Assignment B Lab Book

# Week 2 – Lab B

Date: 17h Oct 2022

## Q1. Timing

Locate the Solution Explorer within Visual Studio and select the Timing project. Right click on this project and select Build. This should compile and link the project. Now run the Timing program. This application attempts to time a very small piece of code to CPU clock precision.

### Solution:

Run the application on your PC and take a look at the output.

* Overhead – time taken to call the timing functions
* Median – The middle time, when the list of times are sorted
* Mean – The mean of the times, discounting the lower and upper 10% of measurements

1. The Overhead value was 673 when run on my computer
2. The median Duration was 179
3. The mean was 178.76

Sample Output:

A picture containing text, screenshot, monitor, black

Description automatically generated

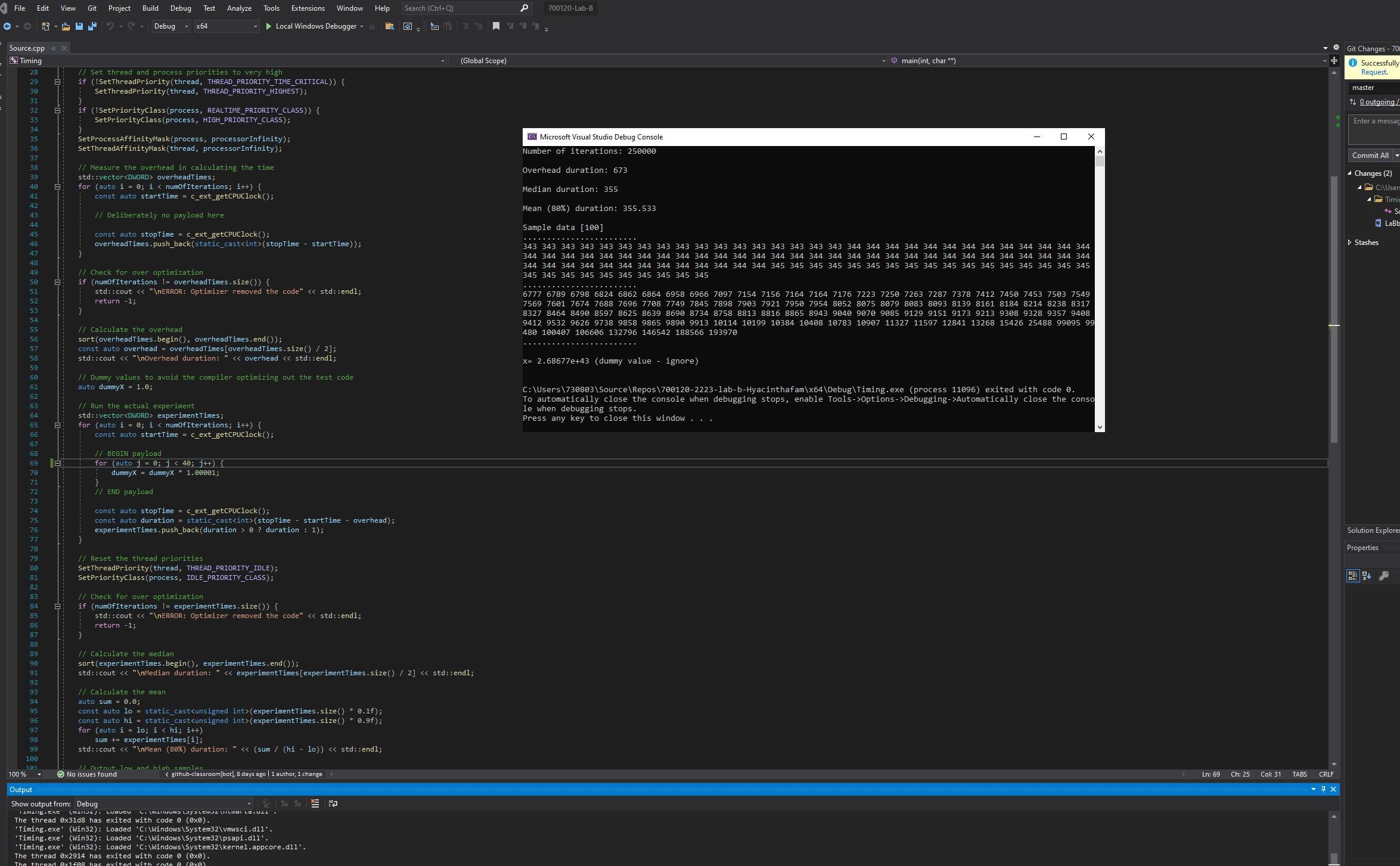
Question : Try increasing the limit of the loop from 20 to 40. Can you explain the result?

