Spike: 2
Title: Performance Measurement

Author: Ryan Chessum, 102564760

Goals / deliverables:

- Code, see /07 – spike - Performance Measurement/Performance Measurement/
- Spike Report

Technologies, Tools, and Resources used:

List of information needed by someone trying to reproduce this work

- Visual Studio 2019
- C plus plus reference (https://www.cplusplus.com/reference/)
- Canvas sample code

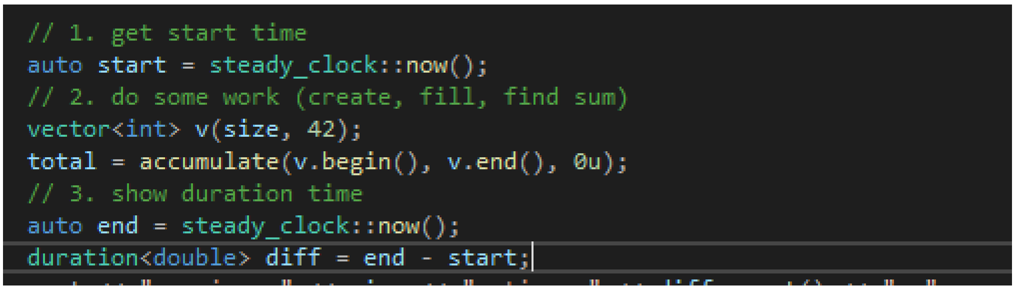
Tasks undertaken:

- Download and install Visual Studio
- Create a new C++ project

What we found out:

Performance measurement is.

We can measure the time it takes to measure a function by recording the time before we star doing anything, performing the tasks we want to record time for, recording the time again afterwards and then getting the difference between the two recorded times.

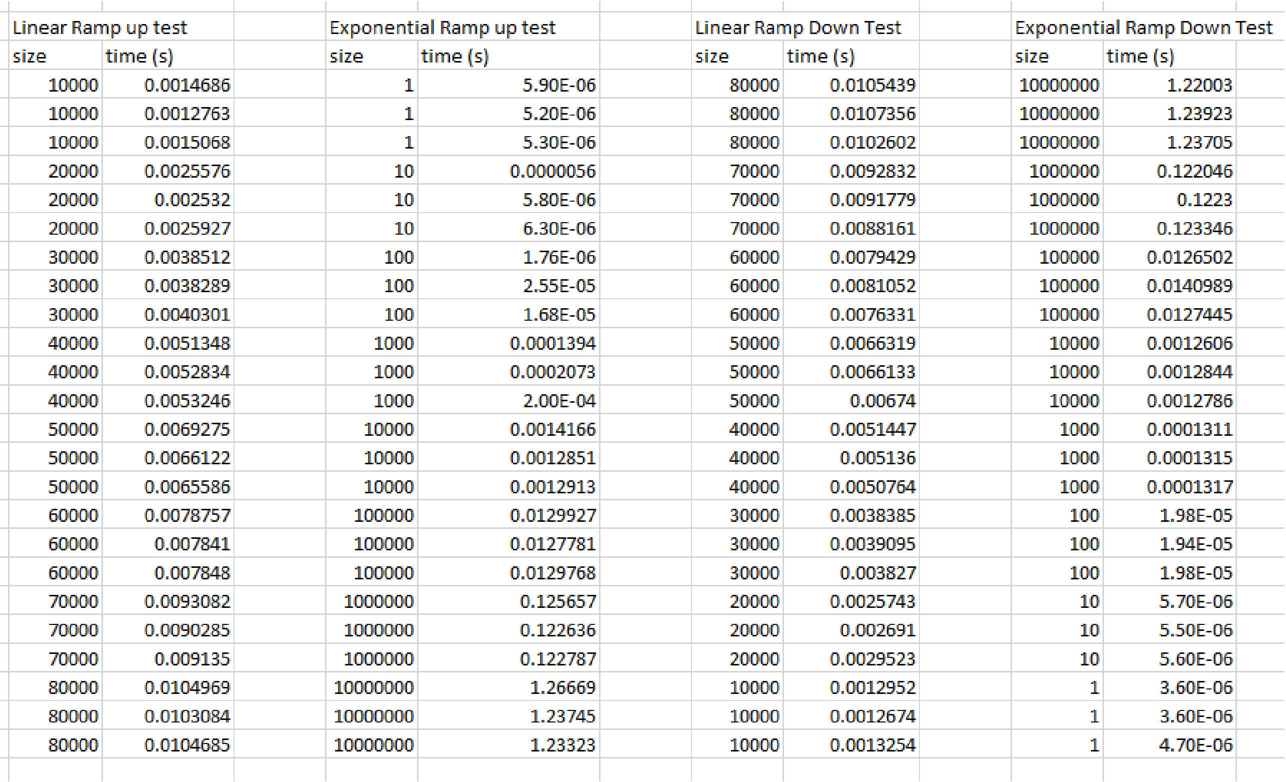


```
// 1. get start time
auto start = steady_clock::now();
// 2. do some work (create, fill, find sum)
vector<int> v(size, 42);
total = accumulate(v.begin(), v.end(), 0u);
// 3. show duration time
auto end = steady_clock::now();
duration<double> diff = end - start;
```

Ramp tests change the variable size to determine how the operation will perform with different variable sizes. Ramp up tests increase the size while ramp down tests decrease it.

An exponential ramping test changes the size in exponential increments. This can help us notice difference in performance that may not be clear at smaller when we change the size linearly.

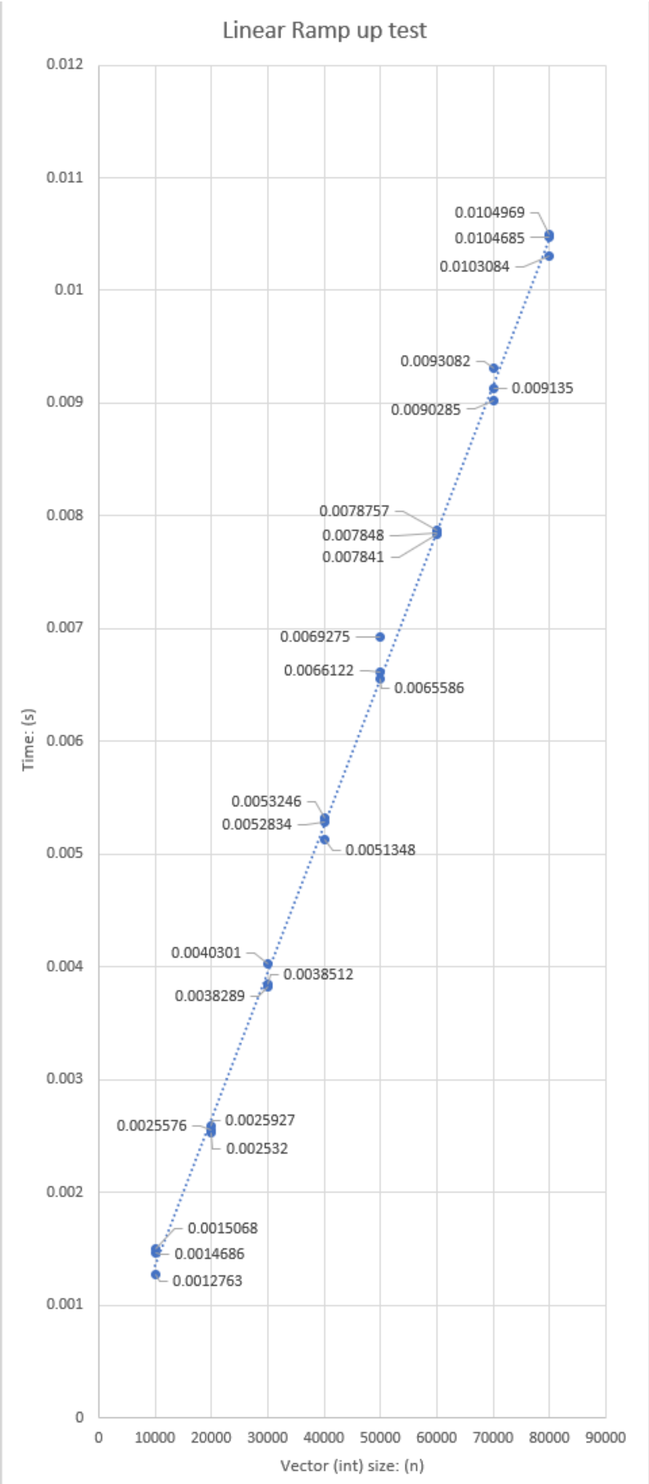
Using the test code provided I ran each test 3 times and recorded some data on linear and exponential ramp up and down tests. I've recorded multiple times for the same size as the time can vary. The computer is running other tasks at the same time so depending on how intensive those tasks are the time to execute will be longer or shorter.

