

Spike : 8

Title: Performance Optimizations

Author: Parth Madhani, 101901858

Technologies, Tools, and Resources used:

- Visual Studio IDE
- Canvas

Goals / deliverables:

- Code
- Debugging the sample codes and analysing ramp up/ramp down tests.

Tasks undertaken:

- Applying the skills learnt from previous spike, to fix/analyse the code using the debugger.

What we found out:

Ramp Up Tests:

First we check linear and then we check exponentially which has significant time difference due to size in exponential being less than that in linear.

```
C:\Users\murty\source\repos\sample1\Debug\sample1.exe
<< Linear Ramp-up Test >>
- size: 10000, time: 2302900 s, time/int: 230s/int
- size: 20000, time: 3514400 s, time/int: 175s/int
- size: 30000, time: 5277300 s, time/int: 175s/int
- size: 40000, time: 7098100 s, time/int: 177s/int
- size: 50000, time: 9225800 s, time/int: 184s/int
done.
<< Exponential Ramp-up Test >>
- size: 1, time: 15700 s, time/int: 15700s/int
- size: 100, time: 51300 s, time/int: 513s/int
- size: 10000, time: 1423200 s, time/int: 142s/int
- size: 1000000, time: 157485600 s, time/int: 157s/int
- size: 100000000, time: 13068081900 s, time/int: 130s/int
done.
result: 5
result: 5
```

Size	Exponential time taken (in nanosec)	Linear Time taken(in nanosec)
Tier 1	15700	2302900
Tier 2	51300	3514400
Tier 3	1423200	5277300
Tier 4	157485600	7098100
Tier 5	13068081900	9225800

First run of program

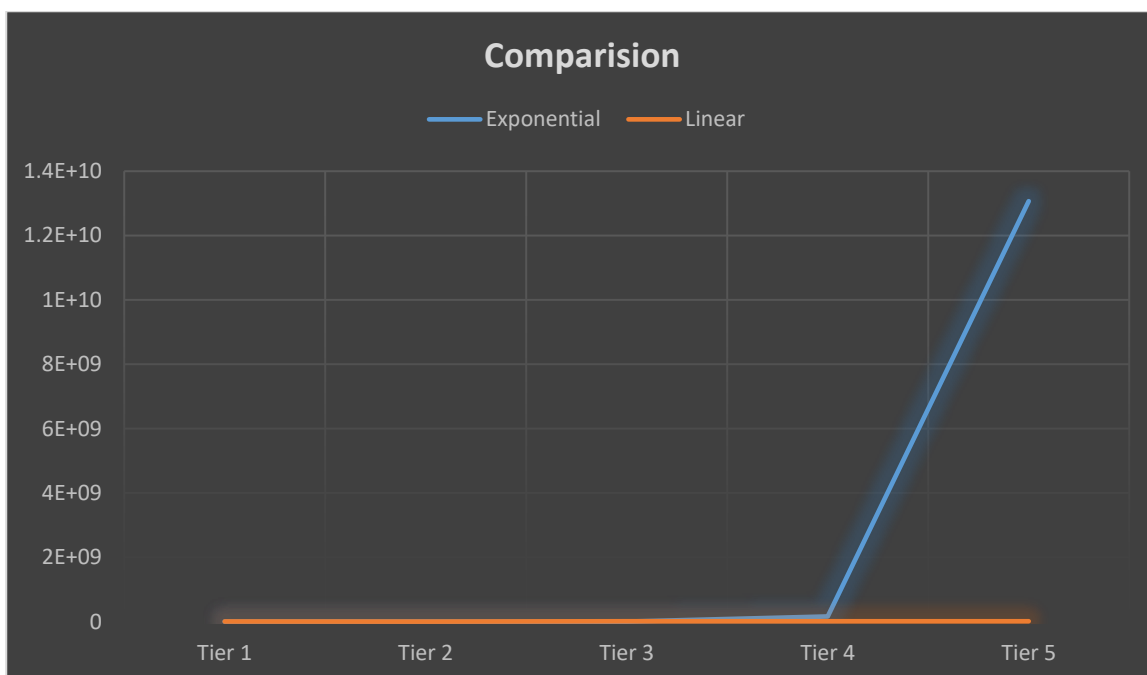
As we can deduce from the first run, time taken by exponential for same 1000 which is tier 1 for linear and tier 3 for exponential is different which clearly shows that exponential takes much less time while linear takes 230ns/ int when size 10000 with linear just taking 142ns/int.