Solution : On increasing the loop limit to 40, the Overhead duration remained unchanged at 673

The median duration doubled from 179 to 355

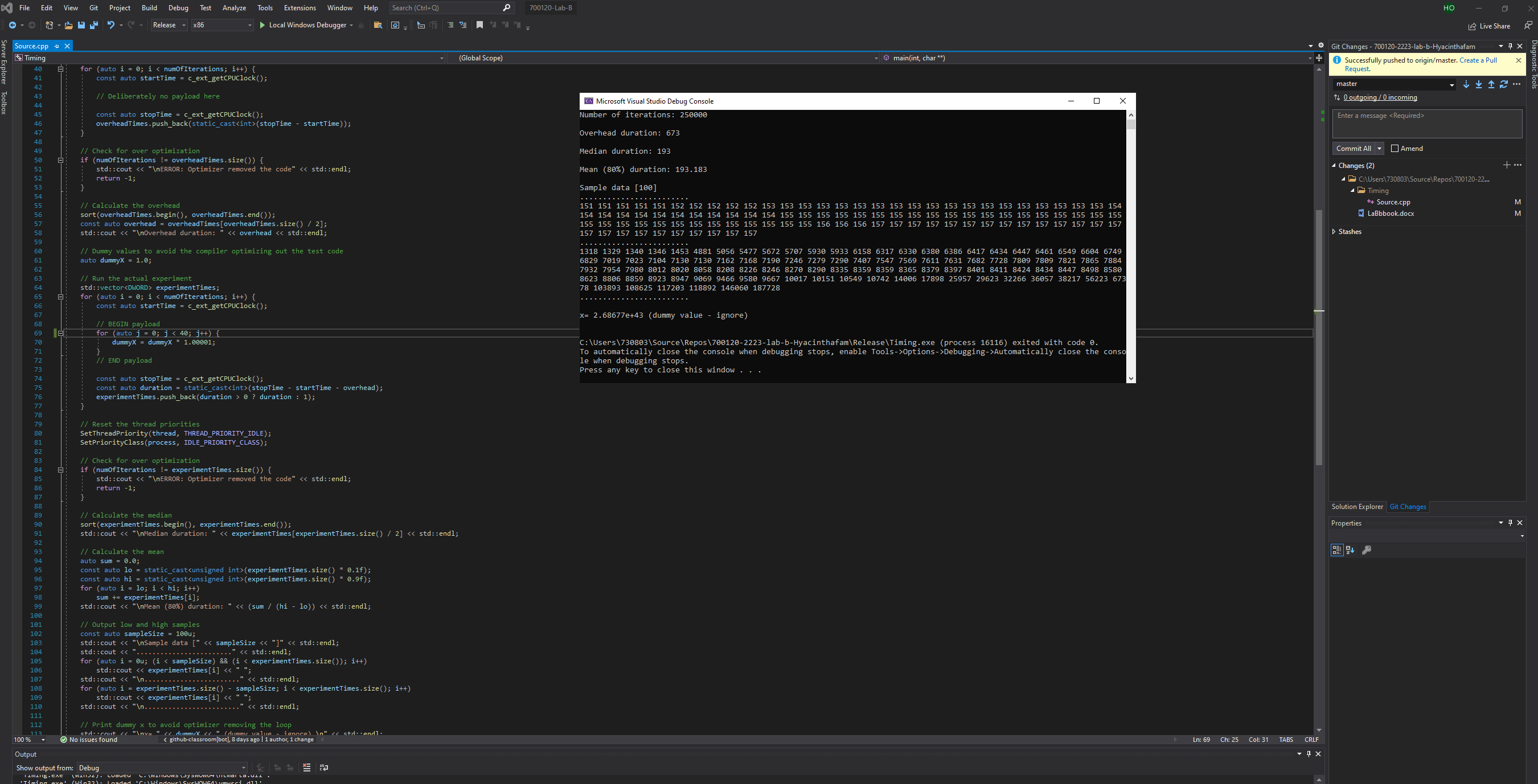
The mean also doubled from 178.76 to 355.533

Sample Output:

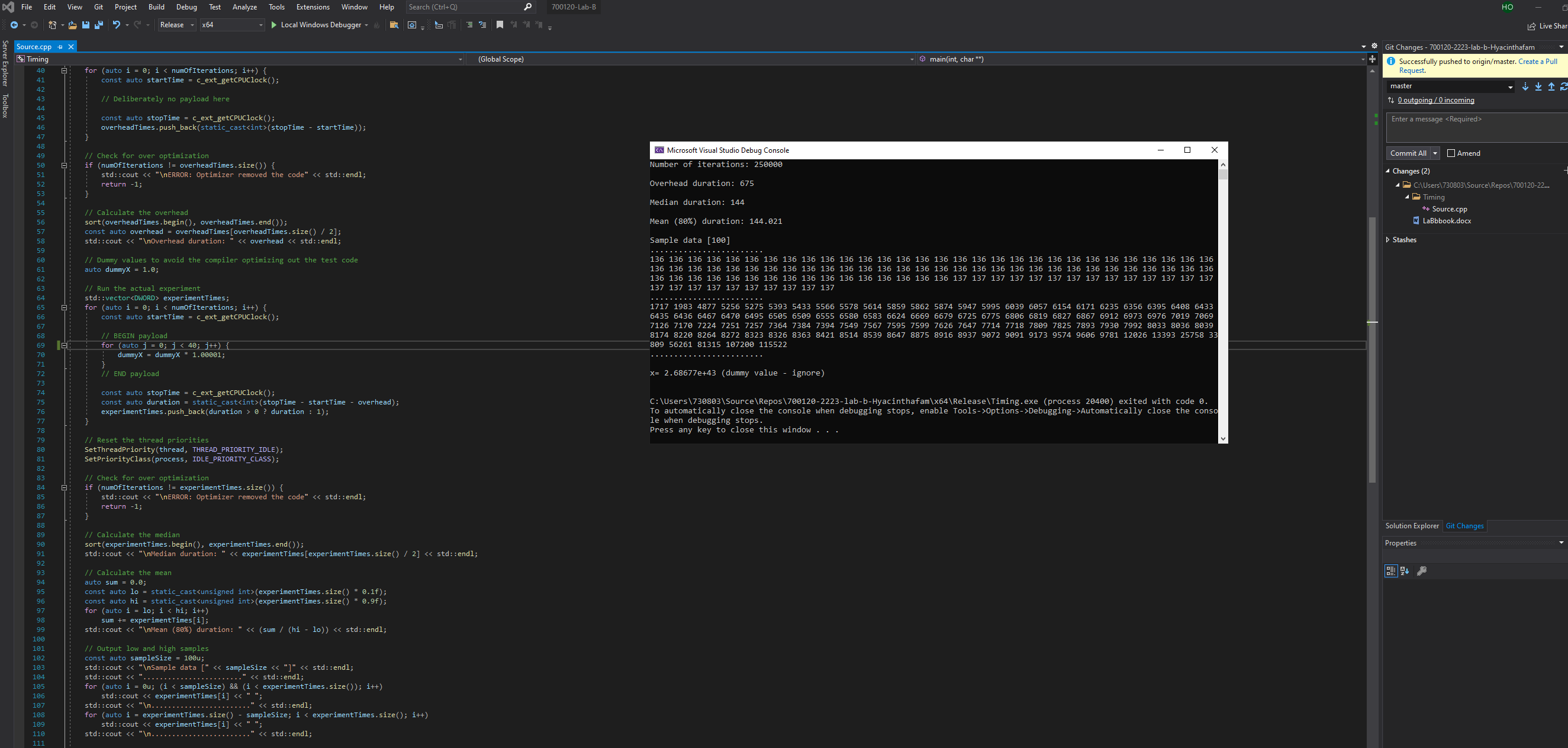


Now run the application both in Release x86 and Release x64 modes:

X86 Sample Output



X64 Sample Output



### Test data:

N/A

### Reflection:

The CPU timing varied over different conditions, the first case was increasing the loop from 20 to 40, the loop increment affected the Overhead Duration, the Median Duration and the Mean values of the CPU respectively. Also running the timing on different architectures i.e x86 and x64 gave different results, as well as changing the debug mode to release mode. There was significant increase when I changed the compiler from debug to release mode.

### Metadata:

N/A

### Further information:

N/A

## Q2. Timing Own Code

Replace the payload with some of your own code. When adding code to the payload, try and write code that the compiler will not optimise away. A trick is to calculate a dummy value that is later printed. See dummyX in the example payload:

for (auto j = 0; j < 60; j++) {

dummyX = dummyX \* 1.00001;

}

Sample Output:

A screenshot of a computer

Description automatically generated with medium confidence

**Test Data:**

4 was used as the test data to terminate the console.

