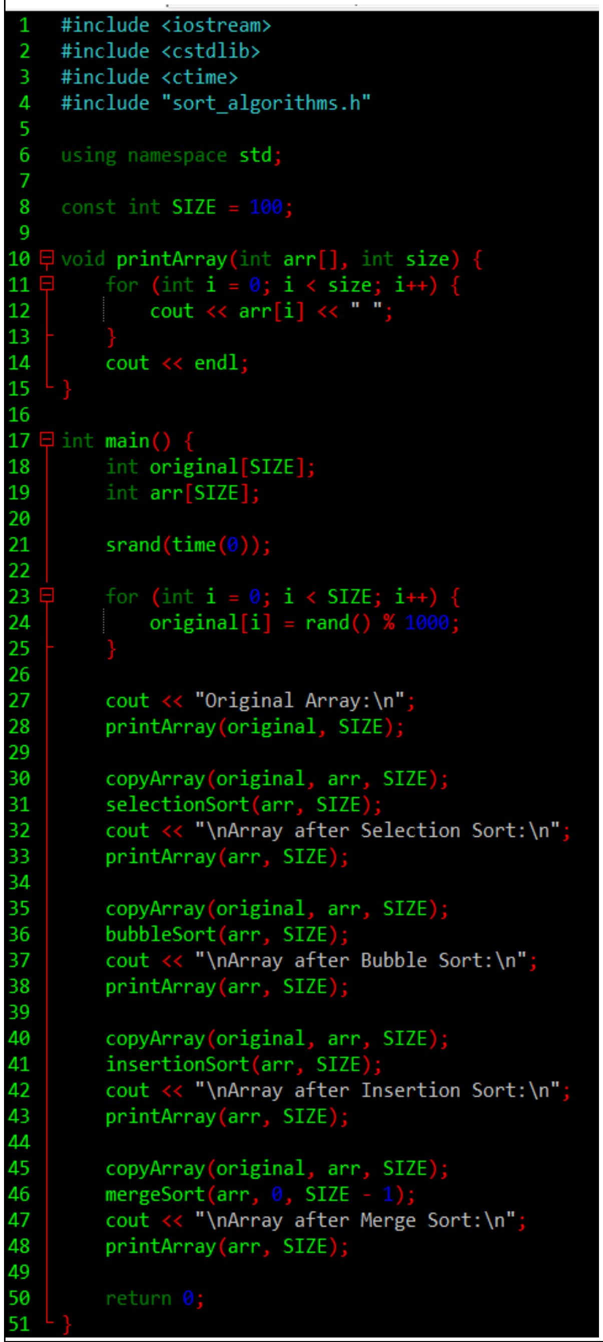
Activity No. <7>	
SORTING ALGORITHMS: BUBBLE, SELECTION, AND INSERTION SORT	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 9/18/25**
Section: CPE21S4	**Date Submitted: 9/18/25**
Name(s): Alcantara, Jason P.	**Instructor: Engr. Jimlord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