Other than that exponential time taken gradually increases like linear time. As size increases the speed increases too as we see that for exponential time once size reaches its highest point time taken per int decreases all the way to 130ns/int which was 15700s/int for just 1 size.

```
C:\Users\murty\source\repos\sample1\Debug\sample1.exe
<< Linear Ramp-up Test >>
- size: 10000, time: 1428000 s, time/int: 142s/int
- size: 20000, time: 2978400 s, time/int: 148s/int
- size: 30000, time: 4594600 s, time/int: 153s/int
- size: 40000, time: 6651100 s, time/int: 166s/int
- size: 50000, time: 8421900 s, time/int: 168s/int
done.
<< Exponential Ramp-up Test >>
- size: 1, time: 16500 s, time/int: 16500s/int
- size: 100, time: 28200 s, time/int: 282s/int
- size: 10000, time: 1772200 s, time/int: 177s/int
- size: 1000000, time: 140707700 s, time/int: 140s/int
- size: 100000000, time: 13604359300 s, time/int: 136s/int
done.
result: 5
result: 5
```

2nd time program running

2nd time results were similar but time differs everytime we run the program with it being similar but never same.



Comparison between ramp up and ramp down and Repeatability

When it comes to ramp down test the time taken for rampdown test for 10000 size is less then that for ramp up test. For instance time taken for ramp up test for looping 10000 time size was 128ns/int for first try while ramp down just took 108ns/int which is way less.

C:\Users\murty\source\repos\sample1\Debug\sample1.exe

```
<< Particle Ramp-up Test >>
- size: 1, time: 8000 ns (ns/count): 8000
- size: 10, time: 4900 ns (ns/count): 490
- size: 100, time: 12400 ns (ns/count): 124
- size: 1000, time: 110600 ns (ns/count): 110
- size: 10000, time: 1285900 ns (ns/count): 128
done.
<< Particle Ramp-down Test >>
- size: 10000, time: 1086700 ns (ns/count): 108
- size: 1000, time: 92300 ns (ns/count): 92
- size: 100, time: 21000 ns (ns/count): 210
- size: 10, time: 8400 ns (ns/count): 840
- size: 1, time: 5200 ns (ns/count): 5200
done.
```

Size	Time taken/int (Ramp UP)	Time taken/int(Ramp DOWN)
1	8000	5200
10	490	840
100	124	210
1000	110	92
10000	128	108

As we can deduce from first try that time taken for higher size for rampdown but for lower size except for size 1 the time taken for ramp up in this case is lesser than that for ramp down.

C:\Users\murty\source\repos\sample1\Debug\sample1.exe

```
<< Particle Ramp-up Test >>
- size: 1, time: 9000 ns (ns/count): 9000
- size: 10, time: 9400 ns (ns/count): 940
- size: 100, time: 12800 ns (ns/count): 128
- size: 1000, time: 124800 ns (ns/count): 124
- size: 10000, time: 1020700 ns (ns/count): 102
done.
<< Particle Ramp-down Test >>
- size: 10000, time: 1197100 ns (ns/count): 119
- size: 1000, time: 89700 ns (ns/count): 89
- size: 100, time: 13300 ns (ns/count): 133
- size: 10, time: 4600 ns (ns/count): 460
- size: 1, time: 3200 ns (ns/count): 3200
done.
```

2nd try

As we can see for second try the results were significantly different with this time ramp down having lower time taken for almost all cases except for when there is lower size.