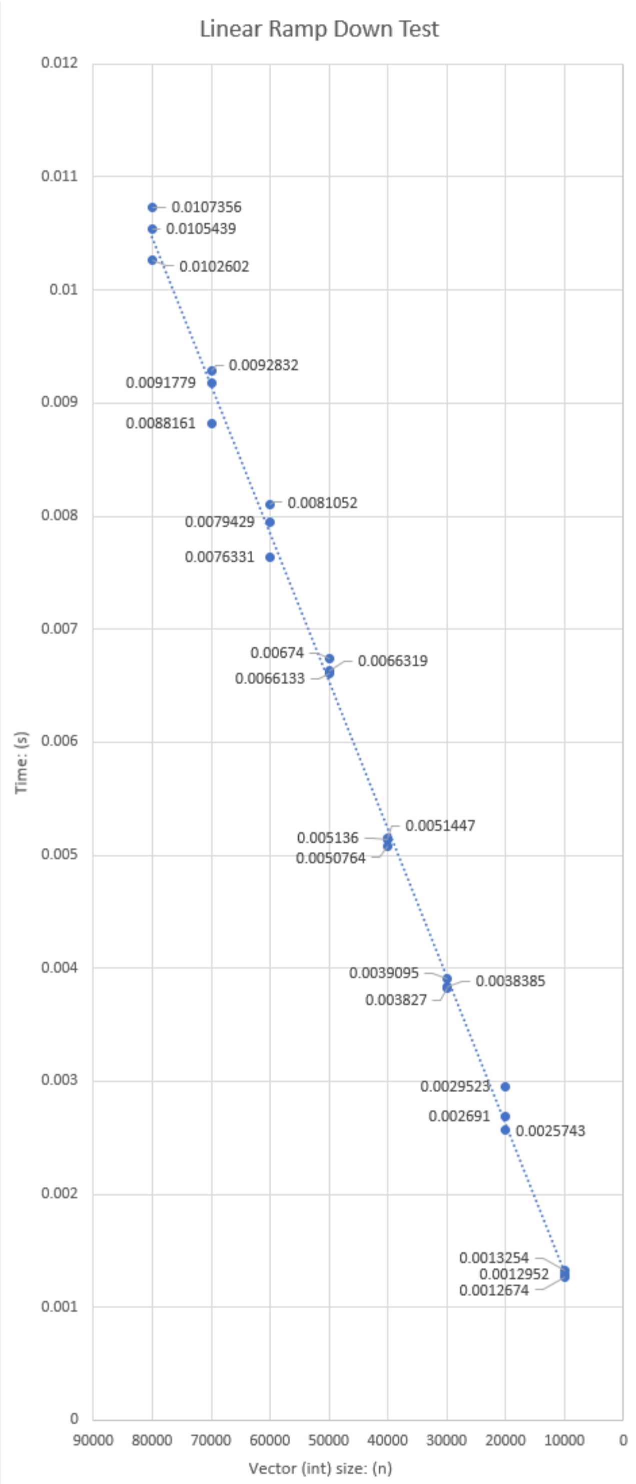


Linear Ramp up test		Exponential Ramp up test		Linear Ramp Down Test		Exponential Ramp Down Test	
size	time (s)	size	time (s)	size	time (s)	size	time (s)
10000	0.0014686	1	5.90E-06	80000	0.0105439	10000000	1.22003
10000	0.0012763	1	5.20E-06	80000	0.0107356	10000000	1.23923
10000	0.0015068	1	5.30E-06	80000	0.0102602	10000000	1.23705
20000	0.0025576	10	0.0000056	70000	0.0092832	1000000	0.122046
20000	0.002532	10	5.80E-06	70000	0.0091779	1000000	0.1223
20000	0.0025927	10	6.30E-06	70000	0.0088161	1000000	0.123346
30000	0.0038512	100	1.76E-06	60000	0.0079429	100000	0.0126502
30000	0.0038289	100	2.55E-05	60000	0.0081052	100000	0.0140989
30000	0.0040301	100	1.68E-05	60000	0.0076331	100000	0.0127445
40000	0.0051348	1000	0.0001394	50000	0.0066319	10000	0.0012606
40000	0.0052834	1000	0.0002073	50000	0.0066133	10000	0.0012844
40000	0.0053246	1000	2.00E-04	50000	0.00674	10000	0.0012786
50000	0.0069275	10000	0.0014166	40000	0.0051447	1000	0.0001311
50000	0.0066122	10000	0.0012851	40000	0.005136	1000	0.0001315
50000	0.0065586	10000	0.0012913	40000	0.0050764	1000	0.0001317
60000	0.0078757	100000	0.0129927	30000	0.0038385	100	1.98E-05
60000	0.007841	100000	0.0127781	30000	0.0039095	100	1.94E-05
60000	0.007848	100000	0.0129768	30000	0.003827	100	1.98E-05
70000	0.0093082	1000000	0.125657	20000	0.0025743	10	5.70E-06
70000	0.0090285	1000000	0.122636	20000	0.002691	10	5.50E-06
70000	0.009135	1000000	0.122787	20000	0.0029523	10	5.60E-06
80000	0.0104969	10000000	1.26669	10000	0.0012952	1	3.60E-06
80000	0.0103084	10000000	1.23745	10000	0.0012674	1	3.60E-06
80000	0.0104685	10000000	1.23323	10000	0.0013254	1	4.70E-06

It's only a small data size but we can see that at each size we are getting roughly the same time with no outliers that are much bigger than the other times.

If we graph the data, we can have a look at how the time taken to perform the function changes with the size of the vector.

