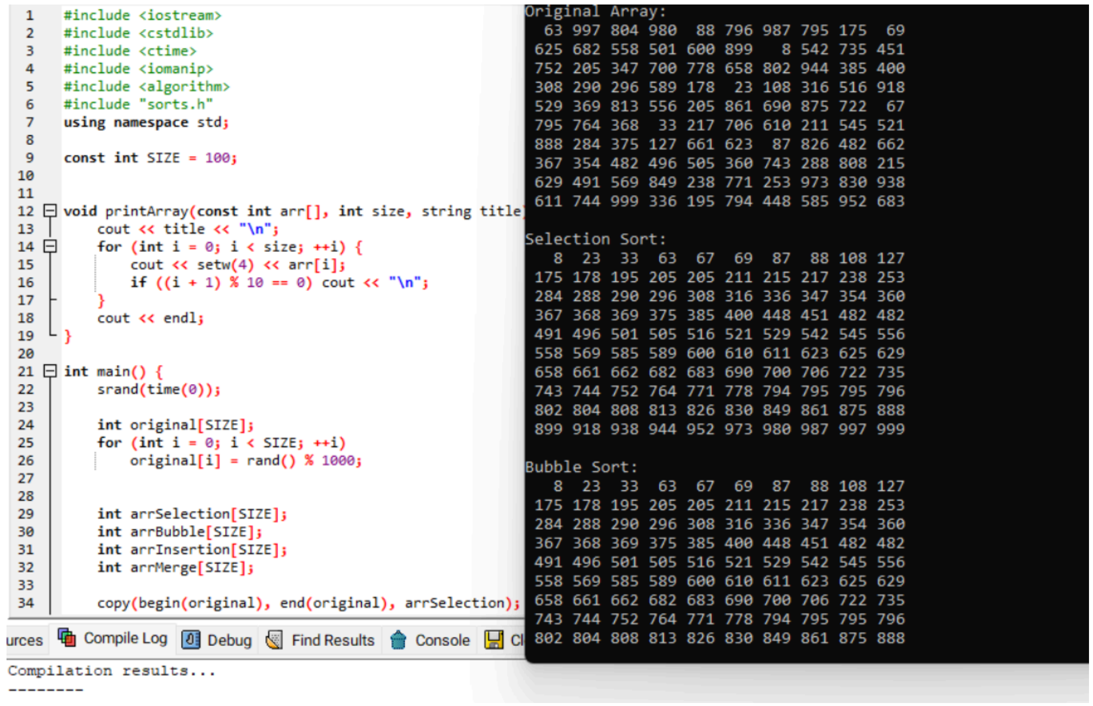
Hands-on Activity 7.1	
Sorting Algorithms Pt1	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 9/17/25
Section: CPE21S4	**Date Submitted:** 9/18/25
Name(s): Avila, Vince Gabriel V.	**Instructor:** Engr. Jimlord Quejado
6. Output	