C:\Users\murty\source\repos\sample1\Debug\sample1.exe

```
<< Particle Ramp-up Test >>
- size: 1, time: 21800 ns (ns/count): 21800
- size: 10, time: 4600 ns (ns/count): 460
- size: 100, time: 13400 ns (ns/count): 134
- size: 1000, time: 107400 ns (ns/count): 107
- size: 10000, time: 972300 ns (ns/count): 97
done.
<< Particle Ramp-down Test >>
- size: 10000, time: 932700 ns (ns/count): 93
- size: 1000, time: 93800 ns (ns/count): 93
- size: 100, time: 12300 ns (ns/count): 123
- size: 10, time: 4300 ns (ns/count): 430
- size: 1, time: 3300 ns (ns/count): 3300
```

3rd try

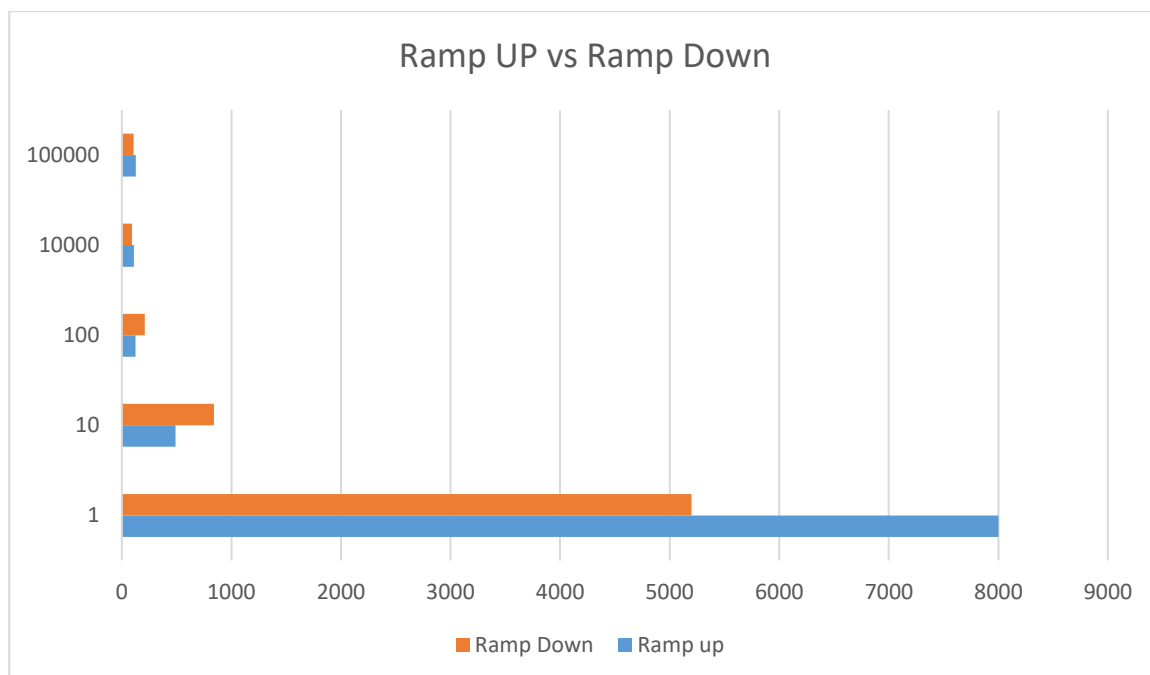
For 3rd try the results are much more similar to 1st try than that of 2nd try with ramp down taking lesser time than ramp up for most cases.

```
C:\Users\murty\source\repos\sample1\Debug\sample1.exe

<< Particle Ramp-up Test >>
- size: 1, time: 12500 ns (ns/count): 12500
- size: 10, time: 22300 ns (ns/count): 2230
- size: 100, time: 33400 ns (ns/count): 334
- size: 1000, time: 151600 ns (ns/count): 151
- size: 10000, time: 1227200 ns (ns/count): 122
done.
<< Particle Ramp-down Test >>
- size: 10000, time: 859800 ns (ns/count): 85
- size: 1000, time: 118100 ns (ns/count): 118
- size: 100, time: 17700 ns (ns/count): 177
- size: 10, time: 4500 ns (ns/count): 450
- size: 1, time: 3300 ns (ns/count): 3300
```

4th try

For 4th try as we can see again rampdown has lesser time than rampup again which means when ramping down the compiler takes lesser time to count while ramping down the compiler gradually goes slower than that for ramp up.



