Activity No. <7>	
SORTING ALGORITHMS: BUBBLE, SELECTION, AND INSERTION SORT	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 9/18/25**
Section: CPE21S4	**Date Submitted: 9/18/25**
Name(s): Quioyo, Angelo	**Instructor: Engr. Jimlord Quejado**

6. Output:

ILO_A:

Main CPP:

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

using namespace std;

const int SIZE = 100;

int main() {
    srand(static_cast<unsigned int>(time(0)));

    int originalArray[SIZE];

    for(int i = 0; i < SIZE; ++i) {
        originalArray[i] = rand() % 1000;
    }

    cout << "Original Array:\n";
    printArray(originalArray, SIZE);

    int selectionArray[SIZE];
    int bubbleArray[SIZE];
    int insertionArray[SIZE];
    int mergeArray[SIZE];

    copy(begin(originalArray), end(originalArray), begin(selectionArray));
    copy(begin(originalArray), end(originalArray), begin(bubbleArray));
    copy(begin(originalArray), end(originalArray), begin(insertionArray));
    copy(begin(originalArray), end(originalArray), begin(mergeArray));

    selectionSort(selectionArray, SIZE);
    cout << "\nSelection Sort:\n";
    printArray(selectionArray, SIZE);

    bubbleSort(bubbleArray, SIZE);
    cout << "\nBubble Sort:\n";
    printArray(bubbleArray, SIZE);
```

```cpp
    cout << "Original Array:\n";
    printArray(originalArray, SIZE);

    int selectionArray[SIZE];
    int bubbleArray[SIZE];
    int insertionArray[SIZE];
    int mergeArray[SIZE];

    copy(begin(originalArray), end(originalArray), begin(selectionArray));
    copy(begin(originalArray), end(originalArray), begin(bubbleArray));
    copy(begin(originalArray), end(originalArray), begin(insertionArray));
    copy(begin(originalArray), end(originalArray), begin(mergeArray));

    selectionSort(selectionArray, SIZE);
    cout << "\nSelection Sort:\n";
    printArray(selectionArray, SIZE);

    bubbleSort(bubbleArray, SIZE);
    cout << "\nBubble Sort:\n";
    printArray(bubbleArray, SIZE);

    insertionSort(insertionArray, SIZE);
    cout << "\nInsertion Sort:\n";
    printArray(insertionArray, SIZE);

    mergeSort(mergeArray, 0, SIZE - 1);
    cout << "\nMerge Sort:\n";
    printArray(mergeArray, SIZE);

    return 0;
}
```

Header File:

Ilo_A1.cpp | [*] sorting_algorithms.h

```cpp
#ifndef SORTING_ALGORITHMS_H
#define SORTING_ALGORITHMS_H

#include <iostream>
using namespace std;

void selectionSort(int arr[], int size) {
    for(int i = 0; i < size - 1; ++i) {
        int minIndex = i;
        for(int j = i + 1; j < size; ++j) {
            if(arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        swap(arr[i], arr[minIndex]);
    }
}

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

void insertionSort(int arr[], int size) {
    for(int i = 1; i < size; ++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;
    }
}
```

```cpp

void merge(int arr[], int left, int mid, int right) {
    int size1 = mid - left + 1;
    int size2 = right - mid;

    int* L = new int[size1];
    int* R = new int[size2];

    for(int i = 0; i < size1; ++i)
        L[i] = arr[left + i];
    for(int j = 0; j < size2; ++j)
        R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;
    while(i < size1 && j < size2) {
        if(L[i] <= R[j]) {
            arr[k++] = L[i++];
        } else {
            arr[k++] = R[j++];
        }
    }

    while(i < size1)
        arr[k++] = L[i++];
    while(j < size2)
        arr[k++] = R[j++];

    delete[] L;
    delete[] R;
}

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

void printArray(const int arr[], int size) {
    for(int i = 0; i < size; ++i) {
        cout << arr[i] << (i < size - 1 ? ", " : "\n");
    }
}

#endif

```

Output:

```
Original Array:
803, 100, 981, 251, 484, 383, 392, 494, 228, 711, 318, 649, 387, 857, 304, 761, 864, 523, 639, 188, 459, 3, 664, 153, 139, 934, 78
5, 728, 585, 55, 739, 828, 981, 328, 184, 72, 776, 429, 246, 313, 242, 255, 21, 10, 788, 79, 182, 397, 608, 548, 341, 921, 96, 686
, 432, 831, 685, 156, 176, 938, 777, 435, 951, 414, 94, 295, 610, 583, 904, 399, 642, 109, 985, 244, 30, 55, 92, 275, 651, 102, 86
0, 772, 446, 979, 685, 657, 441, 260, 971, 962, 708, 35, 51, 32, 350, 223, 878, 681, 109, 823