**Reflection:**

The cin>> keyword is used to accept input into the program in a C++ project.

**Metadata:**

N/A.

Further Information:

**Q3.** Conditionals

In the lectures you will have covered the if, switch and ?: conditional statements Add each in turn to the payload to try and identify any performance differences.

a). Are the results as you expected?

b). Is there anything you need to write in your log book for future reference?

Solution: using (if)

// BEGIN payload

for (auto j = 0; j < 40; j++) {

//dummyX = dummyX \* 2.00001;

if (j > 20) {

dummyX = dummyX \* 2.000;

}

else {

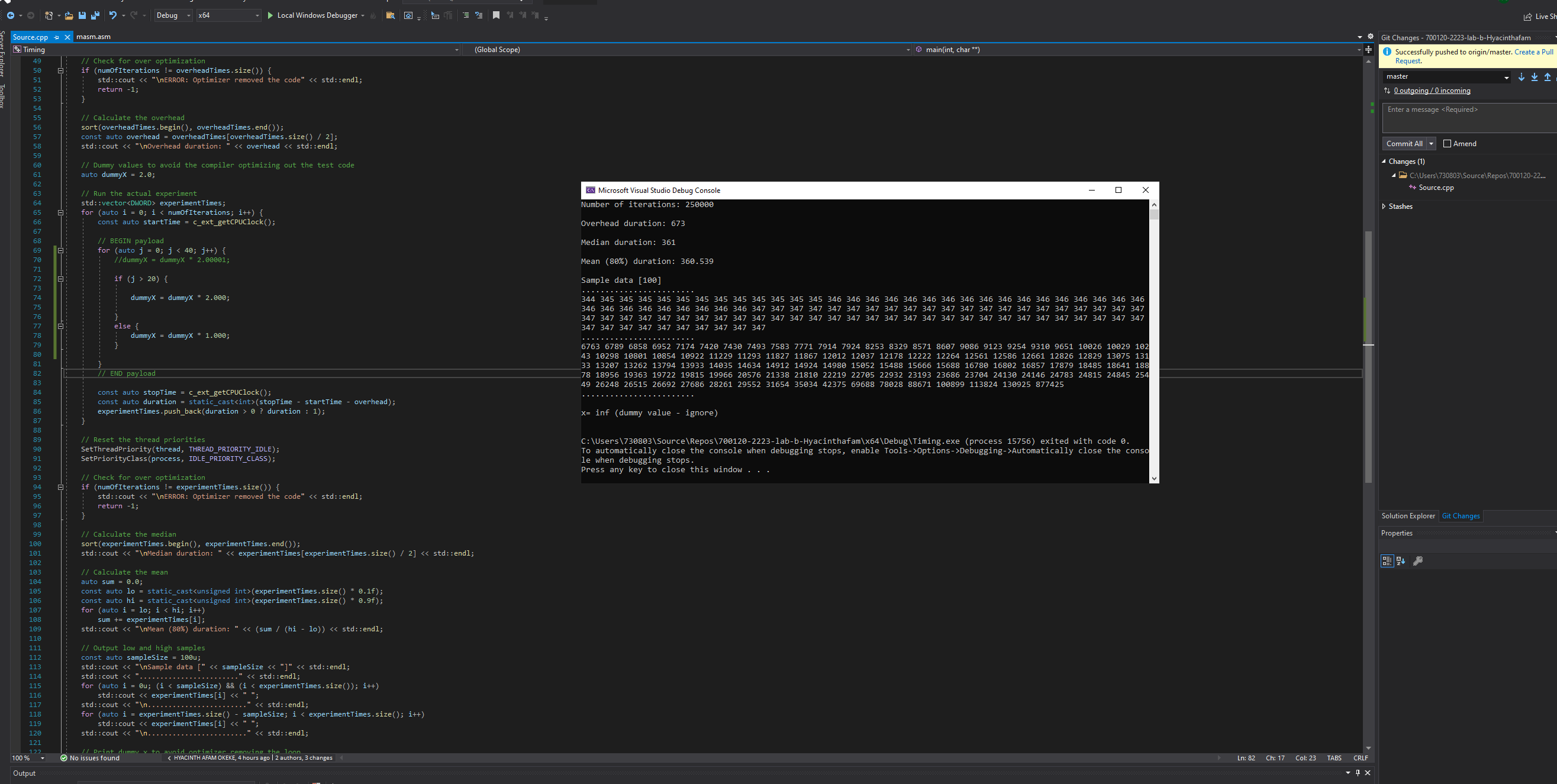
dummyX = dummyX \* 1.000;