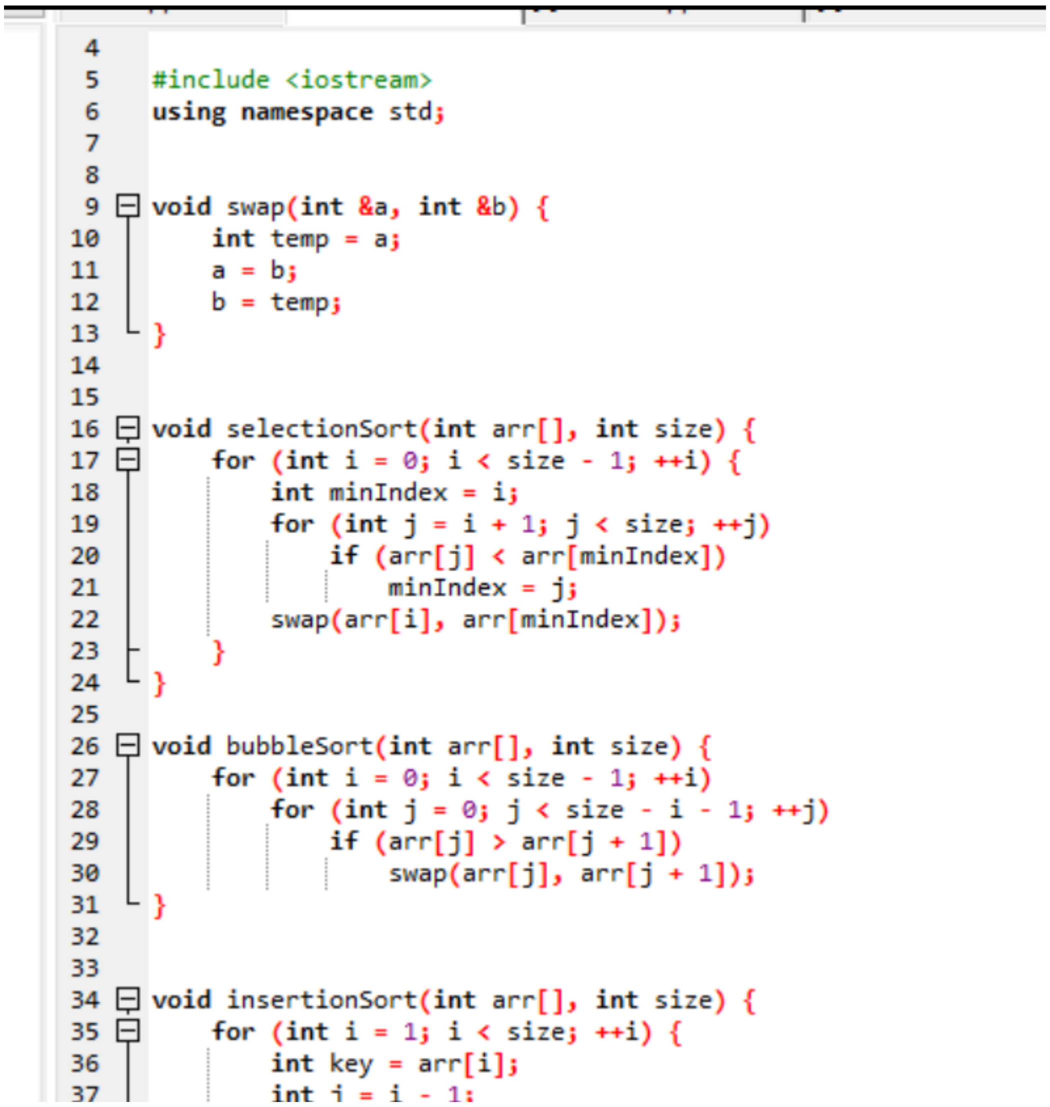
ILO A:



```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <iomanip>
#include <algorithm>
#include "sorts.h"
using namespace std;

const int SIZE = 100;


void printArray(const int arr[], int size, string title
    cout << title << "\n";
    for (int i = 0; i < size; ++i) {
        cout << setw(4) << arr[i];
        if ((i + 1) % 10 == 0) cout << "\n";
    }
    cout << endl;
}

int main() {
    srand(time(0));

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


    int arrSelection[SIZE];
    int arrBubble[SIZE];
    int arrInsertion[SIZE];
    int arrMerge[SIZE];

    copy(begin(original), end(original), arrSelection);
```

```
Compilation results...
--------
```

```
Original Array:
  63 997 804 980  88 796 987 795 175  69
 625 682 558 501 600 899   8 542 735 451
 752 205 347 700 778 658 802 944 385 400
 308 290 296 589 178  23 108 316 516 918
 529 369 813 556 205 861 690 875 722  67
 795 764 368  33 217 706 610 211 545 521
 888 284 375 127 661 623  87 826 482 662
 367 354 482 496 505 360 743 288 808 215
 629 491 569 849 238 771 253 973 830 938
 611 744 999 336 195 794 448 585 952 683

Selection Sort:
   8  23  33  63  67  69  87  88 108 127
 175 178 195 205 205 211 215 217 238 253
 284 288 290 296 308 316 336 347 354 360
 367 368 369 375 385 400 448 451 482 482
 491 496 501 505 516 521 529 542 545 556
 558 569 585 589 600 610 611 623 625 629
 658 661 662 682 683 690 700 706 722 735
 743 744 752 764 771 778 794 795 795 796
 802 804 808 813 826 830 849 861 875 888
 899 918 938 944 952 973 980 987 997 999

Bubble Sort:
   8  23  33  63  67  69  87  88 108 127
 175 178 195 205 205 211 215 217 238 253
 284 288 290 296 308 316 336 347 354 360
 367 368 369 375 385 400 448 451 482 482
 491 496 501 505 516 521 529 542 545 556
 558 569 585 589 600 610 611 623 625 629
 658 661 662 682 683 690 700 706 722 735
 743 744 752 764 771 778 794 795 795 796
 802 804 808 813 826 830 849 861 875 888
```