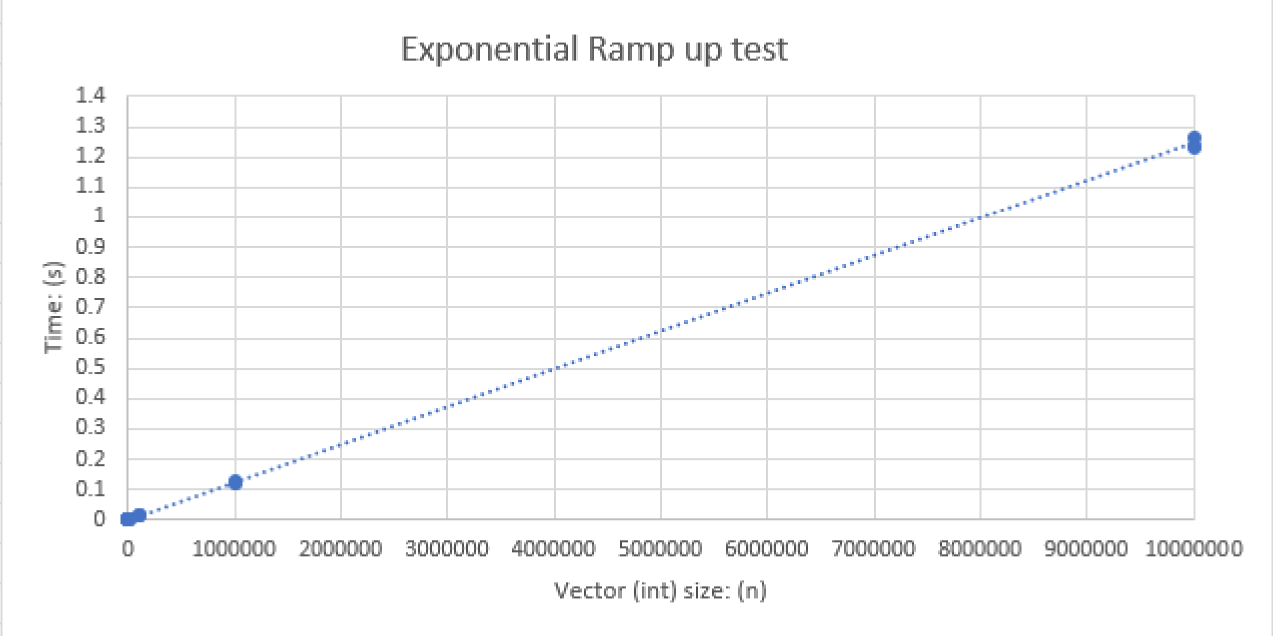


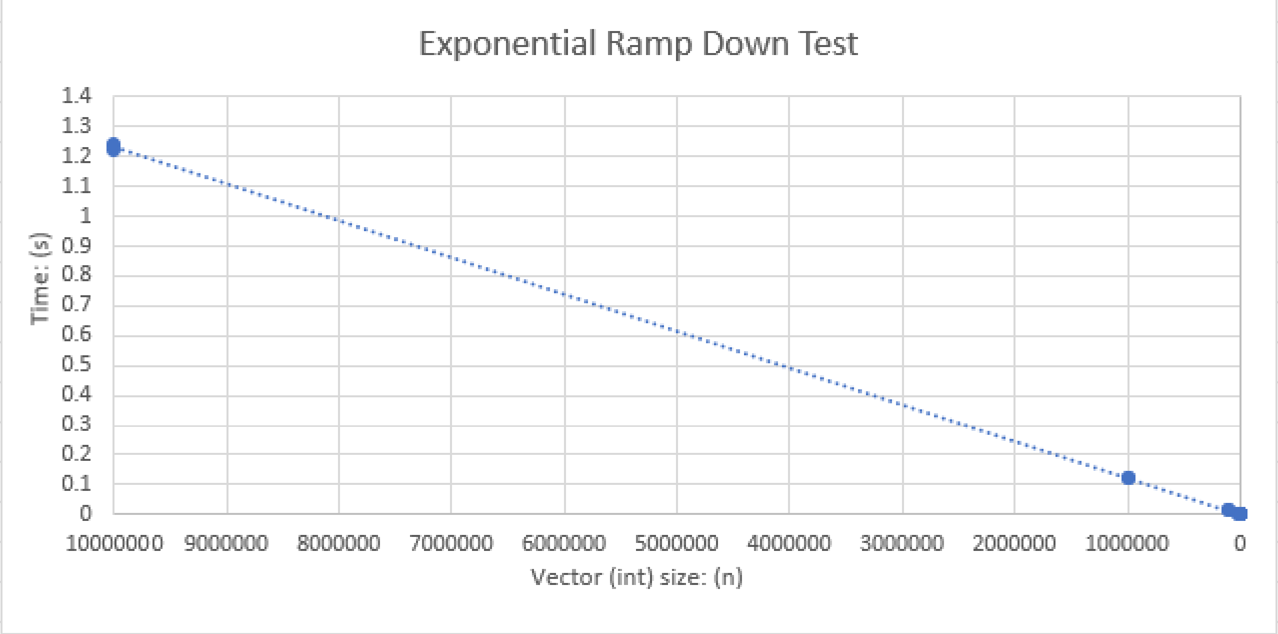


Looking at both charts for the linear ramp tests it seems like the time it takes to execute the function increases linearly. It also gives us a good look at the outliers compared to the linear trend.

However, the increments we are using are quite small. Sometimes we may find that with larger values the time it takes to complete tasks will become much bigger as they get more complex.

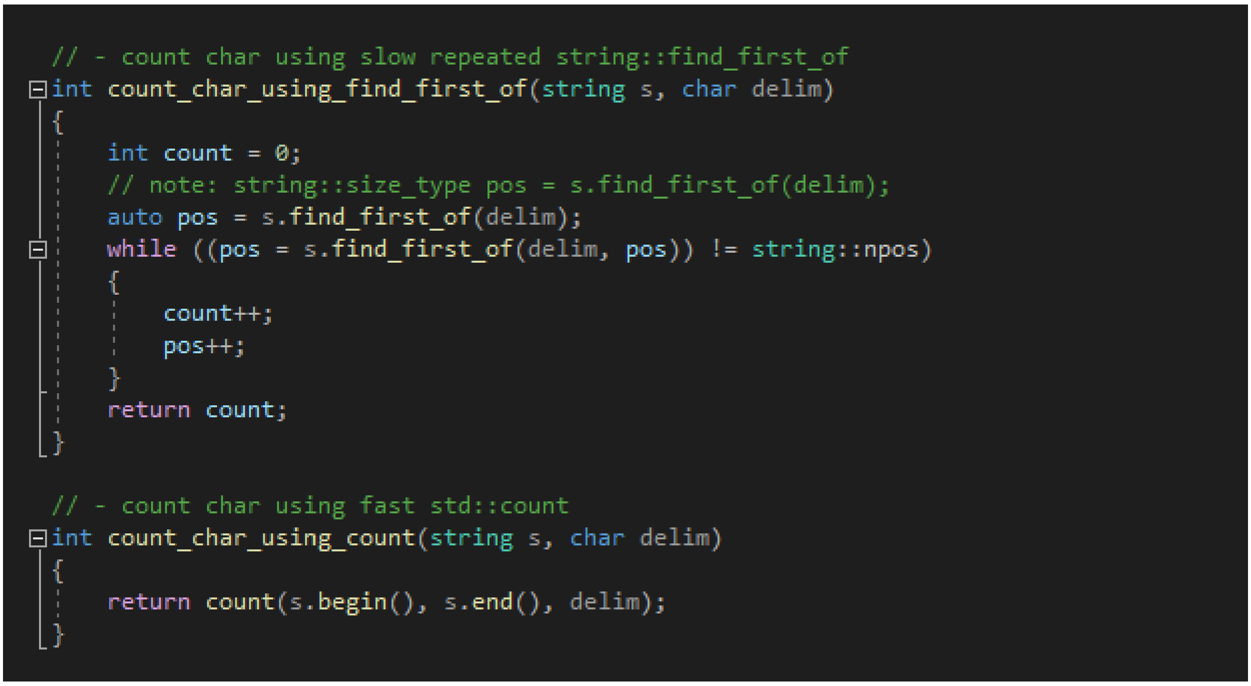
If we look at the charts for the exponential ramp tests, we can see that the relationship between vector size and function execution time is still linear. Though the time and size get so big that we can’t really see the other smaller points of data. That’s ok though as we can confirm that the execution time increases linearly. So, if we wanted to look into the variance we could test a different set of data that would help us find that.





