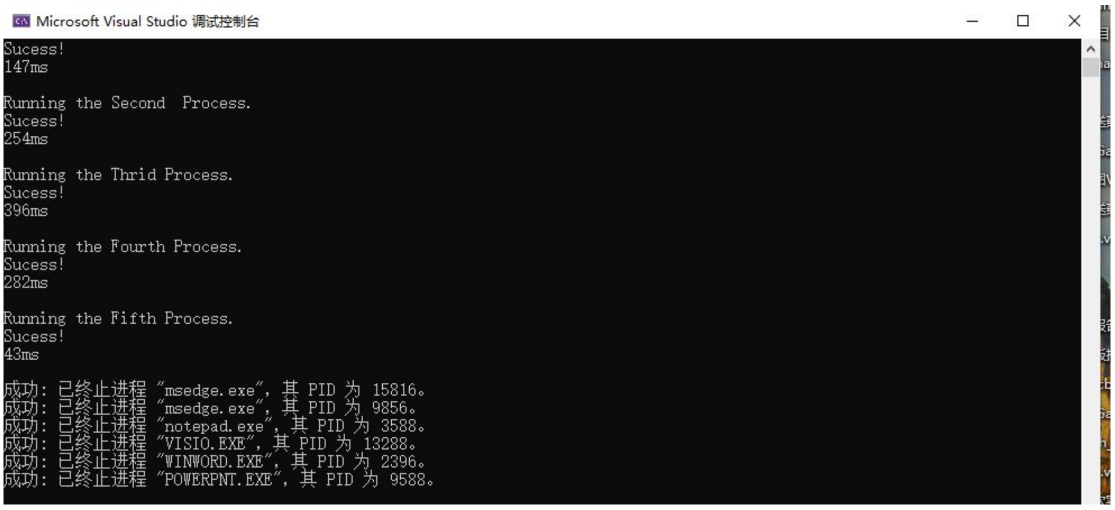
Part I. (50 points)

1) Measure the average time it takes to start up a process, by writing a program that does it. Use five different processes from your computer and report the time. (5 Points) (Hint: use Chrono)


```
Sucess!
147ms

Running the Second  Process.
Sucess!
254ms

Running the Thrid Process.
Sucess!
396ms

Running the Fourth Process.
Sucess!
282ms

Running the Fifth Process.
Sucess!
43ms

成功: 已终止进程 “msedge.exe”, 其 PID 为 15816。
成功: 已终止进程 “msedge.exe”, 其 PID 为 9856。
成功: 已终止进程 “notepad.exe”, 其 PID 为 3588。
成功: 已终止进程 “VISIO.EXE”, 其 PID 为 13288。
成功: 已终止进程 “WINWORD.EXE”, 其 PID 为 2396。
成功: 已终止进程 “POWERPNT.EXE”, 其 PID 为 9588。
```

Figure 1: Process startup time

2) Measure the average time it takes to start up a thread, by writing a program that does it. (5 Points) (Hint: use Chrono) (you may create your own thread in this case) (Just one thread is enough here.)

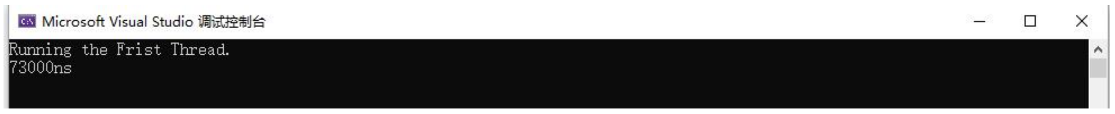

```
Running the Frist Thread.
73000ns
```

Figure 2: Thread startup time

3+)Run it several times, and present a histogram of your results for process and threads separately. (10 Points)