Selection Sort:
3, 10, 21, 30, 32, 35, 51, 55, 55, 72, 79, 92, 94, 96, 100, 102, 109, 109, 139, 153, 156, 176, 182, 184, 188, 223, 228, 242, 244,
246, 251, 255, 260, 275, 295, 304, 313, 318, 328, 341, 350, 383, 387, 392, 397, 399, 414, 429, 432, 435, 441, 446, 459, 484, 494,
523, 548, 583, 585, 608, 610, 639, 642, 649, 651, 657, 664, 681, 685, 685, 686, 708, 711, 728, 739, 761, 772, 776, 777, 785, 788,
803, 823, 828, 831, 857, 860, 864, 878, 904, 921, 934, 938, 951, 962, 971, 979, 981, 981, 985

Bubble Sort:
3, 10, 21, 30, 32, 35, 51, 55, 55, 72, 79, 92, 94, 96, 100, 102, 109, 109, 139, 153, 156, 176, 182, 184, 188, 223, 228, 242, 244,
246, 251, 255, 260, 275, 295, 304, 313, 318, 328, 341, 350, 383, 387, 392, 397, 399, 414, 429, 432, 435, 441, 446, 459, 484, 494,
523, 548, 583, 585, 608, 610, 639, 642, 649, 651, 657, 664, 681, 685, 685, 686, 708, 711, 728, 739, 761, 772, 776, 777, 785, 788,
803, 823, 828, 831, 857, 860, 864, 878, 904, 921, 934, 938, 951, 962, 971, 979, 981, 981, 985

Insertion Sort:
3, 10, 21, 30, 32, 35, 51, 55, 55, 72, 79, 92, 94, 96, 100, 102, 109, 109, 139, 153, 156, 176, 182, 184, 188, 223, 228, 242, 244,
246, 251, 255, 260, 275, 295, 304, 313, 318, 328, 341, 350, 383, 387, 392, 397, 399, 414, 429, 432, 435, 441, 446, 459, 484, 494,
523, 548, 583, 585, 608, 610, 639, 642, 649, 651, 657, 664, 681, 685, 685, 686, 708, 711, 728, 739, 761, 772, 776, 777, 785, 788,
803, 823, 828, 831, 857, 860, 864, 878, 904, 921, 934, 938, 951, 962, 971, 979, 981, 981, 985

Merge Sort:
3, 10, 21, 30, 32, 35, 51, 55, 55, 72, 79, 92, 94, 96, 100, 102, 109, 109, 139, 153, 156, 176, 182, 184, 188, 223, 228, 242, 244,
246, 251, 255, 260, 275, 295, 304, 313, 318, 328, 341, 350, 383, 387, 392, 397, 399, 414, 429, 432, 435, 441, 446, 459, 484, 494,
523, 548, 583, 585, 608, 610, 639, 642, 649, 651, 657, 664, 681, 685, 685, 686, 708, 711, 728, 739, 761, 772, 776, 777, 785, 788,
803, 823, 828, 831, 857, 860, 864, 878, 904, 921, 934, 938, 951, 962, 971, 979, 981, 981, 985

--------------------------------
Process exited after 0.1482 seconds with return value 0
Press any key to continue . . .
```

Explanation:

This program generation an array of 100 random integers and uses four sorting algorithm like selection sort, bubble sort, insertion, sort and merge sort.

Table 7_2:

Main CPP:

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

using namespace std;

void printArray(const int arr[], int size) {
    for (int i = 0; i < size; ++i) {
        cout << arr[i] << (i < size - 1 ? ", " : "\n");
    }
}

int main() {
    const int SIZE = 10;
    int arr[SIZE];

    srand(static_cast<unsigned>(time(0)));
    for (int i = 0; i < SIZE; ++i) {
        arr[i] = rand() % 100;
    }

    cout << "Original Array:\n";
    printArray(arr, SIZE);

    bubbleSort(arr, SIZE);

    cout << "\nSorted Array (Descending - Bubble Sort):\n";
    printArray(arr, SIZE);

    return 0;
}