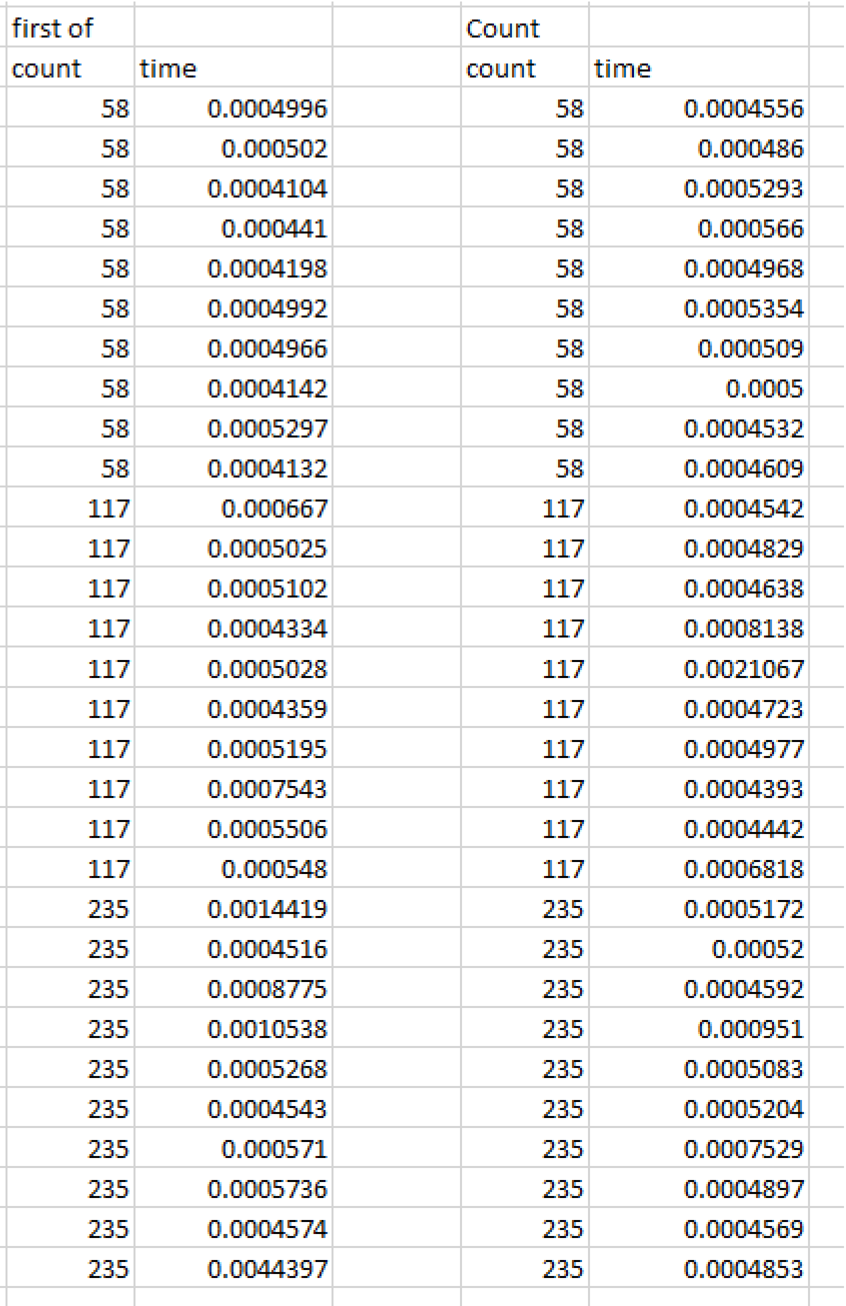
We can use these ideas to compare the performance of different functions as well.

Let’s compare these two functions.



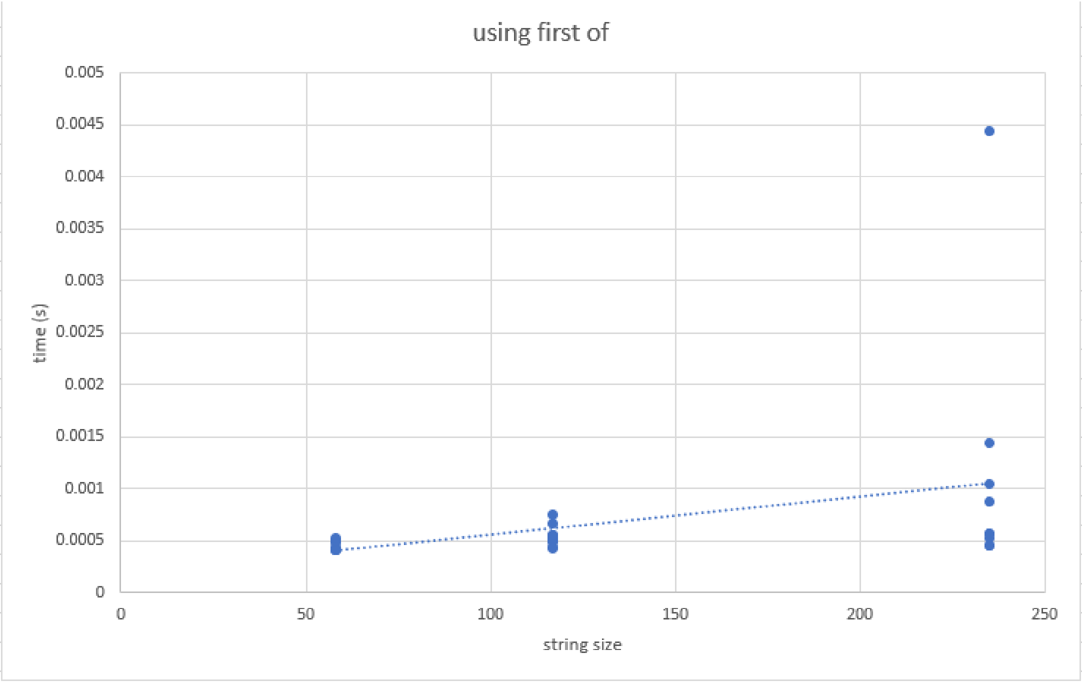
```cpp
// - count char using slow repeated string::find_first_of
int count_char_using_find_first_of(string s, char delim)
{
    int count = 0;
    // note: string::size_type pos = s.find_first_of(delim);
    auto pos = s.find_first_of(delim);
    while ((pos = s.find_first_of(delim, pos)) != string::npos)
    {
        count++;
        pos++;
    }
    return count;
}

// - count char using fast std::count
int count_char_using_count(string s, char delim)
{
    return count(s.begin(), s.end(), delim);
}
```

I tested each one with 3 different string sizes to get some sample data.