}

}

// END payload

Sample Output :



Solution Using: Switch

Solution Using: Ternary (?)

for (auto j = 0; j < 20; j++) {

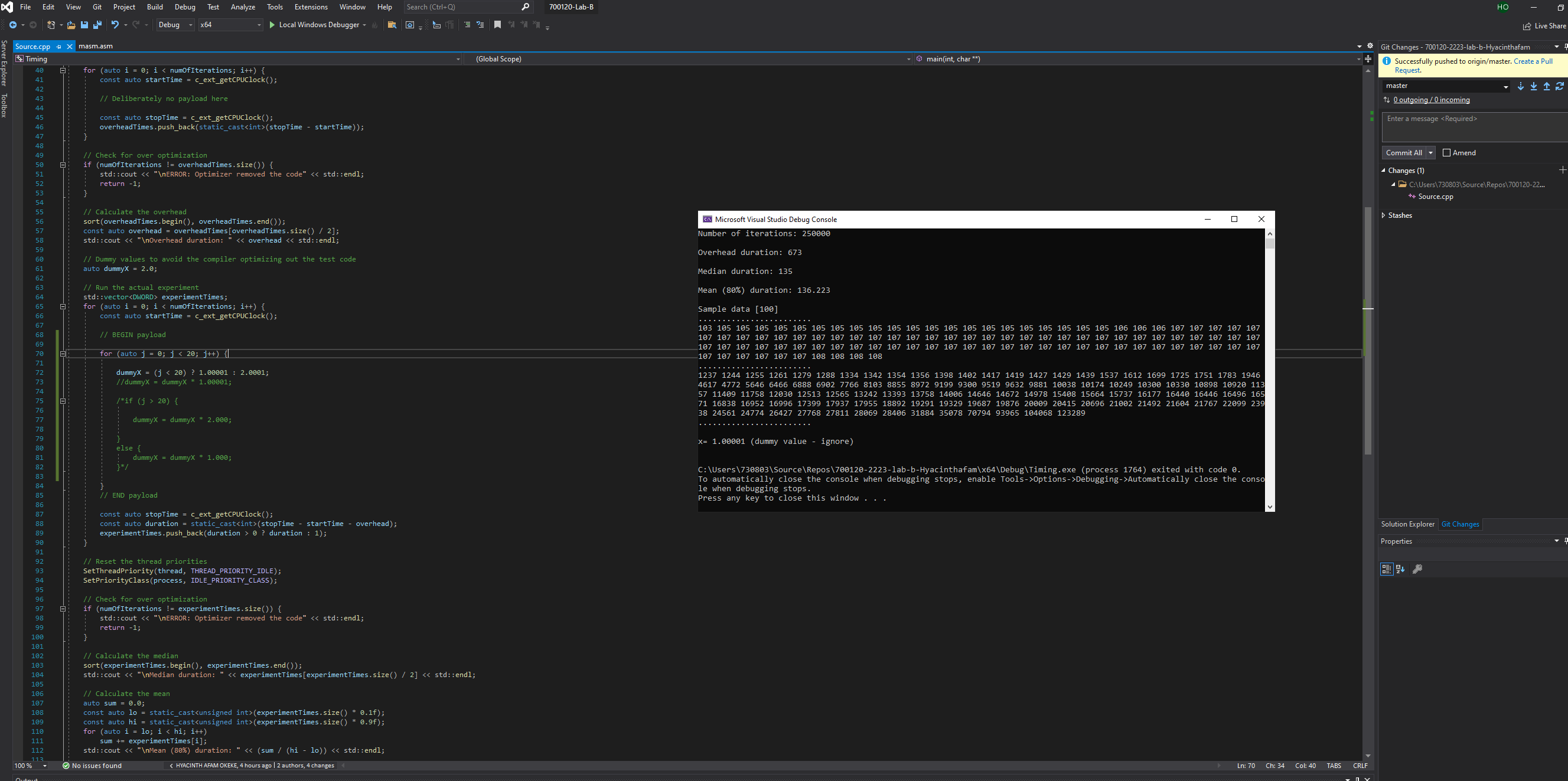
dummyX = (j < 20) ? 1.00001 : 2.0001;

}

// END payload

**Test date : N/A**

**Sample Output:**

****

**Reflection:**

The ternary operator (?) was a more convenient conditional to use because it is less ambiguous and direct when the conditions are met.