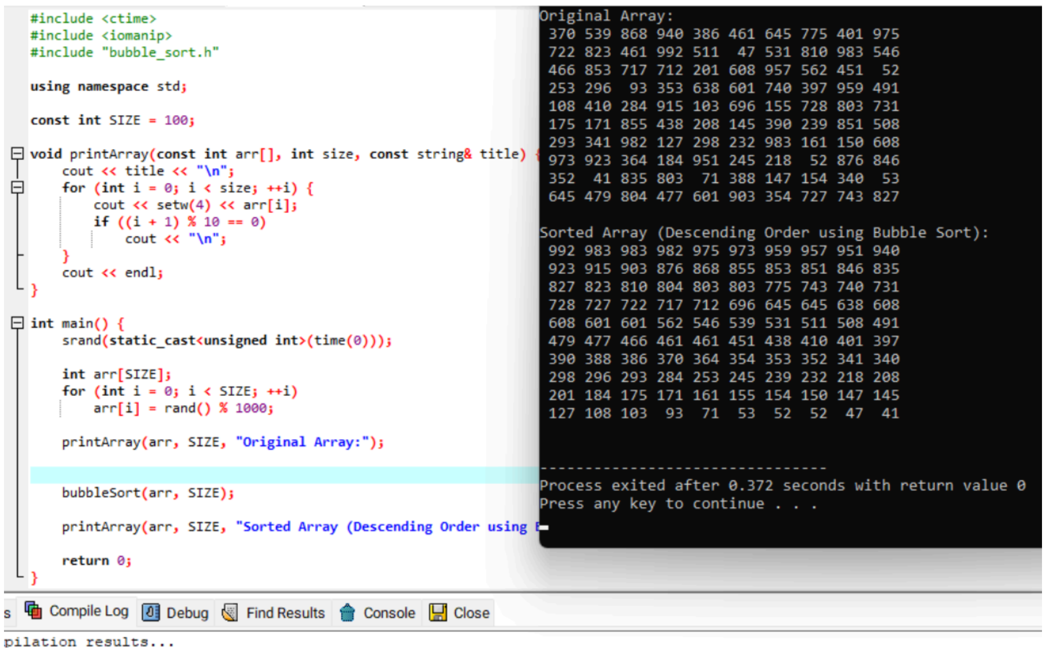


```cpp

#include <iostream>
using namespace std;


void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}


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;
```

ANALYSIS:

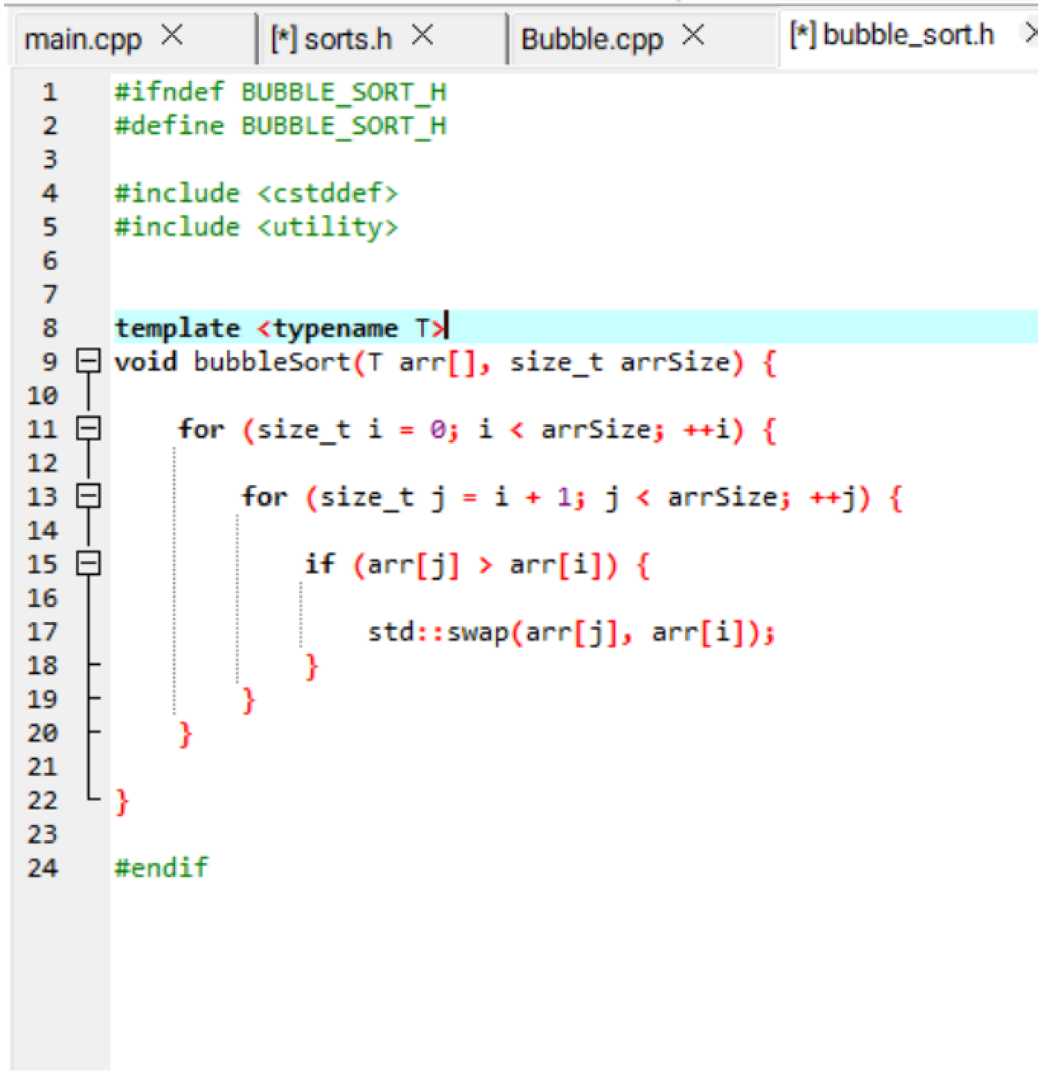
My code sorts the same random list of numbers using four different methods and shows the results. To improve it, I could organize the code better, make the merge sort faster with less memory, and check how long each sort takes.

BUBBLE SORT:



```cpp
#include <ctime>
#include <iomanip>
#include "bubble_sort.h"

using namespace std;

const int SIZE = 100;

void printArray(const int arr[], int size, const string& title) {
    cout << title << "\n";
    for (int i = 0; i < size; ++i) {
        cout << setw(4) << arr[i];
        if ((i + 1) % 10 == 0)
            cout << "\n";
    }
    cout << endl;
}

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

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

    printArray(arr, SIZE, "Original Array:");


    bubbleSort(arr, SIZE);

    printArray(arr, SIZE, "Sorted Array (Descending Order using

    return 0;
}
```

```
Original Array:
 370 539 868 940 386 461 645 775 401 975
 722 823 461 992 511  47 531 810 983 546
 466 853 717 712 201 608 957 562 451  52
 253 296  93 353 638 601 740 397 959 491
 108 410 284 915 103 696 155 728 803 731
 175 171 855 438 208 145 390 239 851 508
 293 341 982 127 298 232 983 161 150 608
 973 923 364 184 951 245 218  52 876 846
 352  41 835 803  71 388 147 154 340  53
 645 479 804 477 601 903 354 727 743 827

Sorted Array (Descending Order using Bubble Sort):
 992 983 983 982 975 973 959 957 951 940
 923 915 903 876 868 855 853 851 846 835
 827 823 810 804 803 803 775 743 740 731
 728 727 722 717 712 696 645 645 638 608
 608 601 601 562 546 539 531 511 508 491
 479 477 466 461 461 451 438 410 401 397
 390 388 386 370 364 354 353 352 341 340
 298 296 293 284 253 245 239 232 218 208
 201 184 175 171 161 155 154 150 147 145
 127 108 103  93  71  53  52  52  47  41


--------------------------------
Process exited after 0.372 seconds with return value 0
Press any key to continue . . .
```