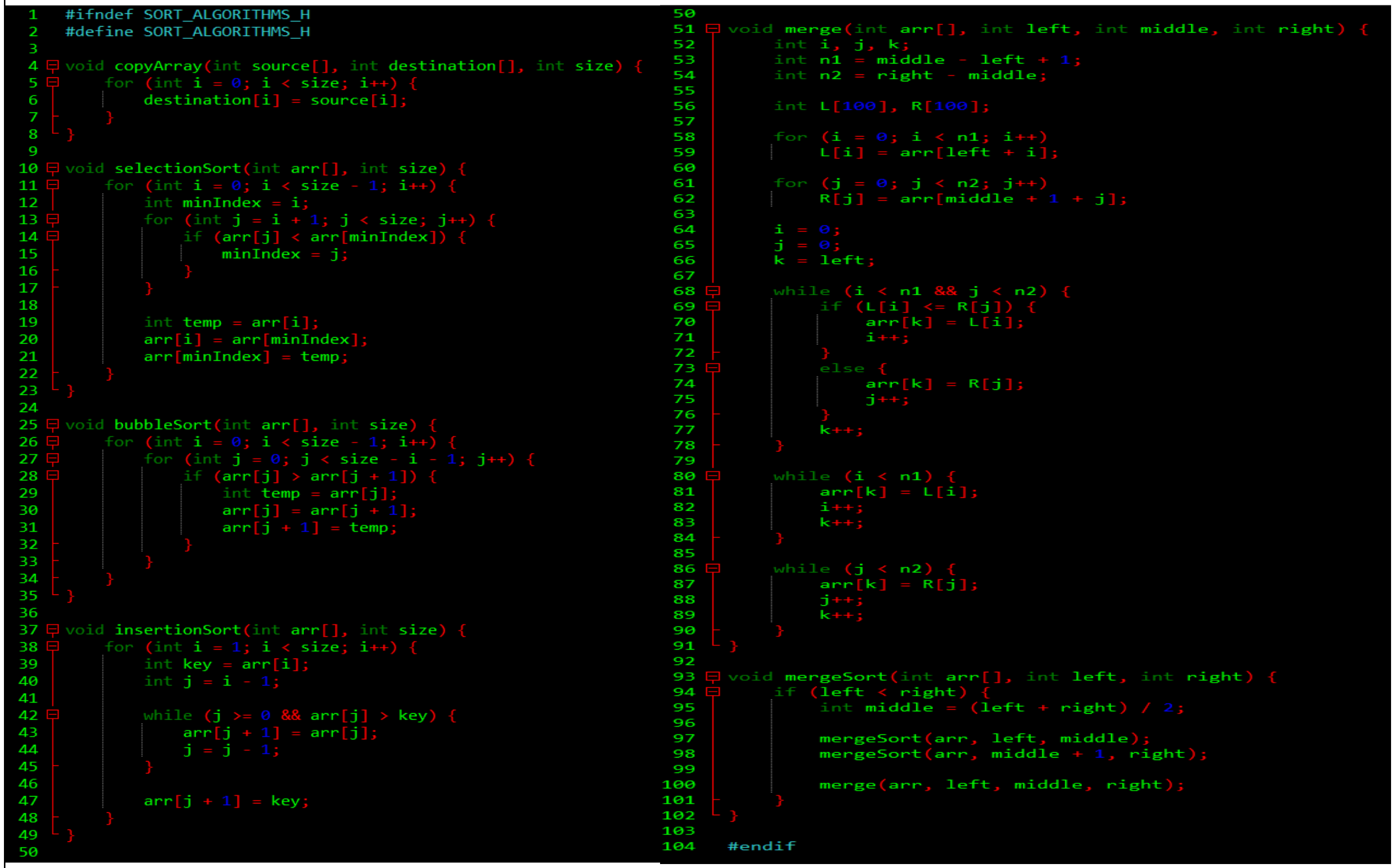
6. Output:

Table 7-1:
Main CPP:



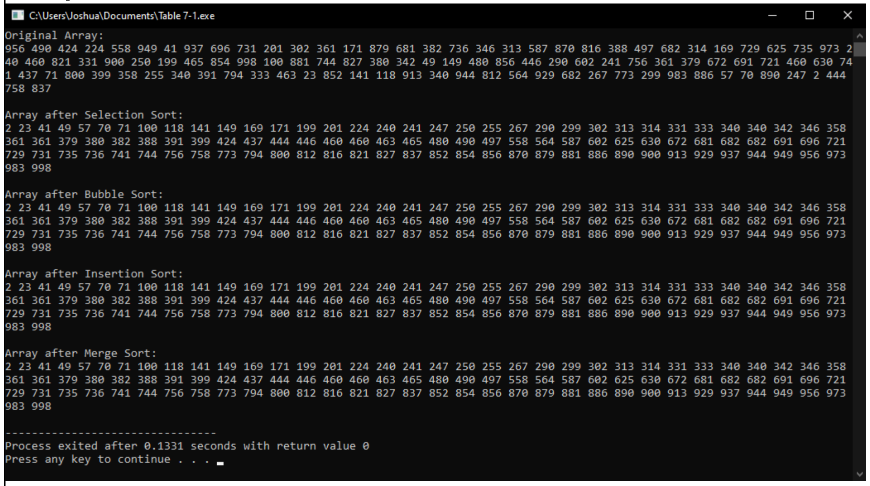
```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "sort_algorithms.h"

using namespace std;

const int SIZE = 100;

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int original[SIZE];
    int arr[SIZE];

    srand(time(0));

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

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

    copyArray(original, arr, SIZE);
    selectionSort(arr, SIZE);
    cout << "\nArray after Selection Sort:\n";
    printArray(arr, SIZE);

    copyArray(original, arr, SIZE);
    bubbleSort(arr, SIZE);
    cout << "\nArray after Bubble Sort:\n";
    printArray(arr, SIZE);

    copyArray(original, arr, SIZE);
    insertionSort(arr, SIZE);
    cout << "\nArray after Insertion Sort:\n";
    printArray(arr, SIZE);

    copyArray(original, arr, SIZE);
    mergeSort(arr, 0, SIZE - 1);
    cout << "\nArray after Merge Sort:\n";
    printArray(arr, SIZE);

    return 0;
}
```

Header File:



```cpp
#ifndef SORT_ALGORITHMS_H
#define SORT_ALGORITHMS_H

void copyArray(int source[], int destination[], int size) {
    for (int i = 0; i < size; i++) {
        destination[i] = source[i];
    }
}

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;
            }
        }

        int temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
}

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]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

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 = j - 1;
        }

        arr[j + 1] = key;
    }
}


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

    int L[100], R[100];

    for (i = 0; i < n1; i++)
        L[i] = arr[left + i];

    for (j = 0; j < n2; j++)
        R[j] = arr[middle + 1 + j];

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

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

        merge(arr, left, middle, right);
    }
}

#endif
```

Output:



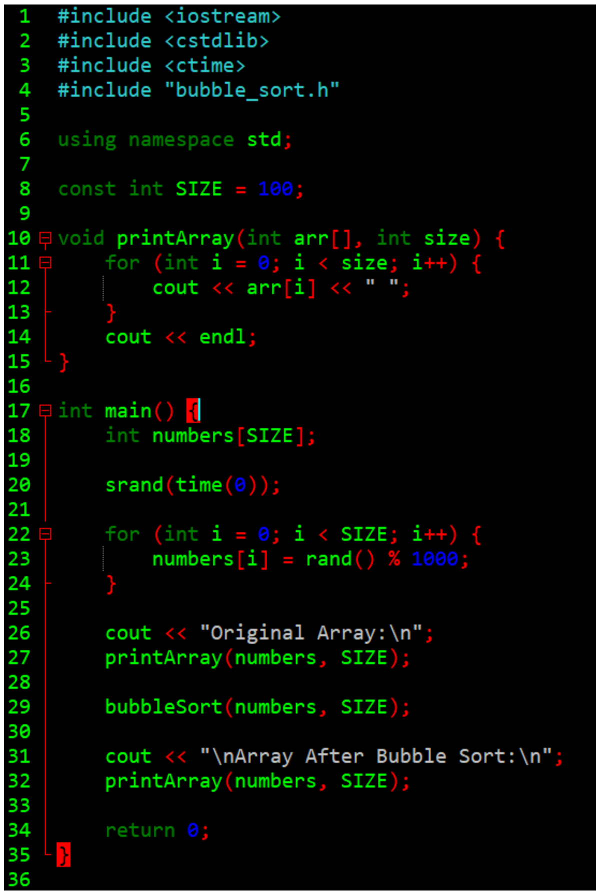
```
C:\Users\Joshua\Documents\Table 7-1.exe
Original Array:
956 490 424 224 558 949 41 937 696 731 201 302 361 171 879 681 382 736 346 313 587 870 816 388 497 682 314 169 729 625 735 973 2
40 460 821 331 900 250 199 465 854 998 100 881 744 827 380 342 49 149 480 856 446 290 602 241 756 361 379 672 691 721 460 630 74
1 437 71 800 399 358 255 340 391 794 333 463 23 852 141 118 913 340 944 812 564 929 682 267 773 299 983 886 57 70 890 247 2 444
758 837

Array after Selection Sort:
2 23 41 49 57 70 71 100 118 141 149 169 171 199 201 224 240 241 247 250 255 267 290 299 302 313 314 331 333 340 340 342 346 358
361 361 379 380 382 388 391 399 424 437 444 446 460 460 463 465 480 490 497 558 564 587 602 625 630 672 681 682 682 691 696 721
729 731 735 736 741 744 756 758 773 794 800 812 816 821 827 837 852 854 856 870 879 881 886 890 900 913 929 937 944 949 956 973
983 998

Array after Bubble Sort:
2 23 41 49 57 70 71 100 118 141 149 169 171 199 201 224 240 241 247 250 255 267 290 299 302 313 314 331 333 340 340 342 346 358
361 361 379 380 382 388 391 399 424 437 444 446 460 460 463 465 480 490 497 558 564 587 602 625 630 672 681 682 682 691 696 721
729 731 735 736 741 744 756 758 773 794 800 812 816 821 827 837 852 854 856 870 879 881 886 890 900 913 929 937 944 949 956 973
983 998

Array after Insertion Sort:
2 23 41 49 57 70 71 100 118 141 149 169 171 199 201 224 240 241 247 250 255 267 290 299 302 313 314 331 333 340 340 342 346 358
361 361 379 380 382 388 391 399 424 437 444 446 460 460 463 465 480 490 497 558 564 587 602 625 630 672 681 682 682 691 696 721
729 731 735 736 741 744 756 758 773 794 800 812 816 821 827 837 852 854 856 870 879 881 886 890 900 913 929 937 944 949 956 973
983 998

Array after Merge Sort:
2 23 41 49 57 70 71 100 118 141 149 169 171 199 201 224 240 241 247 250 255 267 290 299 302 313 314 331 333 340 340 342 346 358
361 361 379 380 382 388 391 399 424 437 444 446 460 460 463 465 480 490 497 558 564 587 602 625 630 672 681 682 682 691 696 721
729 731 735 736 741 744 756 758 773 794 800 812 816 821 827 837 852 854 856 870 879 881 886 890 900 913 929 937 944 949 956 973
983 998

--------------------------------
Process exited after 0.1331 seconds with return value 0
Press any key to continue . . .
```

Explanation:

In this code gives a practical demonstrate of four element sorts that can be easily understood by beginners. These algorithms are broken down into simple function, loops, or arrays. The logic has been decoupled into a main file and header.

Table 7-2:

Main CPP:



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

using namespace std;

const int SIZE = 100;

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int numbers[SIZE];

    srand(time(0));

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

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

    bubbleSort(numbers, SIZE);

    cout << "\nArray After Bubble Sort:\n";
    printArray(numbers, SIZE);

