Activity No. 8	
Hands-on Activity 7.2 Sorting Algorithms	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 10 / 21 / 2024
Section: CpE21 S4	**Date Submitted:** 10 / 21 / 2024
Name(s): BONIFACIO, NYKO ADREIN	**Instructor:** Prof. Sayo

6. Output

Code + Console Screenshot

main.cpp

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

int main() {
    int arr[100];

    for (int i = 0; i < 100; ++i) {
        arr[i] = rand() % 100;
    }

    for (int i = 0; i < 100; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}

```

main.cpp

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

int main() {
    int arr[100];

    for (int i = 0; i < 100; ++i) {
        arr[i] = rand() % 100;
    }

    for (int i = 0; i < 100; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}

```

Share Run

Output Clear

```
/tmp/mq30apUKSo.o
83 86 77 15 93 35 86 92 49 21 62 27 90 59 63 26 40 26 72 36 11 68 67 29 82 30 62 23 67 35 29 2 22
    58 69 67 93 56 11 42 29 73 21 19 84 37 98 24 15 70 13 26 91 80 56 73 62 70 96 81 5 25 84 27 36
    5 46 29 13 57 24 95 82 45 14 67 34 64 43 50 87 8 76 78 88 84 3 51 54 99 32 60 76 68 39 12 26
    86 94 39


=== Code Execution Successful ===
```

Observation	This code creates an array of elements with random values and outputs 100 random numbers that are not sorted.

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

Code + Console Screenshot	

Untitled1.cpp | SortingAlgorithms.h | main.cpp

```cpp
#ifndef SORTINGALGORITHMS_H
#define SORTINGALGORITHMS_H

#include <iostream>
#include <cstdlib>
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;
        }
    }
}

#endif
```

Untitled1.cpp | SortingAlgorithms.h | main.cpp

```cpp
#include "SortingAlgorithms.h"

int main() {
    int arr[100];

    for (int i = 0; i < 100; ++i) {
        arr[i] = rand() % 100;
    }

    cout << "Original array:" << endl;
    for (int i = 0; i < 100; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

    shellSort(arr, 100);

    cout << "Sorted array:" << endl;
    for (int i = 0; i < 100; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Untitled1.cpp | SortingAlgorithms.h | main.cpp

```cpp
#include "SortingAlgorithms.h"

int main() {
    int arr[100];
```

C:\Users\TIPQC\Downloads\dsa\main.exe

```
Original array:
41 67 34 0 69 24 78 58 62 64 5 45 81 27 61 91 95 42 27 36 91 4 2 53 92 82 21 16 18 95 47 26 71 38 69 12 67 99 35 94 3 11
 22 33 73 64 41 11 53 68 47 44 62 57 37 59 23 41 29 78 16 35 90 42 88 6 40 42 64 48 46 5 90 29 70 50 6 1 93 48 29 23 84
54 56 40 66 76 31 8 44 39 26 23 37 38 18 82 29 41
Sorted array:
0 1 2 3 4 5 5 6 6 8 11 11 12 16 16 18 18 21 22 23 23 23 24 26 26 27 27 29 29 29 29 31 33 34 35 35 36 37 37 38 38 39 40 4
0 41 41 41 41 42 42 42 44 44 45 46 47 47 48 48 50 53 53 54 56 57 58 59 61 62 62 64 64 64 66 67 67 68 69 69 70 71 73 76 7
8 78 81 82 82 84 88 90 90 91 91 92 93 94 95 95 99

--------------------------------
Process exited after 0.05453 seconds with return value 0
Press any key to continue . . .
```

Observation	The code demonstrates how to sort an array of random numbers using Shell Sort in C++. It uses a header file to define the Shell Sort function separately, which helps keep

	the code organized. In the main program, an array of random numbers is generated, sorted using the Shell Sort function, and then both the unsorted and sorted arrays are displayed.

Table 8-2. Shell Sort Technique

Code + Console Screenshot	Untitled1.cpp × [*] SortingAlgorithms.h × ma

```cpp
#ifndef SORTINGALGORITHMS_H
#define SORTINGALGORITHMS_H

#include <iostream>
using namespace std;

void merge(int arr[], int left, int middle, int right) {
    int n1 = middle - left + 1;
    int n2 = right - middle;

    int* L = new int[n1];
    int* R = new int[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++;
    }

    delete[] L;
    delete[] R;
}

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

        mergeSort(arr, left, middle);
        mergeSort(arr, middle + 1, right);
        merge(arr, left, middle, right);
    }
}

#endif
```

Untitled1.cpp | [*] SortingAlgorithms.h | main.cpp

```cpp
#include <iostream>
#include <cstdlib>
#include "SortingAlgorithms.h"
using namespace std;

int main() {
    int arr[100];

    for (int i = 0; i < 100; ++i) {
        arr[i] = rand() % 100;
    }

    cout << "Original array:" << endl;
    for (int i = 0; i < 100; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

    mergeSort(arr, 0, 99);

    cout << "Sorted array:" << endl;
    for (int i = 0; i < 100; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Untitled1.cpp | SortingAlgorithms.h | main.cpp

```cpp
#include <iostream>
```