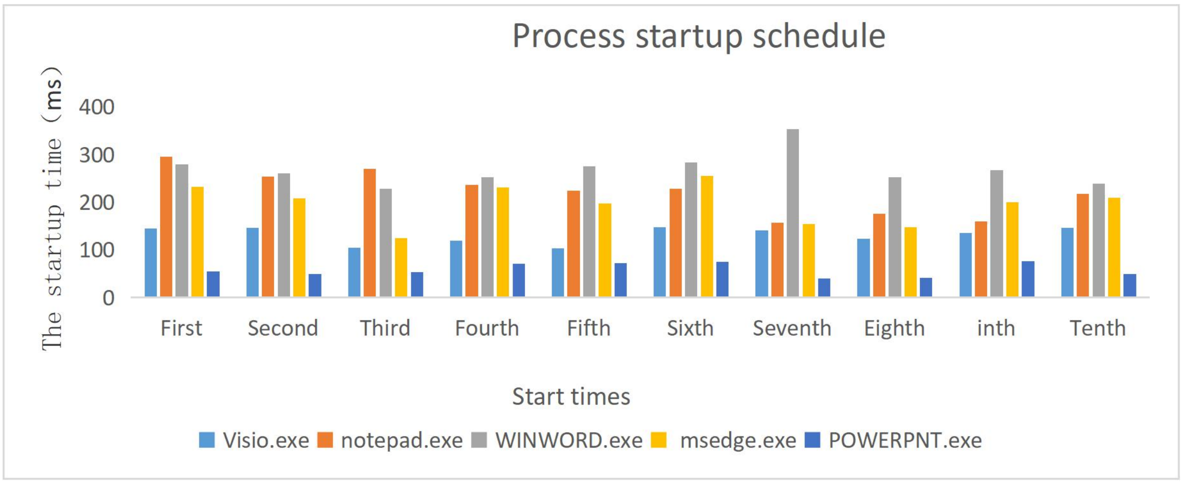
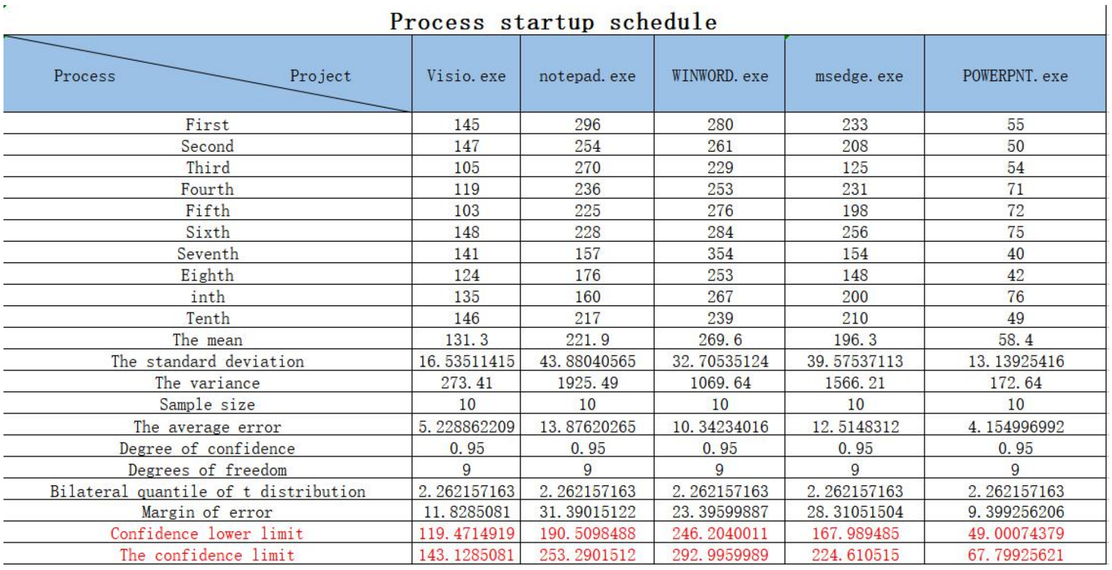


Figure 3: Histogram of process startup time

4) What are mean and standard deviation for your observations for both the process and threads? (10 Points)