**Metadata: N/A**

**Further Information : N/A**

**Q4. Branch Prediction:**

**Solution:**

**Test data: N/A**

**Sample Output**

**Reflection :**

**Metadata: N/A**

**Further Information: N/A**

**Q5. Exiting a nested Loop**

**a). Two conditions in each conditional section of the loops. One for the loop control and the other as the exit condition**

**Solution:**

const int list = 5;

for (auto i = 0u; i < list; i++)

for (auto j = 0u; j < list; j++) {

double dummyX;

std::cin >> dummyX;

if (dummyX < 20.0)

return;

std::cout << dummyX;

}

};

**b). An additional if statement immediately following the inner loop to catch and propagate a break statement**

**solution:**

**c). A goto statement in the inner loop**

**Solution:**

const int list = 5;

for (auto i = 0u; i < list; i++)

for (auto j = 0u; j < list; j++) {

double dummyX;

std::cin >> dummyX;

if (dummyX < 20.0)

return;

goto combinedLoop;

}

};

combinedLoop();

**d). A lambda function**

**Solution :**

const int list = 5;

const auto combinedLoop = [] {

for (auto i = 0u; i < list; i++)

for (auto j = 0u; j < list; j++){

double dummyX;

std::cin >> dummyX;

if (dummyX < 20.0)

return;

std::cout << dummyX;

}

};

combinedLoop();

**Reflection:**

**Using the Goto Statememt to exit a nested loop is not a good idea, I used a lambda function instead to exit the loop.**

**Metadata: N/A**

**Further Information :N/A**

**Q6. Range Based Loop.**

**Solution :**

int list[8] = { 1,2,3,4,5,6,7,8 };

for (auto j :list) {

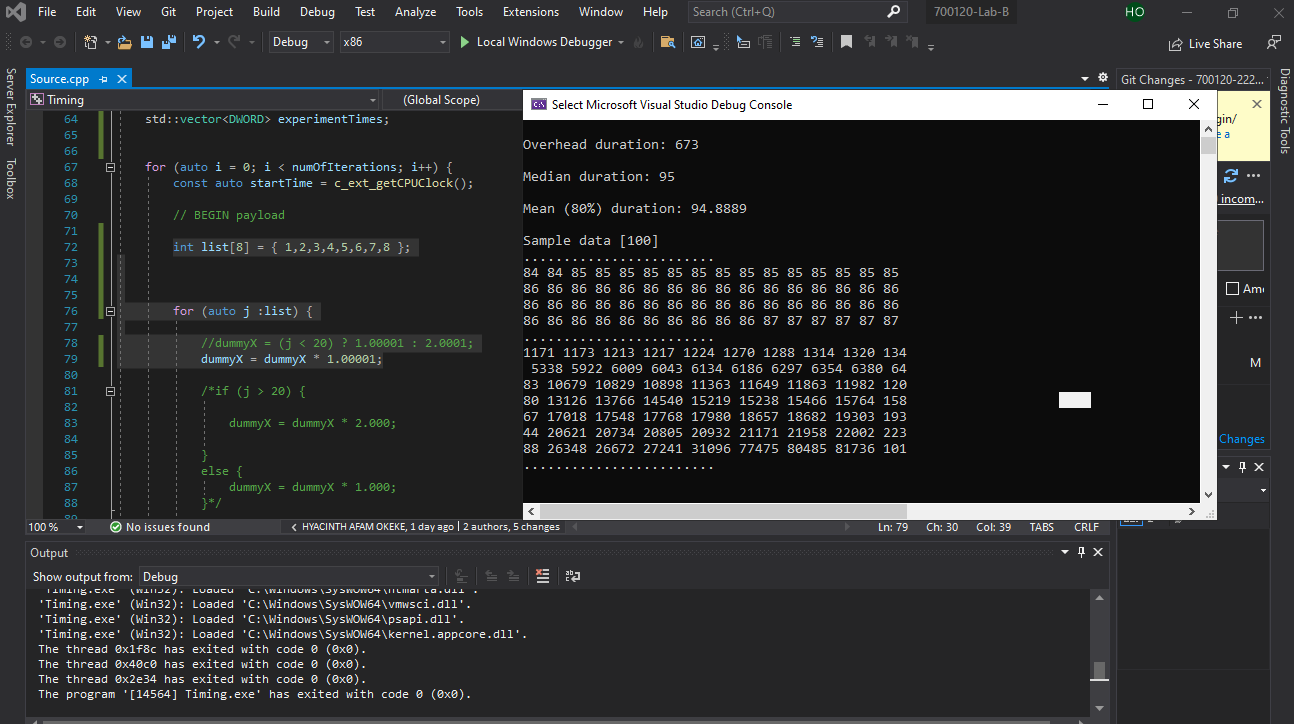
dummyX = dummyX \* 1.00001;