Function Comparison

When comparing the two ways to find total number of character in string we see both of them usually shows up at same time. So to compare more we add chrono start and end time to the functions where it mainly finds the character. This shows us time taken for it to find the character. The results for first try are as follows :

For Sample String : `This is a really simple string but it will do for testing.`

```
sTime taken : 1800 nanoseconds
result: 5
Time taken : 2100 nanoseconds
result: 5
```

When finding total occurrence of 's'

String : `"Thy will be a long o o line o where o the finding character will be at o the end as we wanna check which one be better o when it be like that Example be good like The character we are searching o for is the last o character of alphabets which is zone"`
Finding occurrence of 'z' which is present just once

```
Time taken : 1100 nanoseconds
result: 1
Time taken : 2000 nanoseconds
result: 1
```

As we can see as the number of occurrence of a particular character is lower the time taken for using `find_first_of` is less while using count function it remains more or less the same around 2000-3000 nanoseconds.

When finding o in that string which is like 15 times in that string the time taken for first method is more than that of time taken by using count method which again remains more or less between 2000-3000.

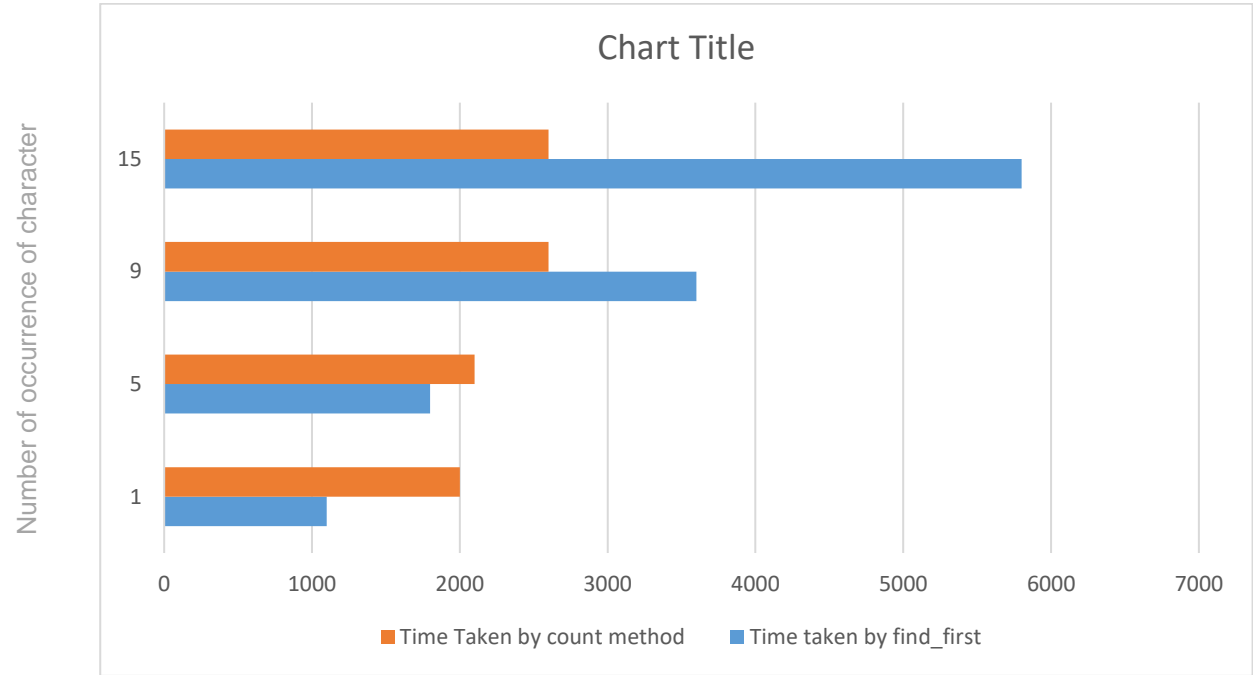
```
C:\Users\murty\source\repos\sample1\Debug\sample1.exe
Time taken : 5800 nanoseconds
result: 15
Time taken : 2600 nanoseconds
result: 15
```