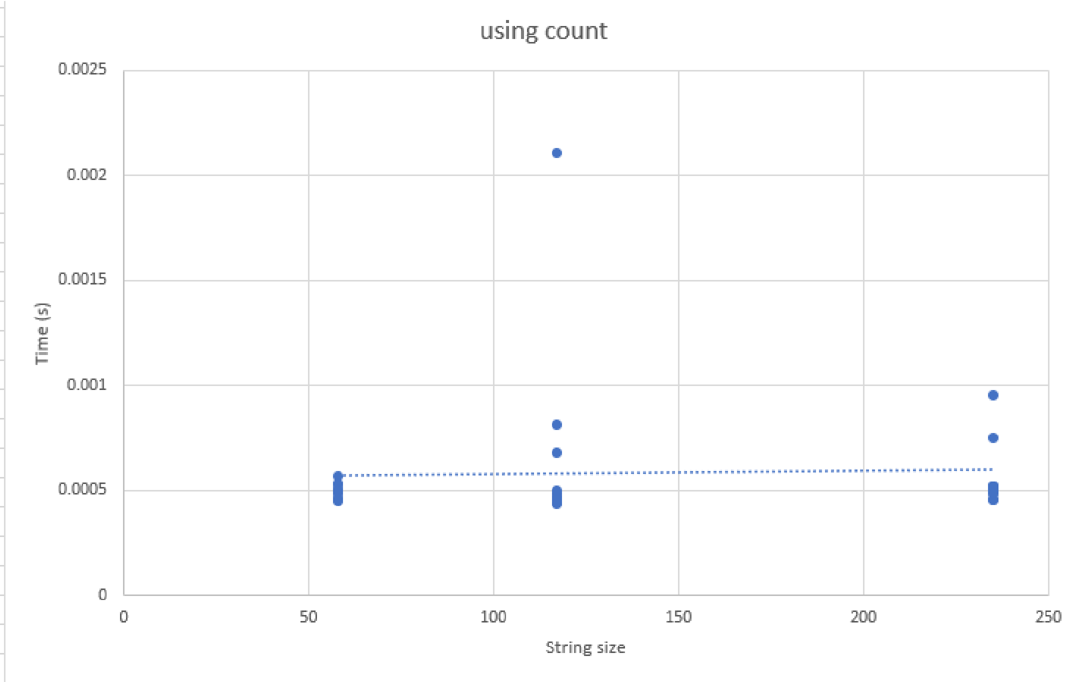


first of			Count	
count	time		count	time
58	0.0004996		58	0.0004556
58	0.000502		58	0.000486
58	0.0004104		58	0.0005293
58	0.000441		58	0.000566
58	0.0004198		58	0.0004968
58	0.0004992		58	0.0005354
58	0.0004966		58	0.000509
58	0.0004142		58	0.0005
58	0.0005297		58	0.0004532
58	0.0004132		58	0.0004609
117	0.000667		117	0.0004542
117	0.0005025		117	0.0004829
117	0.0005102		117	0.0004638
117	0.0004334		117	0.0008138
117	0.0005028		117	0.0021067
117	0.0004359		117	0.0004723
117	0.0005195		117	0.0004977
117	0.0007543		117	0.0004393
117	0.0005506		117	0.0004442
117	0.000548		117	0.0006818
235	0.0014419		235	0.0005172
235	0.0004516		235	0.00052
235	0.0008775		235	0.0004592
235	0.0010538		235	0.000951
235	0.0005268		235	0.0005083
235	0.0004543		235	0.0005204
235	0.000571		235	0.0007529
235	0.0005736		235	0.0004897
235	0.0004574		235	0.0004569
235	0.0044397		235	0.0004853

The time for each size seem pretty random when looking at them by eye.

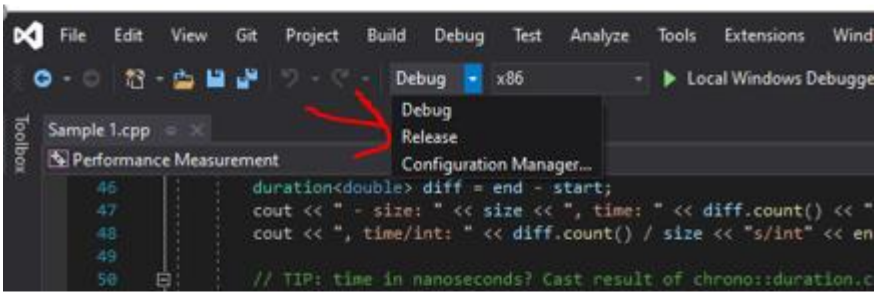


If we look at the first of results chart, we can see some pretty interesting stuff though. To start off the data isn't very varied at all. But as the string size gets bigger the time becomes a lot more varied. The trendline suggests that as the size increases, the possible execution time becomes higher.



The using count chart also indicates that variance occurs as the sting size increases. However, it doesn't increase by nearly as much. The trendline stays mostly flat.

