Time Complexity of Algorithms

On the next page you can find the C++ code for the insertion sort algorithm. In this lab, you will compare the insertion and quicksort sorting algorithms.

Code

1. Write a dummy implementation for both sorting algorithms. For now, this implementation doesn't have to do anything, just take an array of objects as a const std::vector<T>&, copy it, and return the copy.

2. Write a unit test for each of the following inputs, for both sorting algorithms:
   - A sorted array
   - A random array
   - An array sorted in reverse order

Make sure you can easily change the size of each array and use the dummy implementations to produce an output in your unit tests. Your unit tests will obviously still fail since you haven't implemented either algorithm yet.

3. Next, write one more unit test. This test will verify the consistency of your sorting methods. In this unit test, you should have 1 input sequence, and feed it to both algorithms. Next, compare the results of the insertion sort algorithms to the results of the quicksort algorithm. They should both return the array in the exact same order.

4. Next, you can replace the dummy implementation for the insertion sort algorithm with a working implementation using the code you can find on the next page. Use the unit tests you wrote earlier to verify that everything is working correctly.

5. After implementing the insertion sort algorithm, you can also Implement the quick sort algorithm, without using the standard library. You will need to look for the pseudo-code for this algorithm yourself. Try to implement the algorithm yourself without copying an existing implementation from the internet. Use the unit tests you wrote earlier to verify that the algorithm works correctly.

6. Measure the running time of both sorting algorithms in each of the 3 test cases you wrote earlier. Do this for small inputs (<50 elements) and large inputs (> 1000 elements)
   Measure the running time of your algorithm using std::chrono::steady_clock::now() or std::chrono::system_clock::now() or std::chrono::high_resolution_clock::now()

7. Finally, make sure to take a look at the different algorithms in *<algorithm>*

Report

1. Determine the time complexity of the insertion sort algorithm without looking it up. Make use of Big-O notation.
2. Make a graph of the running time of both algorithms + *std::sort*
3. Compare both of your algorithms to *std::sort*, what do you notice?
4. Compare the run-time of the insertion sort algorithm when compiling your code in debug mode and release mode, what do you notice?

```cpp
template <typename T>
std::vector<T> insertion_sort(const std::vector<T>& vector) noexcept
{
    std::vector<T> sorted = vector; // Copy the input data
    for (std::size_t i = 1; i < sorted.size(); ++i) // Start from the second element
    {
        T key = sorted[i]; // Store the current element
        std::size_t j = i; // j tracks the position to insert the current element
        while (j > 0 && sorted[j - 1] > key) // Move elements to make room for key
        {
            sorted[j] = sorted[j - 1];
            j--;
        }
        sorted[j] = key; // Insert the key in the right position
    }
    return sorted;
}
```