C:\Users\TIPQC\Downloads\dsa\main.exe

```
Original array:
41 67 34 0 69 24 78 58 62 64 5 45 81 27 61 91 95 42 27 36 91 4 2 53 92 82 21 16 18 95 47 26 71 38 69 12 67 99 35 94 3 11
 22 33 73 64 41 11 53 68 47 44 62 57 37 59 23 41 29 78 16 35 90 42 88 6 40 42 64 48 46 5 90 29 70 50 6 1 93 48 29 23 84
54 56 40 66 76 31 8 44 39 26 23 37 38 18 82 29 41
Sorted array:
0 1 2 3 4 5 5 6 6 8 11 11 12 16 16 18 18 21 22 23 23 23 24 26 26 27 27 29 29 29 29 31 33 34 35 35 36 37 37 38 38 39 40 4
0 41 41 41 41 42 42 42 44 44 45 46 47 47 48 48 50 53 53 54 56 57 58 59 61 62 62 64 64 64 66 67 67 68 69 69 70 71 73 76 7
8 78 81 82 82 84 88 90 90 91 91 92 93 94 95 95 99

--------------------------------
Process exited after 0.05166 seconds with return value 0
Press any key to continue . . .
```

Observation	The provided code demonstrates how to sort an array of random numbers using Merge Sort in C++. It separates the Merge Sort implementation into a header file, which keeps the code organized. In the main program, an array of random integers is generated, sorted with the Merge Sort function, and both the original and sorted arrays are displayed.

Table 8-3. Merge Sort Algorithm

Code + Console Screenshot	Untitled1.cpp × SortingAlgorithms.h × main.cpp ×

```cpp
#ifndef SORTINGALGORITHMS_H
#define SORTINGALGORITHMS_H

#include <iostream>
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 arr[], int low, int high) {
    if (low < high) {
        int pivot = partition(arr, low, high);
        quickSort(arr, low, pivot - 1);
        quickSort(arr, pivot + 1, high);
    }
}

#endif
```

Untitled1.cpp | SortingAlgorithms.h | main.cpp

```cpp
#include <iostream>
#include <cstdlib>
#include "SortingAlgorithms.h"
using namespace std;

int main() {
    int arr[100];


    for (int i = 0; i < 100; ++i) {
        arr[i] = rand() % 100;
    }

    cout << "Original array:" << endl;
    for (int i = 0; i < 100; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

    quickSort(arr, 0, 99);

    cout << "Sorted array:" << endl;
    for (int i = 0; i < 100; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}

```

Untitled1.cpp | SortingAlgorithms.h | main.cpp

```cpp
#include <iostream>
#include <cstdlib>
#include "SortingAlgorithms.h"
using namespace std;
```

C:\Users\TIPQC\Downloads\dsa\main.exe

```
Original array:
41 67 34 0 69 24 78 58 62 64 5 45 81 27 61 91 95 42 27 36 91 4 2 53 92 82 21 16 18 95 47 26 71 38 69 12 67 99 35 94 3 11
 22 33 73 64 41 11 53 68 47 44 62 57 37 59 23 41 29 78 16 35 90 42 88 6 40 42 64 48 46 5 90 29 70 50 6 1 93 48 29 23 84
64 56 40 66 76 31 8 44 39 26 23 37 38 18 82 29 41
Sorted array:
0 1 2 3 4 5 5 6 6 8 11 11 12 16 16 18 18 21 22 23 23 23 24 26 26 27 27 29 29 29 29 31 33 34 35 35 36 37 37 38 38 39 40 4
0 41 41 41 41 42 42 42 44 44 45 46 47 47 48 48 50 53 53 54 56 57 58 59 61 62 62 64 64 64 66 67 67 68 69 69 70 71 73 76 7
8 78 81 82 82 84 88 90 90 91 91 92 93 94 95 95 99

--------------------------------
Process exited after 0.054 seconds with return value 0
Press any key to continue . . .
```

Observation	The code demonstrates how to sort an array of random numbers using Quick Sort in C++. The Quick Sort implementation is organized in a separate header file, which makes the code easier to read and manage. In the main program, an array of random integers is created, sorted with the Quick Sort function, and both the original and sorted arrays are displayed.

Table 8-4. Quick Sort Algorithm

7. Supplementary Activity

Solve given data sorting problems using appropriate basic sorting algorithms

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, we can sort the left and right sub-lists using other sorting algorithms. For example, we could use merge sort or shell sort on the smaller sub-lists after partitioning.

Example:

Array: {34, 7, 23, 32, 5, 62} *with pivot 23*.
Left Sub-list: {7, 5} (sort using Bubble Sort)
Result: {5, 7}
Right Sub-list: {34, 32, 62} (sort using Selection Sort)
Result: {32, 34, 62}
Final Sorted Array: Combine them: {5, 7, 23, 32, 34, 62}

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 {4, 34, 29, 48, 53, 87, 12, 30, 44, 25, 93, 67, 43, 19, 74}, quick sort will probably be the fastest. Merge sort and quick sort can sort in logarithmic time because they split the array into smaller parts and then solve those parts.

8. Conclusion

In this activity, I learned to sort numbers in C++ using Shell Sort, Merge Sort, and Quick Sort. Quick Sort is fast and sorts the whole list quickly. Merge Sort breaks the list into smaller parts and sorts them. Shell Sort sorts numbers at certain gaps. Using these different sorting methods helps figure out the best way to organize data.

9. Assessment Rubric