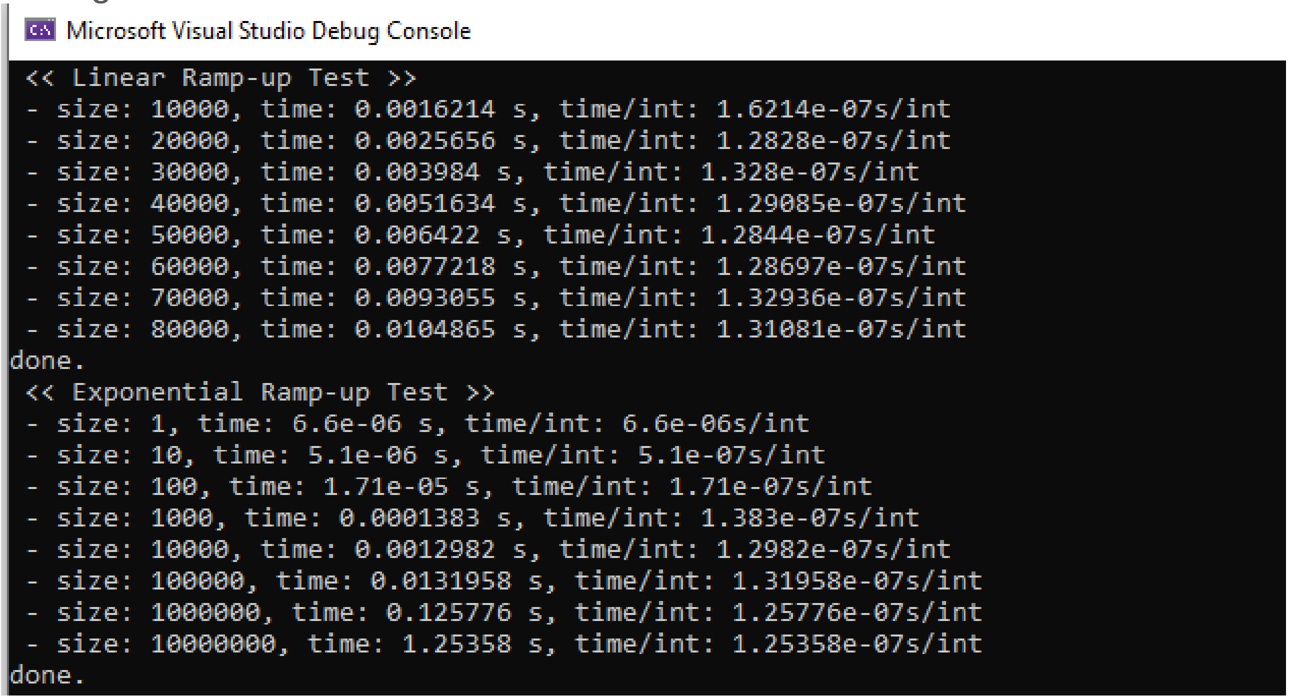
It’s also interesting to note the two outliers in both charts. For some reason that time it took way longer to execute. There must have been something else running in the background which caused it to take longer than usual. Perhaps the outliers are also slightly skewing the trendline? If we wanted to find out we would need a larger sample size.



```
duration<double> diff = end - start;
cout << " - size: " << size << ", time: " << diff.count() << "
cout << ", time/int: " << diff.count() / size << "s/int" << en

// TIP: time in nanoseconds? Cast result of chrono::duration.c
```

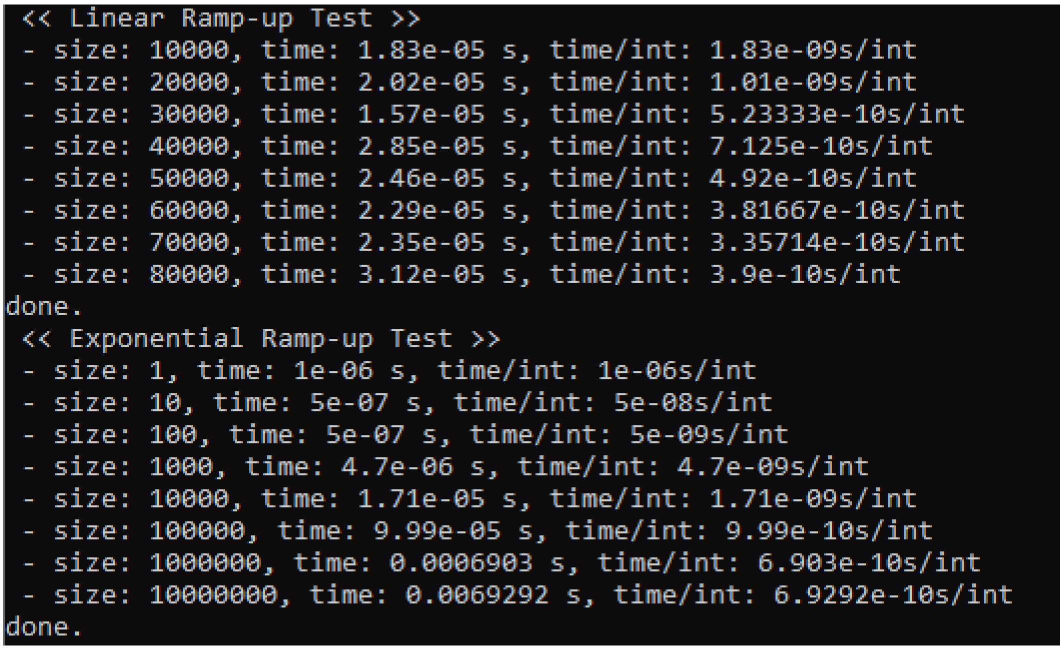
Another thing that can cause execution time to take longer is using debug mode vs release mode. Debug mode is helpful as it can help us catch errors in our code as we’re writing it. But these checks take time to process so they increase the execution time. We can easily see the difference by switching to release mode and comparing the values.

Debug mode:



```
 << Linear Ramp-up Test >>
 - size: 10000, time: 0.0016214 s, time/int: 1.6214e-07s/int
 - size: 20000, time: 0.0025656 s, time/int: 1.2828e-07s/int
 - size: 30000, time: 0.003984 s, time/int: 1.328e-07s/int
 - size: 40000, time: 0.0051634 s, time/int: 1.29085e-07s/int
 - size: 50000, time: 0.006422 s, time/int: 1.2844e-07s/int
 - size: 60000, time: 0.0077218 s, time/int: 1.28697e-07s/int
 - size: 70000, time: 0.0093055 s, time/int: 1.32936e-07s/int
 - size: 80000, time: 0.0104865 s, time/int: 1.31081e-07s/int
done.
 << Exponential Ramp-up Test >>
 - size: 1, time: 6.6e-06 s, time/int: 6.6e-06s/int
 - size: 10, time: 5.1e-06 s, time/int: 5.1e-07s/int
 - size: 100, time: 1.71e-05 s, time/int: 1.71e-07s/int
 - size: 1000, time: 0.0001383 s, time/int: 1.383e-07s/int
 - size: 10000, time: 0.0012982 s, time/int: 1.2982e-07s/int
 - size: 100000, time: 0.0131958 s, time/int: 1.31958e-07s/int
 - size: 1000000, time: 0.125776 s, time/int: 1.25776e-07s/int
 - size: 10000000, time: 1.25358 s, time/int: 1.25358e-07s/int
done.
```

Release Mode:

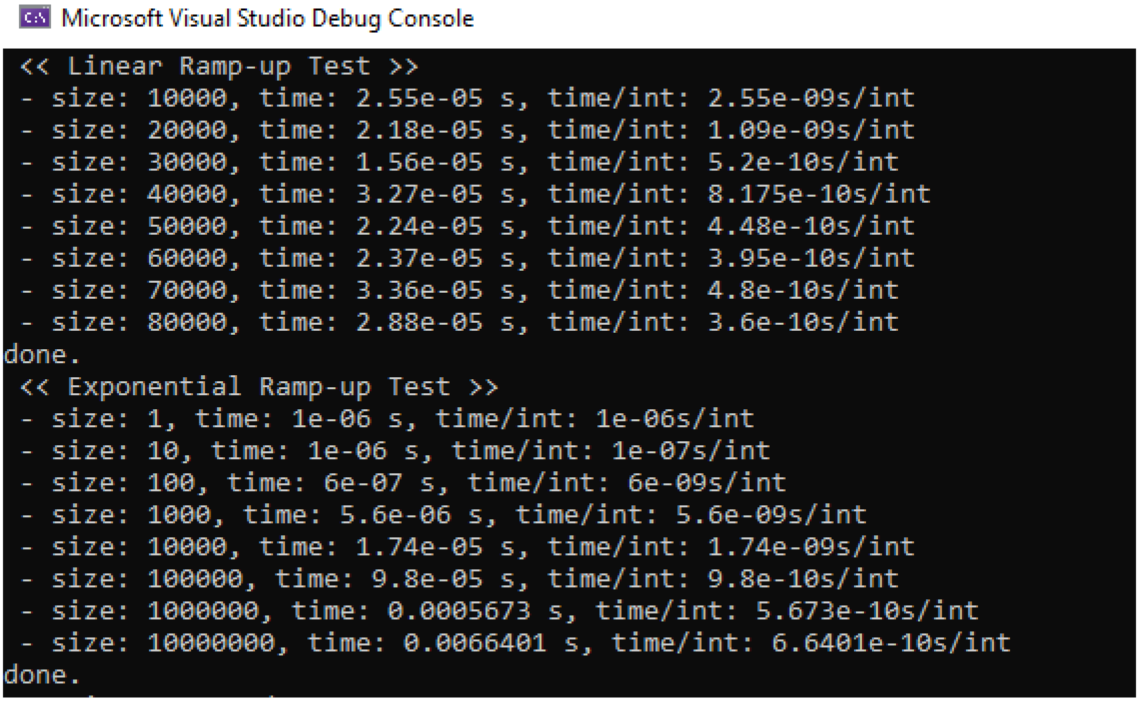


```
 << Linear Ramp-up Test >>
 - size: 10000, time: 1.83e-05 s, time/int: 1.83e-09s/int
 - size: 20000, time: 2.02e-05 s, time/int: 1.01e-09s/int
 - size: 30000, time: 1.57e-05 s, time/int: 5.23333e-10s/int
 - size: 40000, time: 2.85e-05 s, time/int: 7.125e-10s/int
 - size: 50000, time: 2.46e-05 s, time/int: 4.92e-10s/int
 - size: 60000, time: 2.29e-05 s, time/int: 3.81667e-10s/int
 - size: 70000, time: 2.35e-05 s, time/int: 3.35714e-10s/int
 - size: 80000, time: 3.12e-05 s, time/int: 3.9e-10s/int
done.
 << Exponential Ramp-up Test >>
 - size: 1, time: 1e-06 s, time/int: 1e-06s/int
 - size: 10, time: 5e-07 s, time/int: 5e-08s/int
 - size: 100, time: 5e-07 s, time/int: 5e-09s/int
 - size: 1000, time: 4.7e-06 s, time/int: 4.7e-09s/int
 - size: 10000, time: 1.71e-05 s, time/int: 1.71e-09s/int
 - size: 100000, time: 9.99e-05 s, time/int: 9.99e-10s/int
 - size: 1000000, time: 0.0006903 s, time/int: 6.903e-10s/int
 - size: 10000000, time: 0.0069292 s, time/int: 6.9292e-10s/int
done.
```

The times are much faster.

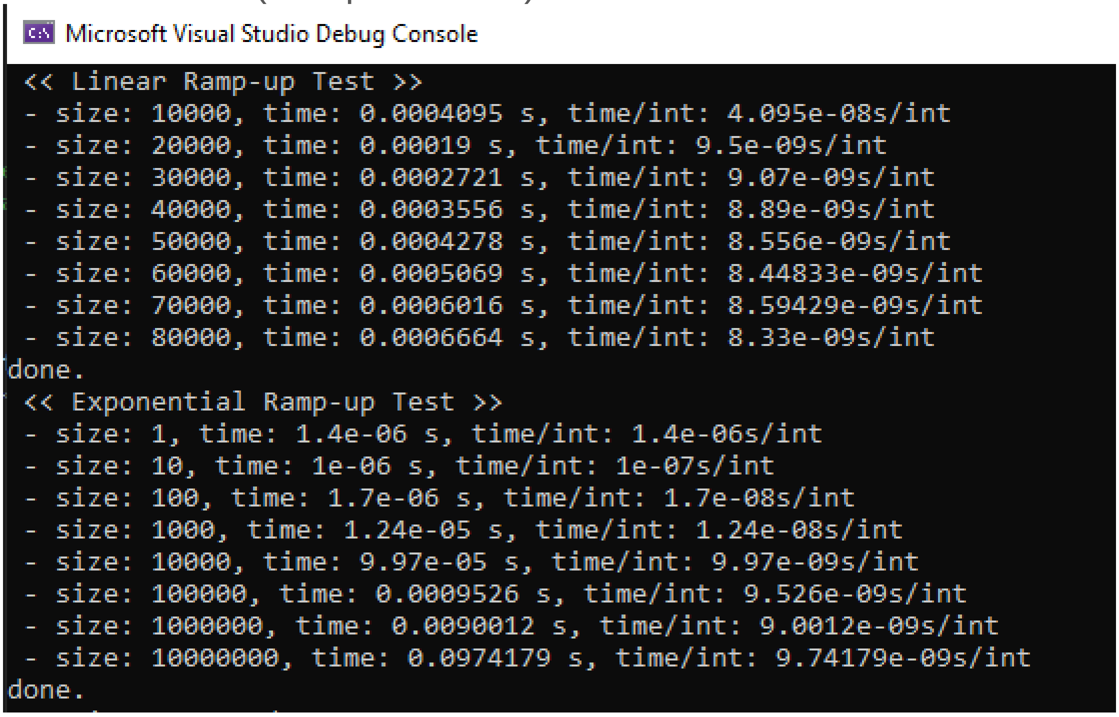
Another thing that affects execution is compiler optimisation. When code is compiled, it often has optimizations made by the compiler to improve performance. If we were to turn these optimizations off the execution time would become much slower.

Release mode (Optimization):



