }
    cout << endl;
    return 0;
}
```

```
Original Array: 5 4 3 2 1 6 7 8 9 10
Sorted Array: 1 2 3 4 5 6 7 8 9 10


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

Problem 2: Suppose we have an array which consists of {4, 34, 29, 48, 53, 87, 12, 30, 44, 25, 93, 67, 43, 19, 74}. What sorting algorithm will give you the fastest time performance? Why can merge sort and quick sort have O(N • log N) for their time complexity?

- For the array, Quick Sort will give the fastest time performance due to its average case efficiency performance. The merge sort and quick sort have O(N • log N) for their time complexity, because merge sort divides the array into halves, sort it, and then merge it back, while quick sort partitions the array around a pivot and sort the array.

8. Conclusion

To conclude, through the laboratory activity that I have done the shell, merge, and quick sort method was demonstrated that it can be applied to an array. The merge sort splits the list into halves, sorts them, and merges them back together. And the shell sort is that sorts elements that are not in order and is sorted later on. Then the quick sort picks a partition to

divide the list and sorts them. In the supplementary activity, I have experimented if the partition part of the quick sort can be utilized to other sort method, and it could be done.
9. Assessment Rubric

I affirm that I will not give or receive any unauthorized help on this activity/exam and that all work will be my own.