2

So it can easily be seen that by using `find_first_of` the time taken to find the character occurrences increases as the number of occurrences increases while it remains around same by using `count()` method.

So overall after about 10 times running with different string and number of characters we found out that when number is 8-10 the time taken for both methods is usually around same range of 2000-3000.

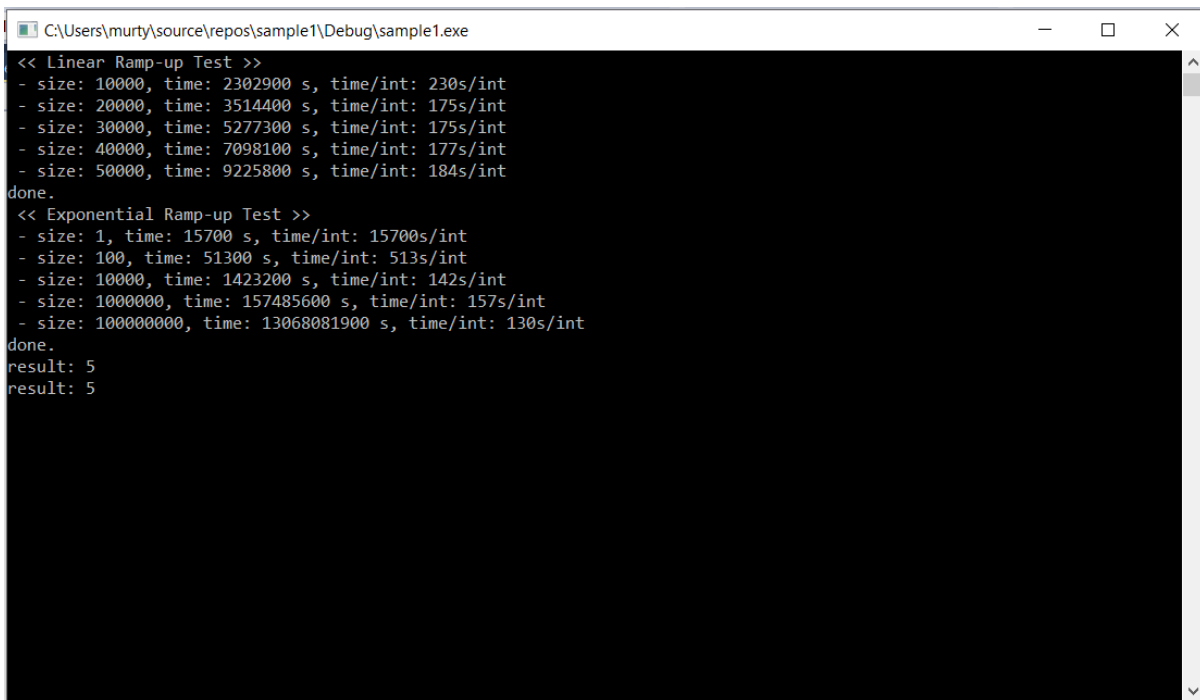
But ye overall if u want to find character using count method will prove to be faster for longer strings with more occurrences of particular character.