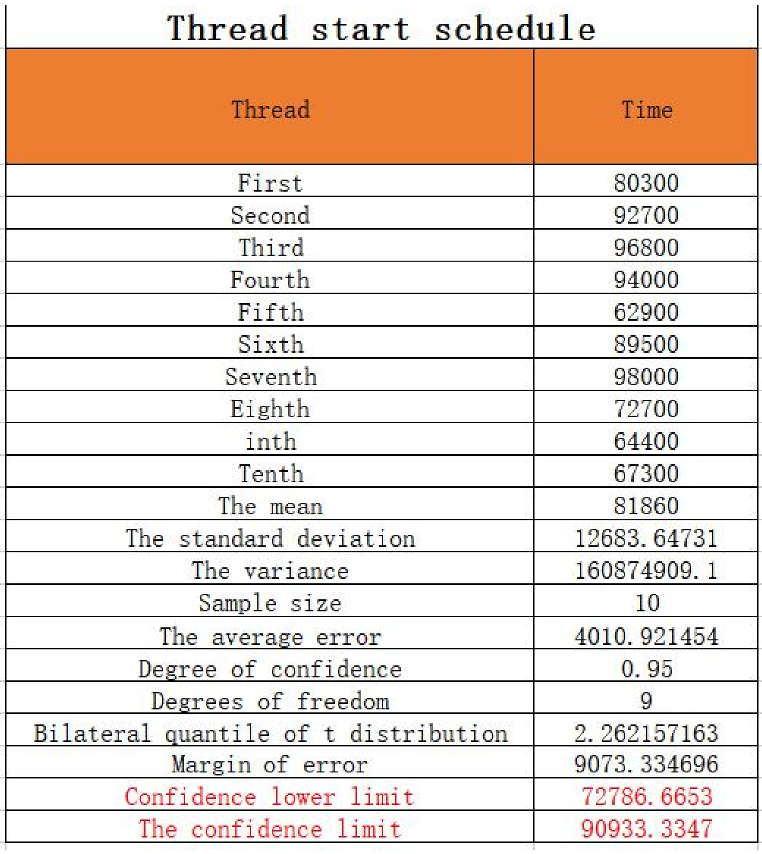
5)What is the 95% confidence level interval for both? Please refer to the link below to understand about 95% confidence interval. (10 Points)



Process startup schedule

Process \ Project	Visio.exe	notepad.exe	WINWORD.exe	msedge.exe	POWERPNT.exe
First	145	296	280	233	55
Second	147	254	261	208	50
Third	105	270	229	125	54
Fourth	119	236	253	231	71
Fifth	103	225	276	198	72
Sixth	148	228	284	256	75
Seventh	141	157	354	154	40
Eighth	124	176	253	148	42
inth	135	160	267	200	76
Tenth	146	217	239	210	49
The mean	131.3	221.9	269.6	196.3	58.4
The standard deviation	16.53511415	43.88040565	32.70535124	39.57537113	13.13925416
The variance	273.41	1925.49	1069.64	1566.21	172.64
Sample size	10	10	10	10	10
The average error	5.228862209	13.87620265	10.34234016	12.5148312	4.154996992
Degree of confidence	0.95	0.95	0.95	0.95	0.95
Degrees of freedom	9	9	9	9	9
Bilateral quantile of t distribution	2.262157163	2.262157163	2.262157163	2.262157163	2.262157163
Margin of error	11.8285081	31.39015122	23.39599887	28.31051504	9.399256206
Confidence lower limit	119.4714919	190.5098488	246.2040011	167.989485	49.00074379
The confidence limit	143.1285081	253.2901512	292.9959989	224.610515	67.79925621

Figure 4: Process startup data



Thread start schedule

Thread	Time
First	80300
Second	92700
Third	96800
Fourth	94000
Fifth	62900
Sixth	89500
Seventh	98000
Eighth	72700
inth	64400
Tenth	67300
The mean	81860
The standard deviation	12683.64731
The variance	160874909.1
Sample size	10
The average error	4010.921454
Degree of confidence	0.95
Degrees of freedom	9
Bilateral quantile of t distribution	2.262157163
Margin of error	9073.334696
Confidence lower limit	72786.6653
The confidence limit	90933.3347

Figure 5: Thread startup data

The data in the graph above was calculated using a function in the EXECL table, with startup times measured in milliseconds for processes and nanoseconds for threads.

