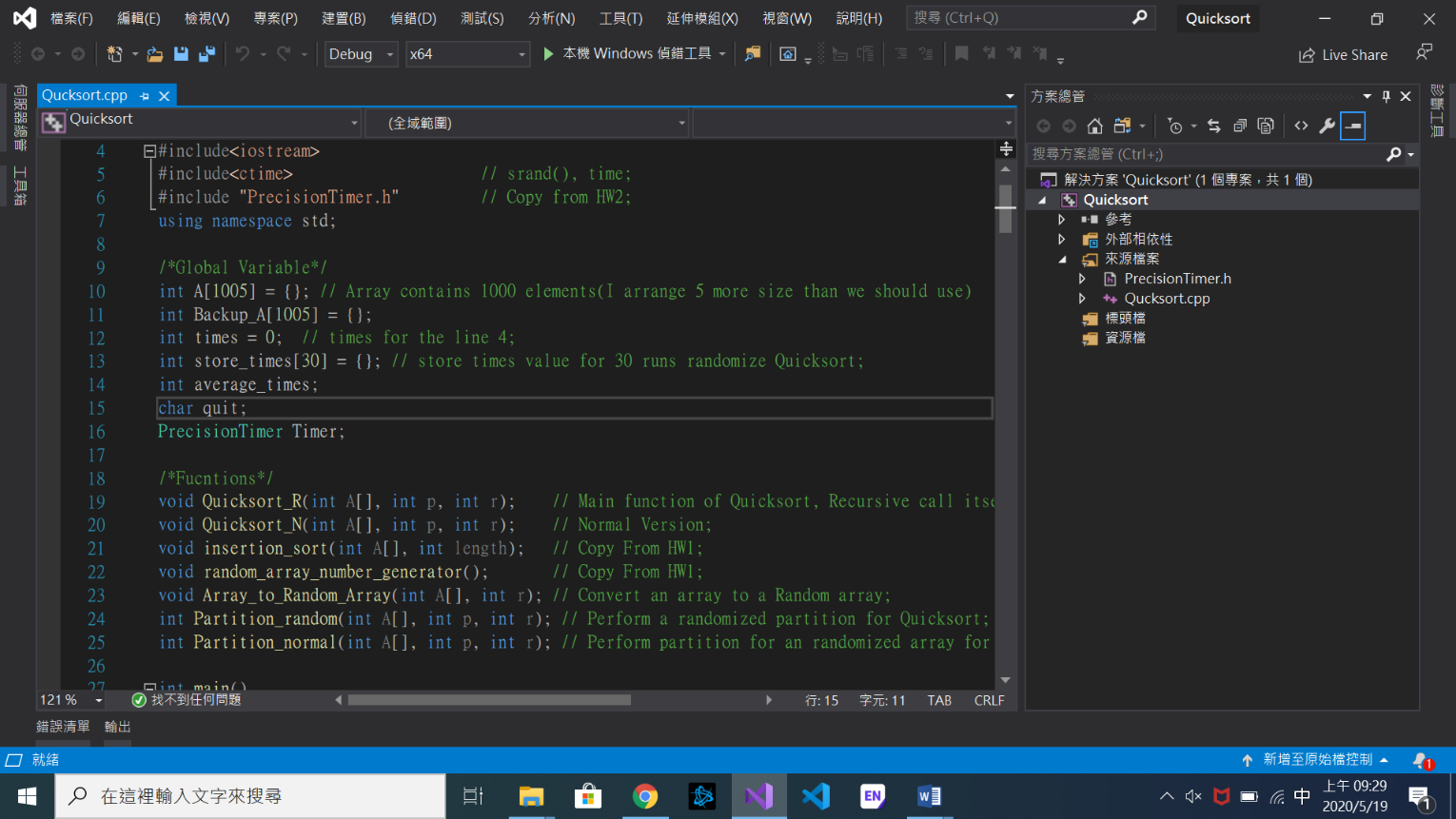
HW3 Randomized Quicksort 0617052林明佑

**Description of implementation:**

7 small functions + 1 main function + 1 header file.



Quicksort\_R: Main function of Quicksort, Recursive call itself

Quicksort\_N: Normal Version of Quicksort(Array is already randomized before).

Array\_to\_Random\_Array: Convert an array to a Random array.

Partition\_random: Perform a randomized partition for Quicksort.

Partition\_normal: Perform partition for an randomized array for Quicksort.

Insertion\_sort: Copy from HW1.

Random\_array\_number\_generator: Copy from HW1.

PrecisionTimer.h : Much more precision timer. Return value is **sec.** Copy from HW2**.**

**Problems:**

**(a)**

11856 times (30 runs of quicksort, taking the average.)

**(b)**

1. **Running time**

Randomized the array before the Quicksort: 0.0178268 sec.

Randomized Quicksort: 0.0107811 sec.

1. **Comparison**

Running time of the two methods is almost the same. (at the array size 1000).

The two methods have the same complexity O(nlogn) in all cases. (Randomized process takes O(n) in the two methods).

From the probability definition, we can know that the worst case has the same probability in the two methods.

(c)

|  |  |  |
| --- | --- | --- |
|  | Insertion sort | Quicksort |
| Running time | 0.0013047 | 0.0001439 |
| Complexity | O(n^2) | O(nlogn), average case |

Insertion sort time complexity is always larger than Quicksort in the average case.

We can see that when the array size is larger than 1000, then Quicksort has the smaller running time than Insertion sort.

**Result picture:**