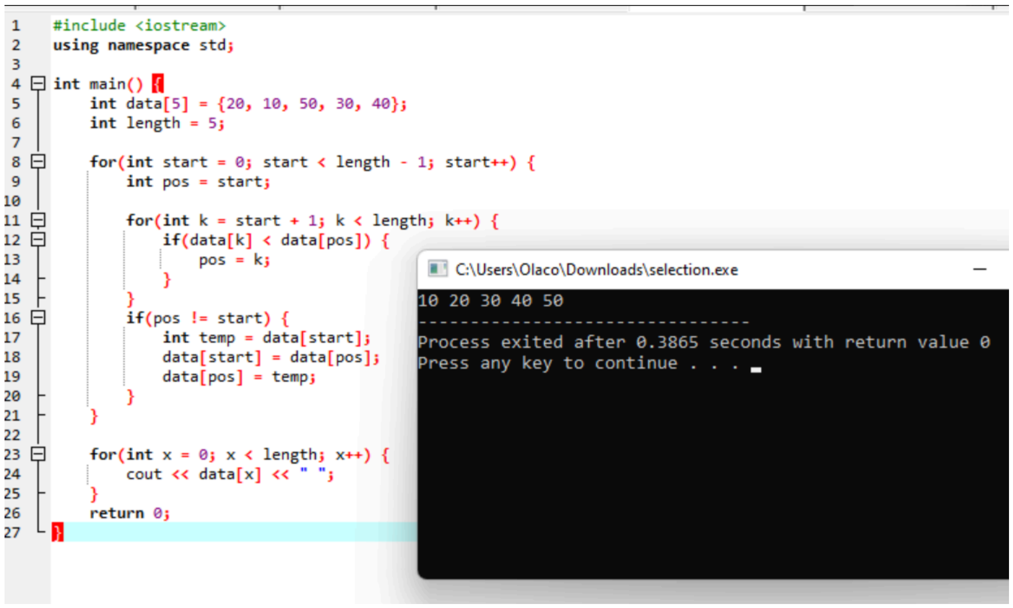


```cpp
#ifndef BUBBLE_SORT_H
#define BUBBLE_SORT_H

#include <cstddef>
#include <utility>


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
```

ANALYSIS:

My program creates a list of 100 random numbers and uses bubble sort to arrange them from largest to smallest. The code is simple and clear, and I used a template in the bubble sort to make it work with different data types if needed.