    return 0;
}

```

Header File:

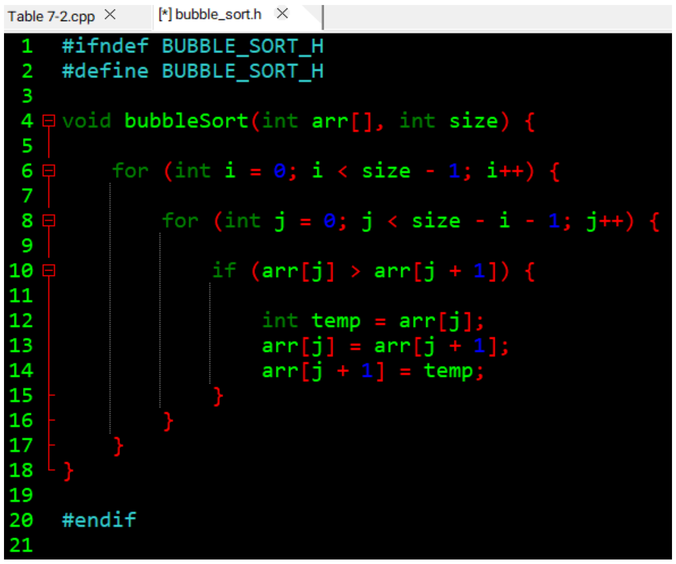
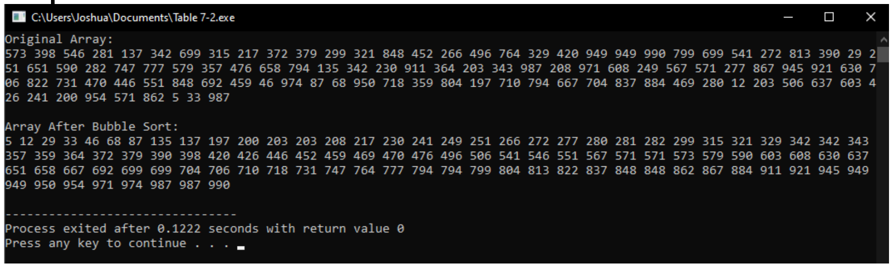


Table 7-2.cpp | [*] bubble_sort.h

```cpp
#ifndef BUBBLE_SORT_H
#define BUBBLE_SORT_H

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]) {

                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

#endif

```

Output:



C:\Users\Joshua\Documents\Table 7-2.exe

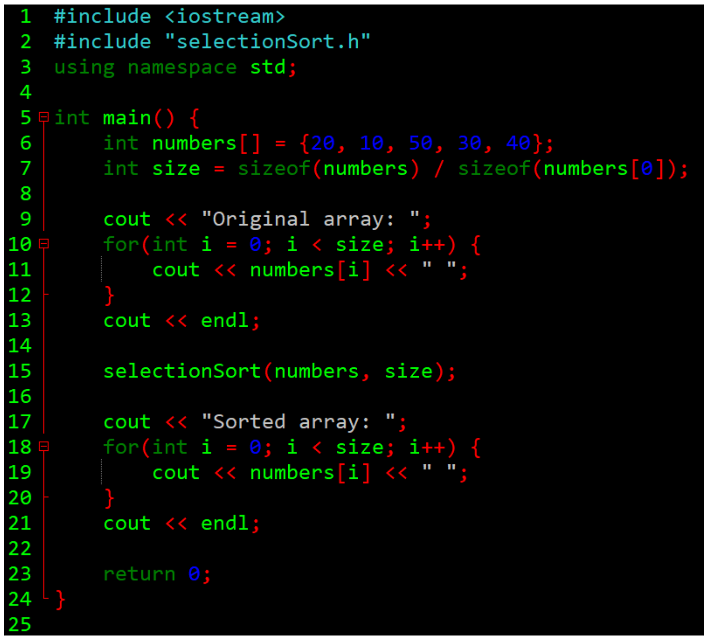
```
Original Array:
573 398 546 281 137 342 699 315 217 372 379 299 321 848 452 266 496 764 329 420 949 949 990 799 699 541 272 813 390 29 2
51 651 590 282 747 777 579 357 476 658 794 135 342 230 911 364 203 343 987 208 971 608 249 567 571 277 867 945 921 630 7
06 822 731 470 446 551 848 692 459 46 974 87 68 950 718 359 804 197 710 794 667 704 837 884 469 280 12 203 506 637 603 4
26 241 200 954 571 862 5 33 987