Rows 13 and 14 in the first table are the mean and standard deviation of process start time respectively, and rows 13 and 14 in the second table are the mean and standard deviation of thread start time respectively.

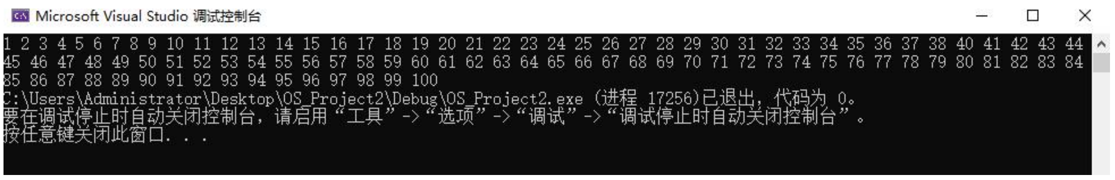
The red data in the first table is the 95% confidence level interval for the process, and the red data in the second table is the 95% confidence level interval for the thread.

6) Why shouldn't it always take exactly the same amount of time to perform this simple action for both? (10 Points)

Part II. (50 points)

This part will test how effective a thread implementation is over a process. Write a program that will do the following.

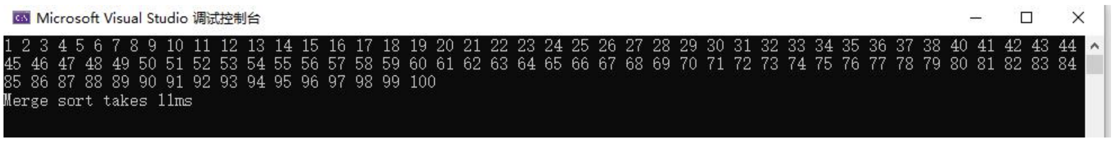
1) Write a C++ program that uses a vector to perform a merge sort on a queue of integers (Use the following queue: 39 1 95 75 92 43 98 77 46 54 48 82 30 85 74 53 37 61 51 4 8 68 17 15 10 56 47 93 23 59 60 86 21 31 36 84 19 20 12 27 76 71 45 25 91 22 13 94 55 34 52 67 96 100 29 57 65 9 49 38 33 66 80 99 44 90 26 3 28 87 64 83 73 11 58 18 79 6 35 16 41 78 88 69 70 97 42 7 2 40 81 62 5 14 72 50 32 89 24 63.) The merge shot should be implemented as a method. You may sort them in an ascending or a descending order. For this you may use any reference that you want and borrow the code if needed. (10 Points)



```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 40 41 42 43 44
45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
C:\Users\Administrator\Desktop\OS_Project2\Debug\OS_Project2.exe (进程 17256)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

Figure 6: Implementation of merge sort

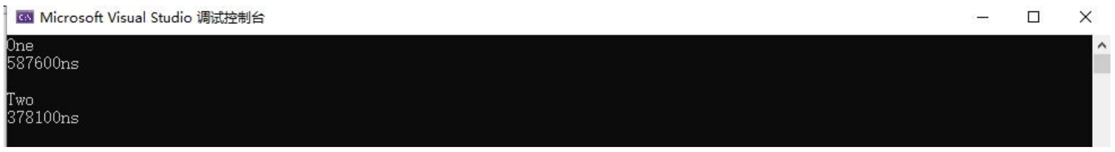
2) Assuming that the entire program is a process by itself, calculate the time to run the program as a whole. (Hint: Use *std::chrono*). (20 points)



```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 40 41 42 43 44
45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Merge sort takes 11ms
```

Figure 7: Merge sort implementation and timing

3) Implement two threads and run the same program using the two threads. Now, calculate the time it takes to run the program successfully. (20 points) (Please remember that using threads the program execution should take less time as compared to a process.)