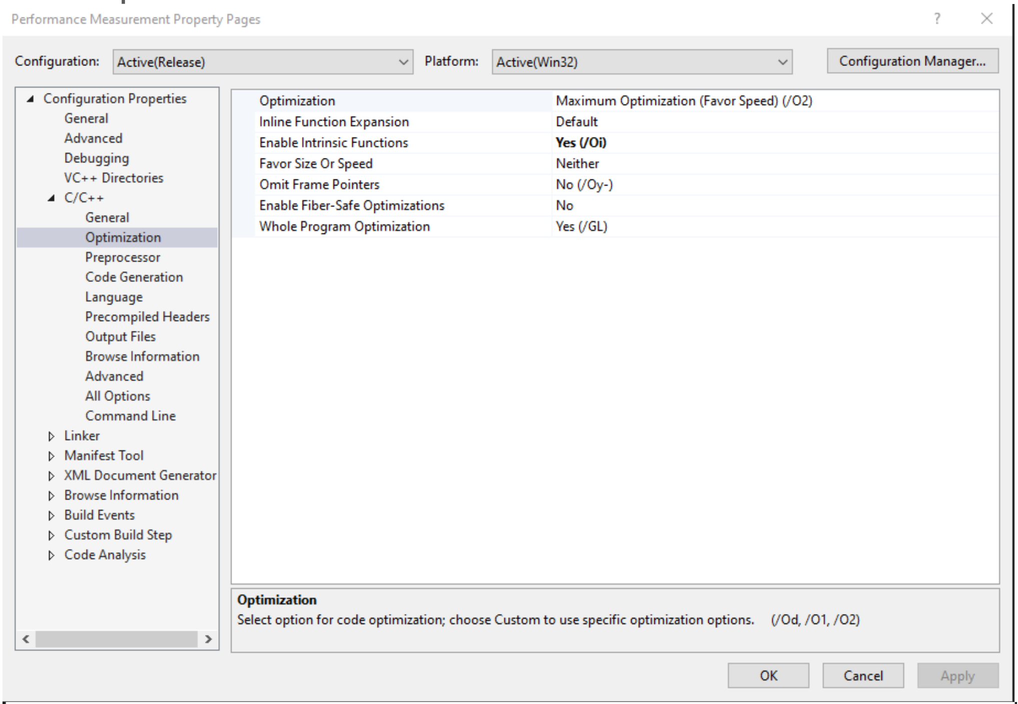
```
 << Linear Ramp-up Test >>
 - size: 10000, time: 2.55e-05 s, time/int: 2.55e-09s/int
 - size: 20000, time: 2.18e-05 s, time/int: 1.09e-09s/int
 - size: 30000, time: 1.56e-05 s, time/int: 5.2e-10s/int
 - size: 40000, time: 3.27e-05 s, time/int: 8.175e-10s/int
 - size: 50000, time: 2.24e-05 s, time/int: 4.48e-10s/int
 - size: 60000, time: 2.37e-05 s, time/int: 3.95e-10s/int
 - size: 70000, time: 3.36e-05 s, time/int: 4.8e-10s/int
 - size: 80000, time: 2.88e-05 s, time/int: 3.6e-10s/int
done.
 << Exponential Ramp-up Test >>
 - size: 1, time: 1e-06 s, time/int: 1e-06s/int
 - size: 10, time: 1e-06 s, time/int: 1e-07s/int
 - size: 100, time: 6e-07 s, time/int: 6e-09s/int
 - size: 1000, time: 5.6e-06 s, time/int: 5.6e-09s/int
 - size: 10000, time: 1.74e-05 s, time/int: 1.74e-09s/int
 - size: 100000, time: 9.8e-05 s, time/int: 9.8e-10s/int
 - size: 1000000, time: 0.0005673 s, time/int: 5.673e-10s/int
 - size: 10000000, time: 0.0066401 s, time/int: 6.6401e-10s/int
done.
```

Release mode (No optimization):



```
 << Linear Ramp-up Test >>
 - size: 10000, time: 0.0004095 s, time/int: 4.095e-08s/int
 - size: 20000, time: 0.00019 s, time/int: 9.5e-09s/int
 - size: 30000, time: 0.0002721 s, time/int: 9.07e-09s/int
 - size: 40000, time: 0.0003556 s, time/int: 8.89e-09s/int
 - size: 50000, time: 0.0004278 s, time/int: 8.556e-09s/int
 - size: 60000, time: 0.0005069 s, time/int: 8.44833e-09s/int
 - size: 70000, time: 0.0006016 s, time/int: 8.59429e-09s/int
 - size: 80000, time: 0.0006664 s, time/int: 8.33e-09s/int
done.
 << Exponential Ramp-up Test >>
 - size: 1, time: 1.4e-06 s, time/int: 1.4e-06s/int
 - size: 10, time: 1e-06 s, time/int: 1e-07s/int
 - size: 100, time: 1.7e-06 s, time/int: 1.7e-08s/int
 - size: 1000, time: 1.24e-05 s, time/int: 1.24e-08s/int
 - size: 10000, time: 9.97e-05 s, time/int: 9.97e-09s/int
 - size: 100000, time: 0.0009526 s, time/int: 9.526e-09s/int
 - size: 1000000, time: 0.0090012 s, time/int: 9.0012e-09s/int
 - size: 10000000, time: 0.0974179 s, time/int: 9.74179e-09s/int
done.
```

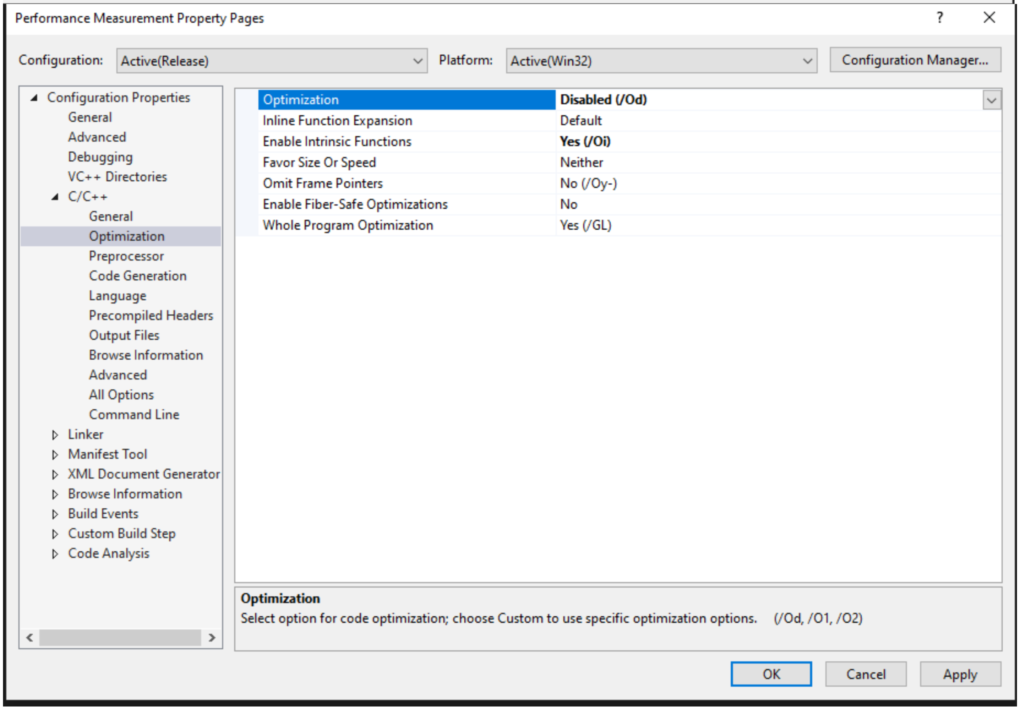
The times with no optimization are much slower, To change optimizations in Visual Studio right click on the project and select properties. Then go to: Configuration Properties -> C/C++ -> Optimization. Then under the optimization tab select the optimization setting you want from the dropdown menu.



Configuration: Active(Release) Platform: Active(Win32)

Optimization	Maximum Optimization (Favor Speed) (/O2)
Inline Function Expansion	Default
Enable Intrinsic Functions	**Yes (/Oi)**
Favor Size Or Speed	Neither
Omit Frame Pointers	No (/Oy-)
Enable Fiber-Safe Optimizations	No
Whole Program Optimization	Yes (/GL)

Optimization
Select option for code optimization; choose Custom to use specific optimization options. (/Od, /O1, /O2)



Configuration: Active(Release) Platform: Active(Win32)

Optimization	**Disabled (/Od)**
Inline Function Expansion	Default
Enable Intrinsic Functions	**Yes (/Oi)**
Favor Size Or Speed	Neither
Omit Frame Pointers	No (/Oy-)
Enable Fiber-Safe Optimizations	No
Whole Program Optimization	Yes (/GL)

Optimization
Select option for code optimization; choose Custom to use specific optimization options. (/Od, /O1, /O2)