Array After Bubble Sort:
5 12 29 33 46 68 87 135 137 197 200 203 203 208 217 230 241 249 251 266 272 277 280 281 282 299 315 321 329 342 342 343
357 359 364 372 379 390 398 420 426 446 452 459 469 470 476 496 506 541 546 551 567 571 571 573 579 590 603 608 630 637
651 658 667 692 699 699 704 706 710 718 731 747 764 777 794 794 799 804 813 822 837 848 848 862 867 884 911 921 945 949
949 950 954 971 974 987 987 990

--------------------------------
Process exited after 0.1222 seconds with return value 0
Press any key to continue . . .
```

Explanation?

In this algorithm brings its largest number to the back, so it checks adjacent elements and swaps them if they are not in the correct order of elements and all this is done in a bubble sorts.

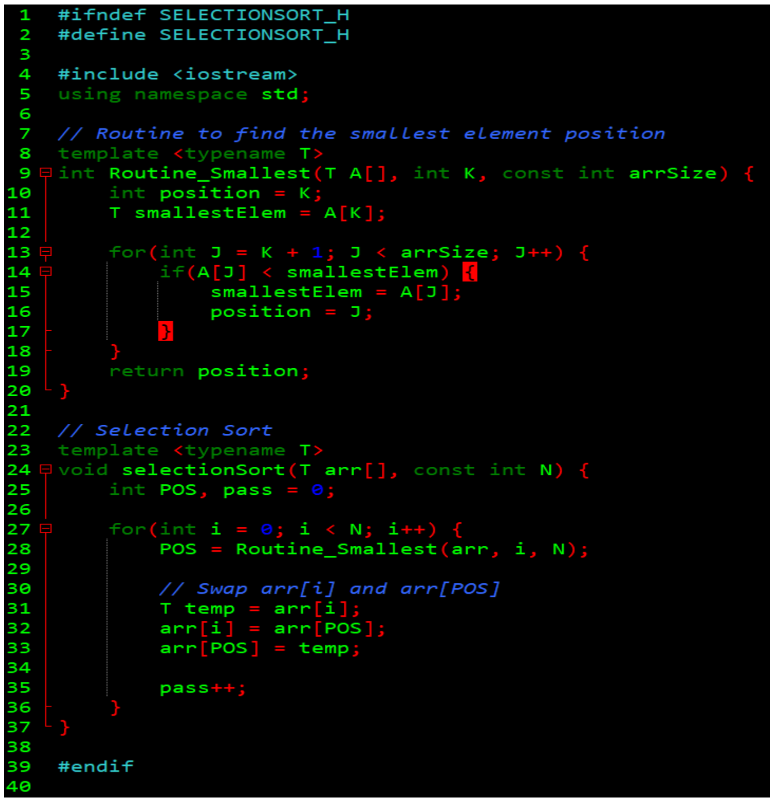
Table 7-3:

Main CPP:



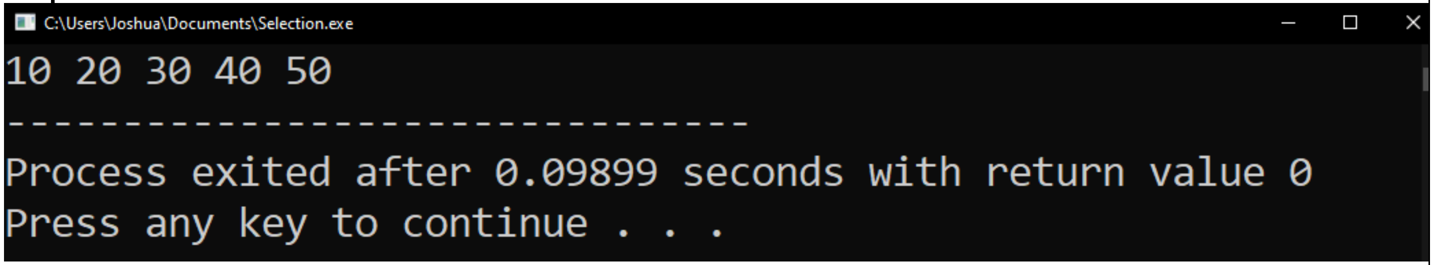
```cpp
#include <iostream>
#include "selectionSort.h"
using namespace std;

int main() {
    int numbers[] = {20, 10, 50, 30, 40};
    int size = sizeof(numbers) / sizeof(numbers[0]);