```
One
587600ns

Two
378100ns
```

Figure 8: Using two threads to execute the same program

The appendix part

Part I. (50 points)

```
// OS_Project.cpp: Master project file
#include <iostream>
//Import I/O header files
#include <stdlib.h>
//Standard Library header files that allow programs to use functions such as
free() and sexit()
#include <chrono>
//Introduces the chrono timing header to calculate the start time of the process
#include <windows.h>
//Introduction of the Windows library, which is an important header file for
writing Windows programs, in order to use ShellExecute function to start the
process ()
#include <thread>

using namespace std;
using namespace std::chrono;//Use the STD namespace to make all identifiers
valid

class Timer {
public:
    Timer() :m_begin(high_resolution_clock::now()) {}
    void reset() { m_begin = high_resolution_clock::now(); }
    //The output of milliseconds
    int64_t elapsed() const
    {
        return
duration_cast<chrono::milliseconds>(high_resolution_clock::now() -
m_begin).count();
    }

    //nanosecond
    int64_t elapsed_nano() const
    {
        return
duration_cast<chrono::nanoseconds>(high_resolution_clock::now() -
m_begin).count();

private:
    time_point<high_resolution_clock> m_begin;
};
```

```
void StratVisio()
{
    cout << "Running the First Process." << endl;
    Timer timer;
    system("start  Visio.exe");
    //Executing Dos Commands
    cout << "Sucess!" << endl;
    cout << timer.elapsed() << "ms\n" << endl;
}

void StratTxt()
{
    cout << "Running the Second  Process." << endl;
    Timer timer;
    ShellExecute(0, "open", "C:\\Users\\Administrator\\Desktop\\介绍.txt",
NULL, NULL, SW_SHOWNORMAL);
    cout << "Sucess!" << endl;
    cout << timer.elapsed() << "ms\n" << endl;
}

void startWINWORD()
{
    cout << "Running the Third Process." << endl;
    Timer timer;
    system("start WINWORD.EXE ");
    cout << "Sucess!" << endl;
    cout << timer.elapsed() << "ms\n" << endl;


}

void StratWeb()
{
    cout << "Running the Fourth Process." << endl;
    Timer timer;
    ShellExecute(NULL, "open", "http://www.zknu.edu.cn", "", "", SW_SHOW);
    //The shellexecute arguments are {specify parent window handle}, specify
actions, specify the file or program to open, and specify parameters for the
program to open;
    //If you're opening a file this should be null, default directory, open
option
    cout << "Sucess!" << endl;
    cout << timer.elapsed() << "ms\n" << endl;
```

```cpp
}

void StratPOWERPNT()
{
    cout << "Running the Fifth Process." << endl;
    Timer timer;
    ShellExecute(NULL, "open", "POWERPNT.EXE", NULL, NULL, SW_SHOWNORMAL);
    cout << "Sucess!" << endl;
    cout << timer.elapsed() << "ms\n" << endl;
}

void Close()
{
    system("taskkill /f /im msedge.exe");
    system("taskkill /f /im notepad.exe");
    system("taskkill /f /im Visio.exe");
    system("taskkill /f /im WINWORD.EXE");
    system("taskkill /f /im POWERPNT.EXE");
}

void Thread1()
{
    int i;
    int a, b, c;
    for (i = 100;i <= 999;i++) {
        a = i / 100;       // hundred
        b = (i / 10) % 10; //ten
        c = i % 10;
        if (a * a * a + b * b * b + c * c * c == i) {}
           //cout << i << endl;

}

void CreatThread()
{
    cout << "Running the First Thread." << endl;
    Timer timer1;
    thread t1(Thread1);
    cout << timer1.elapsed_nano() << "ns\n" << endl;
    t1.detach();
}

int main(int argc, char* argv[])
```

```
{
   /*StratVisio();
   StratTxt();
   startWINWORD();
   StratWeb();
   StratPOWERPNT();
   Close();*/
   CreatThread();
    return 0;
}
```

Part II.