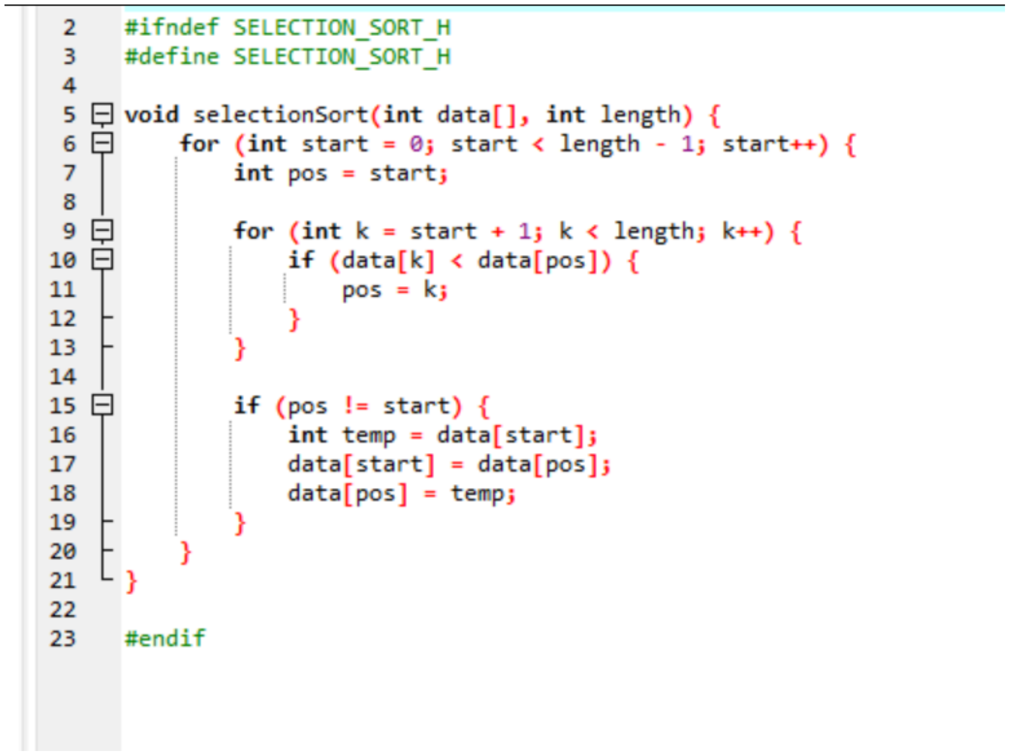
SELECTION SORT:



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

int main() {
    int data[5] = {20, 10, 50, 30, 40};
    int length = 5;

    for(int start = 0; start < length - 1; start++) {
        int pos = start;

        for(int k = start + 1; k < length; k++) {
            if(data[k] < data[pos]) {
                pos = k;
            }
        }
        if(pos != start) {
            int temp = data[start];
            data[start] = data[pos];
            data[pos] = temp;
        }
    }

    for(int x = 0; x < length; x++) {
        cout << data[x] << " ";
    }
    return 0;
}
```

C:\Users\Olaco\Downloads\selection.exe

```
10 20 30 40 50
--------------------------------
Process exited after 0.3865 seconds with return value 0
Press any key to continue . . .
```

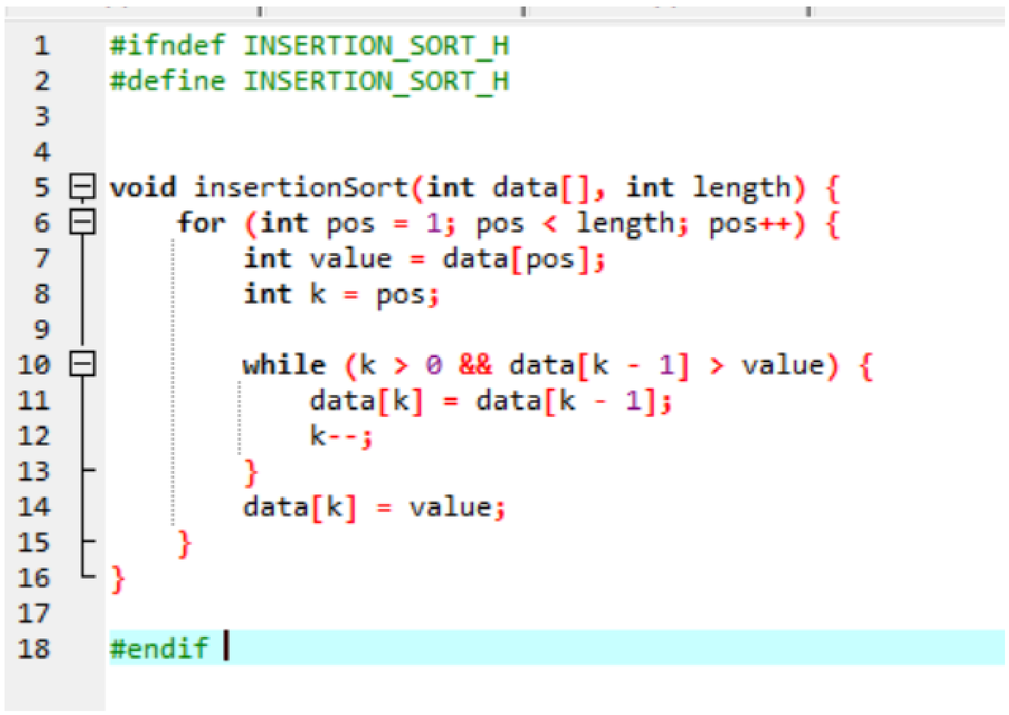