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

    selectionSort(numbers, size);

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

    return 0;
}

```

Header File:



```cpp
#ifndef SELECTIONSORT_H
#define SELECTIONSORT_H

#include <iostream>
using namespace std;

// Routine to find the smallest element position
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;
}

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

    for(int i = 0; i < N; i++) {
        POS = Routine_Smallest(arr, i, N);

        // Swap arr[i] and arr[POS]
        T temp = arr[i];
        arr[i] = arr[POS];
        arr[POS] = temp;

        pass++;
    }
}

#endif

```

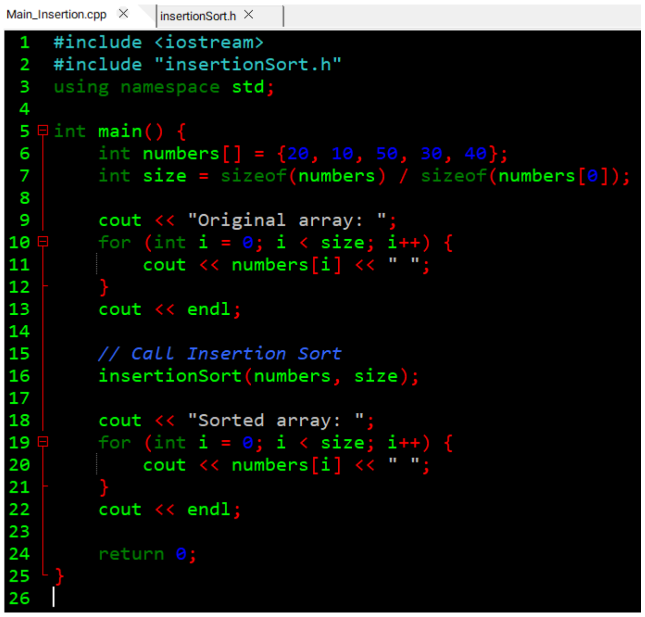
Output:



```
C:\Users\Joshua\Documents\Selection.exe
10 20 30 40 50
--------------------------------
Process exited after 0.09899 seconds with return value 0
Press any key to continue . . .
```

Explanation:
In this program sorts of numbers using method of selection sort. It inspect each position in the array, locating the smallest number to highest number.

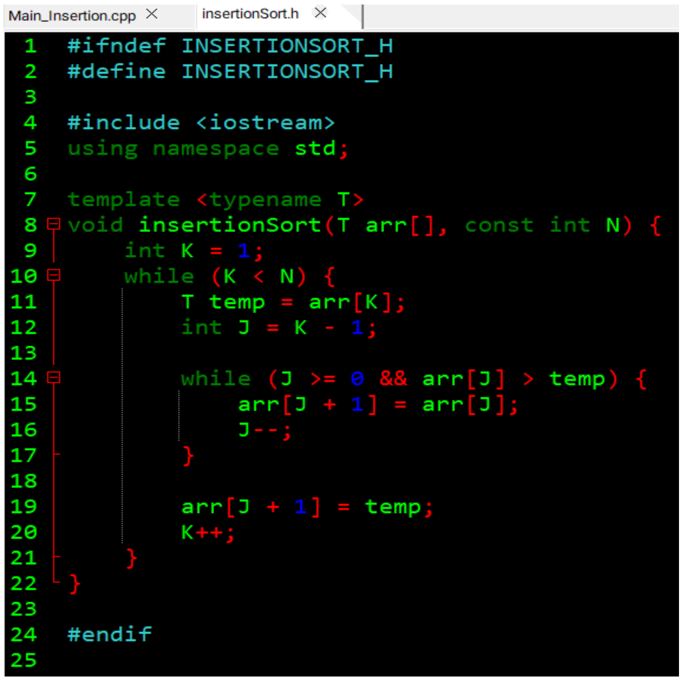
Table 7-4:
Main CPP:



Main_Insertion.cpp insertionSort.h

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

int main() {
    int numbers[] = {20, 10, 50, 30, 40};
    int size = sizeof(numbers) / sizeof(numbers[0]);

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

    // Call Insertion Sort
    insertionSort(numbers, size);

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

    return 0;
}
```