IDE Setting

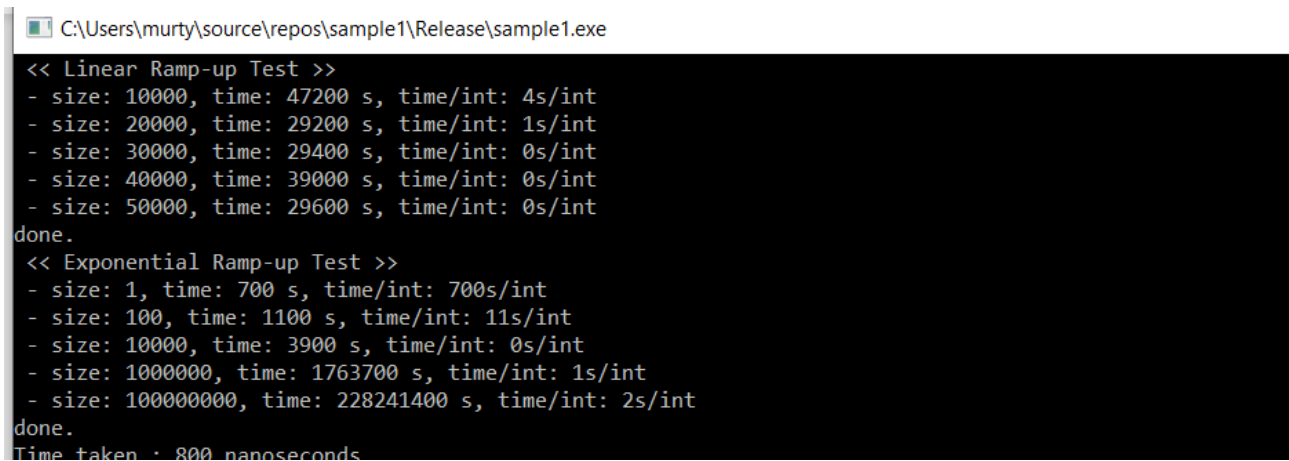
Debug vs Release

Results for Debug :



```
C:\Users\murty\source\repos\sample1\Debug\sample1.exe
<< Linear Ramp-up Test >>
- size: 10000, time: 2302900 s, time/int: 230s/int
- size: 20000, time: 3514400 s, time/int: 175s/int
- size: 30000, time: 5277300 s, time/int: 175s/int
- size: 40000, time: 7098100 s, time/int: 177s/int
- size: 50000, time: 9225800 s, time/int: 184s/int
done.
<< Exponential Ramp-up Test >>
- size: 1, time: 15700 s, time/int: 15700s/int
- size: 100, time: 51300 s, time/int: 513s/int
- size: 10000, time: 1423200 s, time/int: 142s/int
- size: 1000000, time: 157485600 s, time/int: 157s/int
- size: 100000000, time: 13068081900 s, time/int: 130s/int
done.
result: 5
result: 5
```

Result for Release Mode:



```
C:\Users\murty\source\repos\sample1\Release\sample1.exe
<< Linear Ramp-up Test >>
- size: 10000, time: 47200 s, time/int: 4s/int
- size: 20000, time: 29200 s, time/int: 1s/int
- size: 30000, time: 29400 s, time/int: 0s/int
- size: 40000, time: 39000 s, time/int: 0s/int
- size: 50000, time: 29600 s, time/int: 0s/int
done.
<< Exponential Ramp-up Test >>
- size: 1, time: 700 s, time/int: 700s/int
- size: 100, time: 1100 s, time/int: 11s/int
- size: 10000, time: 3900 s, time/int: 0s/int
- size: 1000000, time: 1763700 s, time/int: 1s/int
- size: 100000000, time: 228241400 s, time/int: 2s/int
done.
Time taken : 800 nanoseconds
```

As we can notice from the above output the time taken for debug is way higher while when the program runs in release it will run a lot faster. The difference in execution time is really noticeable. Only other difference being it is easier to debug code and find error if needed in debug mode while harder in release mode so debug mode is ok when the app is still under development while release mode is always faster and better once app is developed and ready to be released.

Compiler Optimization :

By default in visual studio optimization are switched off for debug mode while on for release mode which makes the debug mode lot faster than release mode. So in terms of speed when compiler optimization are turned off the speed should be same as that of debug mode while when on the performance/speed should be better and faster.