```

Header File:

```cpp
#ifndef BUBBLE_SORT_H
#define BUBBLE_SORT_H

#include <iostream>
#include <algorithm>

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]);
            }
        }
    }
}

#endif
```

Output:

```
Original Array:
85, 95, 5, 96, 6, 8, 11, 0, 30, 53

Sorted Array (Descending - Bubble Sort):
0, 5, 6, 8, 11, 30, 53, 85, 95, 96


--------------------------------
Process exited after 0.1019 seconds with return value 0
Press any key to continue . . .
```

Explanation:

The program an array of 10 random integer and sorts it in descending elements order. This algorithm works by comparing each element with the ones after it and swapping whenever a larger element is found

Table 7-3:

Main CPP:

Main selection.cpp | selection_sort.h

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

using namespace std;

void printArray(const int arr[], int size) {
    for (int i = 0; i < size; ++i) {
        cout << arr[i] << (i < size - 1 ? ", " : "\n");
    }
}

int main() {
    const int SIZE = 10;
    int arr[SIZE];

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

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

    cout << "Original Array:\n";
    printArray(arr, SIZE);

    selectionSort(arr, SIZE);

    cout << "\nSorted Array (Ascending - Selection Sort):\n";
    printArray(arr, SIZE);

    return 0;
}

```

Header File:

```cpp
#ifndef SELECTION_SORT_H
#define SELECTION_SORT_H

#include <iostream>
using namespace std;

template <typename T>
int Routine_Smallest(T A[], int K, const int arrSize) {
    int position = K;
    T smallestElem = A[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++;
    }
}

#endif

```

Output:

```
Original Array:
25, 21, 73, 29, 51, 45, 51, 52, 28, 2

Sorted Array (Ascending - Selection Sort):
2, 21, 25, 28, 29, 45, 51, 51, 52, 73

--------------------------------
Process exited after 0.1006 seconds with return value 0
Press any key to continue . . .
```

Explanation:

This program sorts an array using the selection algorithm, where the smallest elements order is found in each pass using a helper function and swapped into its correct position.

Table 7-4:

Main CPP:

Main insertion.cpp | insertion_sort.h

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

using namespace std;

void printArray(const int arr[], int size) {
    for (int i = 0; i < size; ++i) {
        cout << arr[i] << (i < size - 1 ? ", " : "\n");
    }
}

int main() {
    const int SIZE = 10;
    int arr[SIZE];

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

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

    cout << "Original Array:\n";
    printArray(arr, SIZE);

    insertionSort(arr, SIZE);

    cout << "\nSorted Array (Ascending - Insertion Sort):\n";
    printArray(arr, SIZE);

    return 0;
}
```

Header File:

Main insertion.cpp | insertion_sort.h

```cpp
#ifndef INSERTION_SORT_H
#define INSERTION_SORT_H

#include <iostream>
using namespace std;

template <typename T>
void insertionSort(T arr[], const int N) {
    int K = 1, J;
    T temp;

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

        while (J >= 0 && temp <= arr[J]) {
            arr[J + 1] = arr[J];
            J--;
        }

        arr[J + 1] = temp;
        K++;
    }
}

#endif
```

Output:

```
55, 86, 24, 64, 82, 38, 15, 22, 39, 90

Sorted Array (Ascending - Insertion Sort):
15, 22, 24, 38, 39, 55, 64, 82, 86, 90

--------------------------------
Process exited after 0.1003 seconds with return value 0
Press any key to continue . . .
```

Explanation:

This program sorts an array using the insertion sort algorithm. Starting from the 2nd element, and it shifts to larger elements to the right and inserts the current elements in its proper position.

7. Supplementary Activity:

Output Console Showing Sorted Array	Manual Count	Count Result of Algorithm
Sorted Votes: **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 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 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5**	Vote Count: Kerwin: 21 votes Angelo: 23 votes Gabriel: 17 votes Michael: 20 votes Dom: 19 votes	Vote Count: Kerwin: 21 votes Angelo: 23 votes Gabriel: 17 votes Michael: 20 votes Dom: 19 votes Winner is Candidate 2 with 23 votes!

8. Conclusion:

In this laboratory activity we implemented the four basic algorithms, Bubble, Selection, Insertion and Merge Sorting. Each sorting method was tested on the same array of 100 and it generated 100 randomize values on how they work. This programs show how sorting techniques can be applied in real world scenarios like organizing data.

9. Assessment Rubric