Mystery Sort Pseudocode

```
mysterySort(array A)

for i = 1 to length(A) – 1

   key = A[i]  // Current element to be inserted
   j = i – 1  // Index of the last element in the sorted subarray

   // Move elements of the sorted subarray (A[0...i-1]) that are greater than key
   // to one position ahead of their current position
   while j >= 0 and A[j] > key
         A[j + 1] = A[j]
         j = j – 1

   A[j + 1] = key  // Insert the key (current element) into its correct position
```

CS2 Lab Exercise: Sorting Algorithm Implementation

Objective: Implement the sorting algorithm described above to arrange a sequence of numbers in non-decreasing order.

Part 1. Instructions:

1. Algorithm Description:
    - The algorithm works by iteratively building a sorted subarray at the beginning of the input array.
    - At each iteration, the algorithm removes one element from the unsorted portion of the array and finds the correct position for it in the sorted portion.
    - The element is then inserted into that position, shifting other elements to the right if necessary.
    - This process continues until the entire array is sorted.
2. Implementation Details:
    - Write a C++ app that takes an integer array (or vector) as input and sorts it using the algorithm described above.
    - Your function should modify the original array in place (i.e., do not create a new array).
    - Name your function *mysterySort*.

3. Data Sample:
    - Test your implementation with the following data sample: Array/Vector: {5, 2, 8, 6, 1, 3, 9, 7, 4}
4. Visualization:
    - During the execution of your algorithm, display the contents of the array/vector after each insertion step. This will help you visualize how the algorithm works and verify the correctness of your implementation. For example, for the given data sample, the output would look like:

```
Initial array: {5, 2, 8, 6, 1, 3, 9, 7, 4}

After inserting 2: {2, 5, 8, 6, 1, 3, 9, 7, 4}
After inserting 8: {2, 5, 8, 6, 1, 3, 9, 7, 4}
After inserting 6: {2, 5, 6, 8, 1, 3, 9, 7, 4}
After inserting 1: {1, 2, 5, 6, 8, 3, 9, 7, 4}
After inserting 3: {1, 2, 3, 5, 6, 8, 9, 7, 4}
After inserting 9: {1, 2, 3, 5, 6, 8, 9, 7, 4}
After inserting 7: {1, 2, 3, 5, 6, 7, 8, 9, 4}
After inserting 4: {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Hints:

- Start by considering the first two elements of the array. The first element can be considered as a sorted subarray of size one.
- Iterate through the remaining unsorted elements, inserting each into its correct position within the sorted subarray.
- Use a loop to shift elements in the sorted subarray to make space for the element being inserted.
- Print the array/vector after each insertion to visualize the sorting process.

Part 2. Benchmark the mysterySort app on a size varying sample of vectors.

Objective. Measure the computing time that mysterySort takes to execute on vectors holding randomly generated integer data. The sample sizes are listed in the table below.

Sample size	Execution Time (milliseconds)
100	
200	
400	
800	
1600	
3200	
6400	
12800	
25600	
51200	
124000	

Notes.

The following code generates random numbers.

```
srand(time(NULL));  //bind the generator with the internal clock

// Generate and print 5 random integers between 1 and 10000
for (int i = 0; i < 5; ++i) {
    int random_number = 1 + rand() % 10000;  // Range: [1, 10000]
    cout <<  random_number << endl;
}
```

The following code could be used to measure the elapsed time between two events.

```
        auto begin = std::chrono::high_resolution_clock::now();

        //Do something here

        auto end = std::chrono::high_resolution_clock::now();

        std::chrono::duration<double, std::milli> duration = end - begin;

        cout << " Elapsed Time: " << duration.count() << " millis" << endl;
```

You need to include the following header files

```
#include <chrono>
#include <cstdlib>
#include <ctime>
```