}

**Test Data:**

**1 - 8**

**Sample Output:**

****

**Reflection :**

**The Standard Loop makes for more efficient code than the range loop, I prefer the former because the code iterates through the defined loop automatically using the increment operator(++) while in the range loop the programmer will outline all possible arrays to be looped .**

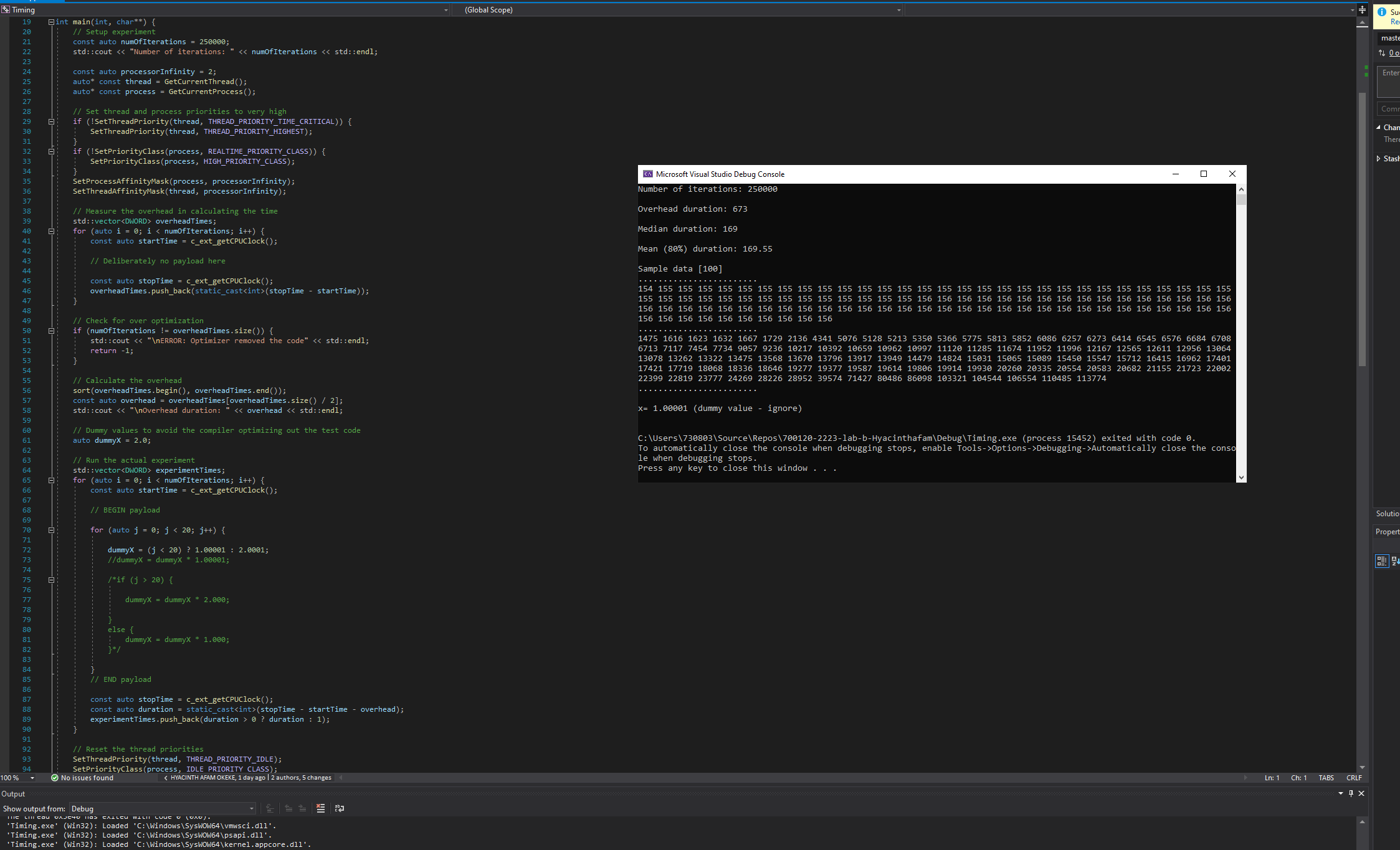