```
#include <iostream>
#include <iomanip>//Formatted output
#include <chrono>
//Introduces the chrono timing file used to calculate the start time of a process
#include <cmath>
#include <thread>
using namespace std::chrono;//Use the STD namespace to make all identifiers valid
using namespace std;

//Merge sort (from small to large)
//Function parameters: array to be sorted, left and right boundaries.
//Returned value: None void
class Timer {
public:
    Timer() :m_begin(high_resolution_clock::now()) {}
    void reset() { m_begin = high_resolution_clock::now(); }

    //The output of milliseconds
    int64_t elapsed() const
    {
        return duration_cast<chrono::milliseconds>(high_resolution_clock::now() -
m_begin).count();
    }

    //  nanosecond
    int64_t elapsed_nano() const
    {
        return duration_cast<chrono::nanoseconds>(high_resolution_clock::now() -
m_begin).count();
    }
private:
    time_point<high_resolution_clock> m_begin;
};
void Merge(long* data, long left, long mid, long right);

void MergeSort(long* data, long left, long right)
{
    if (left >= right)//exit
    {
        return;
    }
    else
    {
        int mid = (left + right) / 2;//binary
        MergeSort(data, left, mid);//Process the left-hand subsequence
        MergeSort(data, mid + 1, right);//Process the right-hand subsequence
        //Merge sort
        Merge(data, left, mid, right);
    }
}
//Function Merges subsequences
//Function arguments: Requires an array to merge, left, middle, and right index
Pointers

void Merge(long* data, long left, long mid, long right)
{
    int i = left, j = mid + 1, k = left;
    int temp[10000] = { 0 };//Temporary array
    //Assign to a temporary array
    while (i <= mid && j <= right)//Left subsequence to mid, right subsequence to
right
    {
        if (data[i] < data[j])
        {
            temp[k++] = data[i++];
        }
        else
        {
            temp[k++] = data[j++];
        }
    }
    while (i <= mid)//The left subsequence is left unassigned, and the remaining
elements are assigned
    {
        temp[k++] = data[i++];
    }
    while (j <= right)
    {
        temp[k++] = data[j++];
    }
    for (int i = left;i <= right;i++)
    {
        data[i] = temp[i];
    }
}

void countPrime() {
    int count = 1000 - 1;
    for (int i = 2; i <= 1000; i++) {
        for (int j = 2; j <= sqrt(i); j++) {
            if (i % j == 0) {
                count--;
                break;
            }
        }
    }
    //cout << count;
}

void CreateThread1()
{
    Timer timer1;
    cout << "One\n";
    thread t1(countPrime);
    cout << timer1.elapsed_nano() << "ns\n" << endl;
    t1.detach();
}

void CreateThread2()
{
    Timer timer2;
    cout << "Two\n";
    thread t2(countPrime);
    cout << timer2.elapsed_nano() << "ns\n" << endl;
    t2.detach();
}

int main()
{
        long data[100] = {
        39,1,  95,75,  92,43,98,77,46,54 ,
        48,82,30,85,  74,53,37,61,51,4,
        8,  68,17,15,  10,56,47,93,23,59,
        60,86,21,31,  36,84,19,20,12,27,
        76,71,45,25,  91,22,13,94,55,34,
        52,67,96,100,29,57,65,9,  49,38,
        33,66,80,99,  44,90,26,3,  28,87,
        64,83,73,11,  58,18,79,6,  35,16,
        41,78,88,69,  70,97,42,7,  2,  40,
        81,62,5,  14,  72,50,32,89,24,63,  };
        Timer timer;
        MergeSort(data, 1, 99);
        for (int i = 1;i <= 99;i++)
        {
          cout << data[i]<<" ";
        }
          cout <<"\nMerge sort takes " << timer.elapsed() << "ms\n" << endl;

    /*CreateThread1();
    CreateThread2();*/
}
```