```cpp
#ifndef SELECTION_SORT_H
#define SELECTION_SORT_H

void selectionSort(int data[], int length) {
    for (int start = 0; start < length - 1; start++) {
        int pos = start;

        for (int k = start + 1; k < length; k++) {
            if (data[k] < data[pos]) {
                pos = k;
            }
        }

        if (pos != start) {
            int temp = data[start];
            data[start] = data[pos];
            data[pos] = temp;
        }
    }
}

#endif
```

ANALYSIS::

This program sorts an array of numbers in ascending order using selection sort, which finds the smallest value in the unsorted part and swaps it to the front step-by-step. The sorting logic is nicely separated into a reusable function in the header file, making the code organized and easy to use elsewhere.

INSERTION SORT:

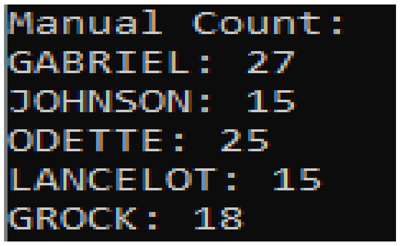
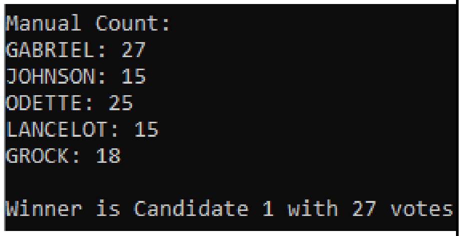


```
#ifndef INSERTION_SORT_H
#define INSERTION_SORT_H


void insertionSort(int data[], int length) {
    for (int pos = 1; pos < length; pos++) {
        int value = data[pos];
        int k = pos;

        while (k > 0 && data[k - 1] > value) {
            data[k] = data[k - 1];
            k--;
        }
        data[k] = value;
    }
}

#endif
```

ANALYSIS:

This program sorts an array using insertion sort, which works by taking each element and inserting it into its correct position in the sorted part of the array. The sorting logic is put in a separate header file, making the code clean and easy to reuse.

7. Supplementary Activity:

Output Console Showing Sorted Array	**Manual Count**	**Count Result of Algorithm**
Sorted Array: 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 3 3 3 3 3 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 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	 Manual Count: GABRIEL: 27 JOHNSON: 15 ODETTE: 25 LANCELOT: 15 GROCK: 18	 Manual Count: GABRIEL: 27 JOHNSON: 15 ODETTE: 25 LANCELOT: 15 GROCK: 18 Winner is Candidate 1 with 27 votes

ANALYSIS:

This program creates a list of 100 random votes for 5 candidates, sorts the votes using insertion sort, and then counts how many votes each candidate got. Finally, it prints the sorted votes, the vote counts, and declares the winner with the most votes.

8. Conclusion:
In these laboratory activity we implemented the Bubble, Selection, Insertion, and Merge sorts which are done in C++. There were 100 randomize numbers which we used for testing all 4 sorts and we learned how each worked. There is also a voting program which uses Insertion Sort to organize 100 votes and then uses it for candidates vote counting and displaying the winner. These programs show how practical sorting algorithms are for organizing votes and counting them.

9. Assessment Rubric