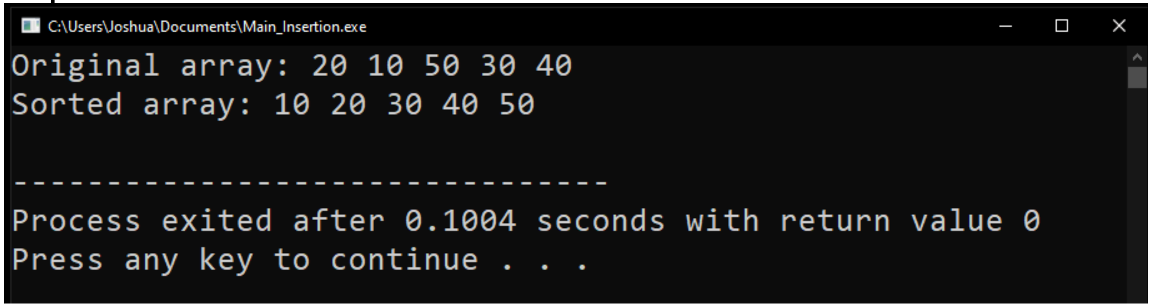
Header File:



Main_Insertion.cpp insertionSort.h

```cpp
#ifndef INSERTIONSORT_H
#define INSERTIONSORT_H

#include <iostream>
using namespace std;

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

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

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

#endif
```

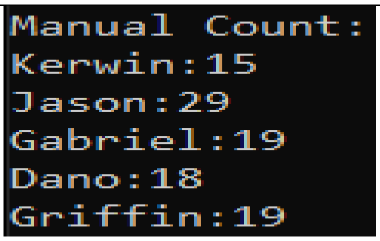
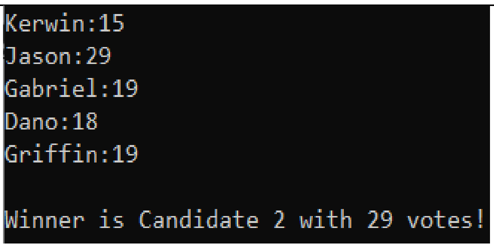
Output:



```
C:\Users\Joshua\Documents\Main_Insertion.exe
Original array: 20 10 50 30 40
Sorted array: 10 20 30 40 50

--------------------------------
Process exited after 0.1004 seconds with return value 0
Press any key to continue . . .
```

Explanation:
Insertion sort takes one number at a time and puts it in the correct position among the already sorted numbers.

7. Supplementary Activity:

Output Console Showing Sorted Array	Manual Count	Count Result of Algorithm
1 1 1 1 1 1 1 1 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 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 3 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**	 Manual Count: Kerwin:15 Jason:29 Gabriel:19 Dano:18 Griffin:19	 Kerwin:15 Jason:29 Gabriel:19 Dano:18 Griffin:19 Winner is Candidate 2 with 29 votes!

8. Conclusion:
So in this laboratory activity we implemented bubble, selection and insertion sort successfully organized a set of 100 random integers in ascending orders. These algorithms are effective for small datasets and help to create a understandable of sorting data elements, they are normally inefficient for lager data inputs compared to the advanced algorithm like merge sort.

9. Assessment Rubric