**Metadata: N/A**

**Further Information : N/A**

**Q7. Architecture**

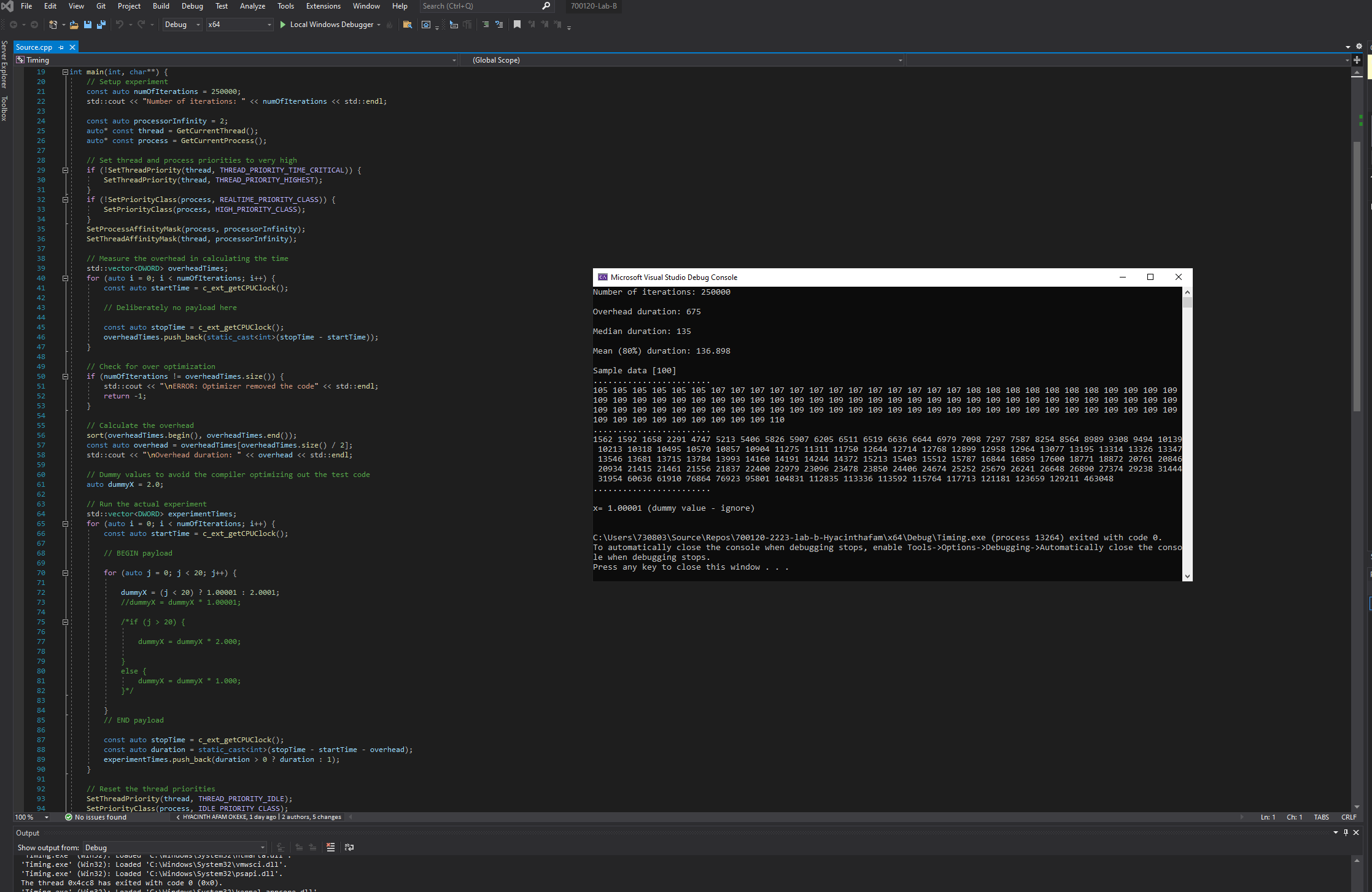
**X86 Architecture:**

**Sample Outputs:**

****

**X64 Architecture :**

**Sample Output:**

****

**Reflection:**

There were remarkable difference when I ran the payload code on different CPU architectures, the x64 architecture was considerably faster in its processing time compared to the x86 architecture..