Spike: Task07
Title: Performance Measurement

Author: Hayden Whiteford, 104001272

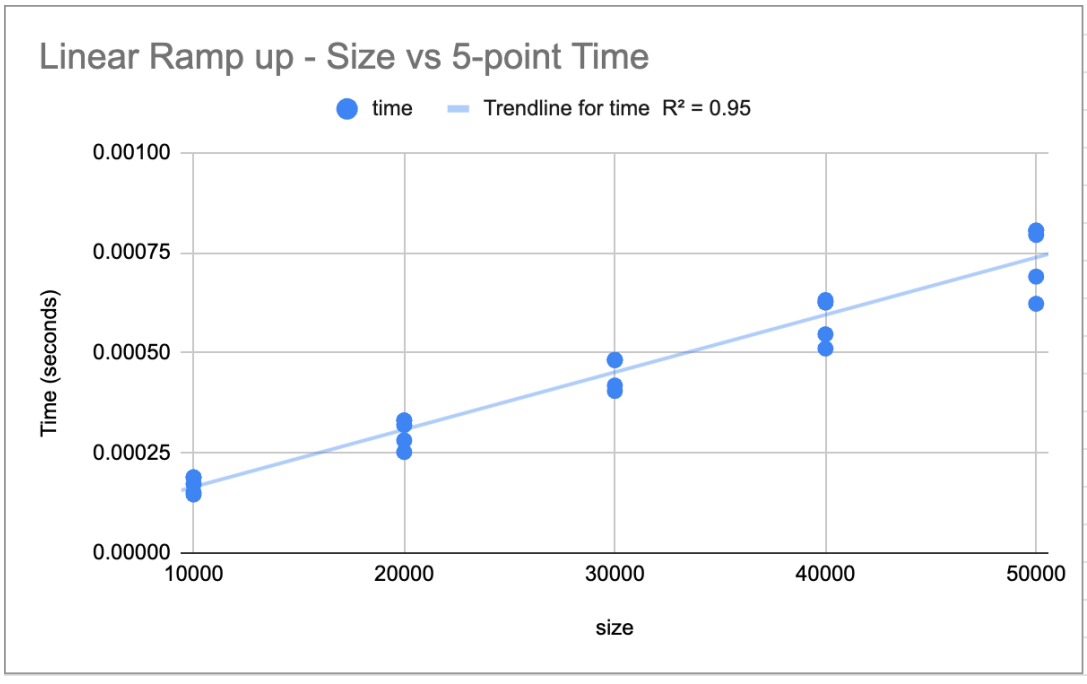
Goals / deliverables:
Demonstrate the following performance and measurement concepts (in any order that matches your work, not the code order necessarily).

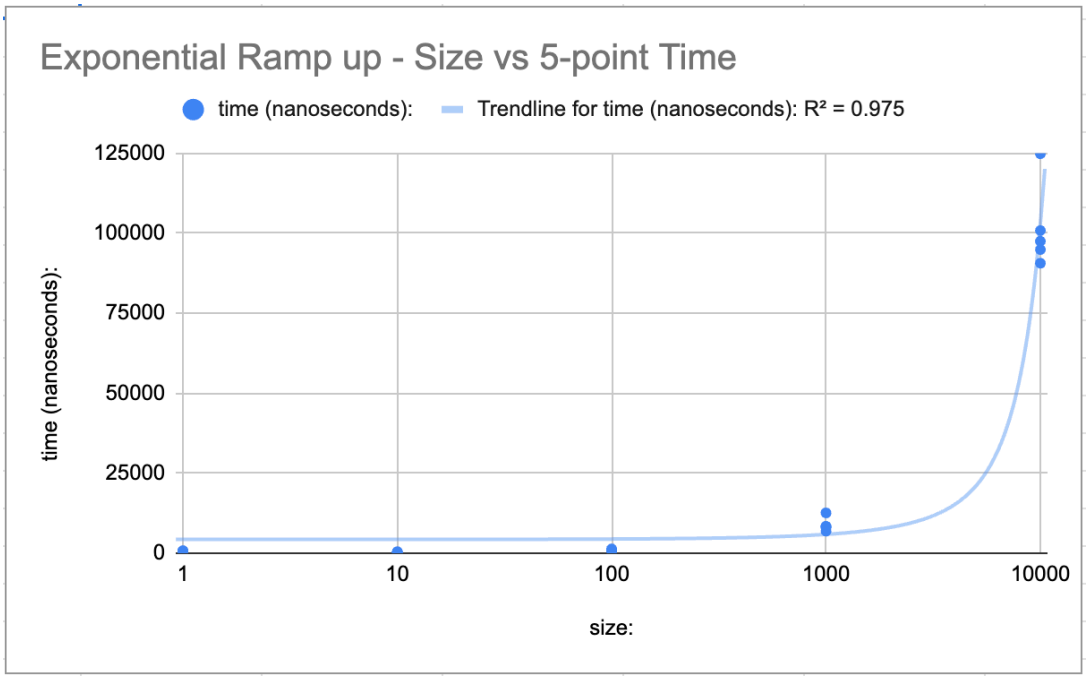
1. **Single Tests**: Demonstrate how to measure both single and multiple function execution time.
2. **Ramp-up Test**: Execute and show, with numbers and **a chart**, both linear and exponential ramp-up testing of
   function execution time. Is there a difference to ramp-down tests?
3. **Repeatability**: Show, with numbers and **a chart**, how ***repeatability*** will vary depending on test conditions.
4. **Function Comparison**: There are two "char in string" counting functions provided (code sample 1). Clearly
   show the difference in performance (if any), and check if the execution time difference is linear with respect to string length (size).
   (Note, you will probably want to create random strings of the various size to test with.)
5. **IDE Settings**: Show what, if any, is the difference in execution time between ***debug*** settings and ***release*** settings. (Remember to have a task that runs for long enough that it matters.)
6. **Compiler Settings:** Turn down/off compiler optimisation and demonstrate a difference. (Make a note of how to do this so that a team member would be able to reproduce your result.)

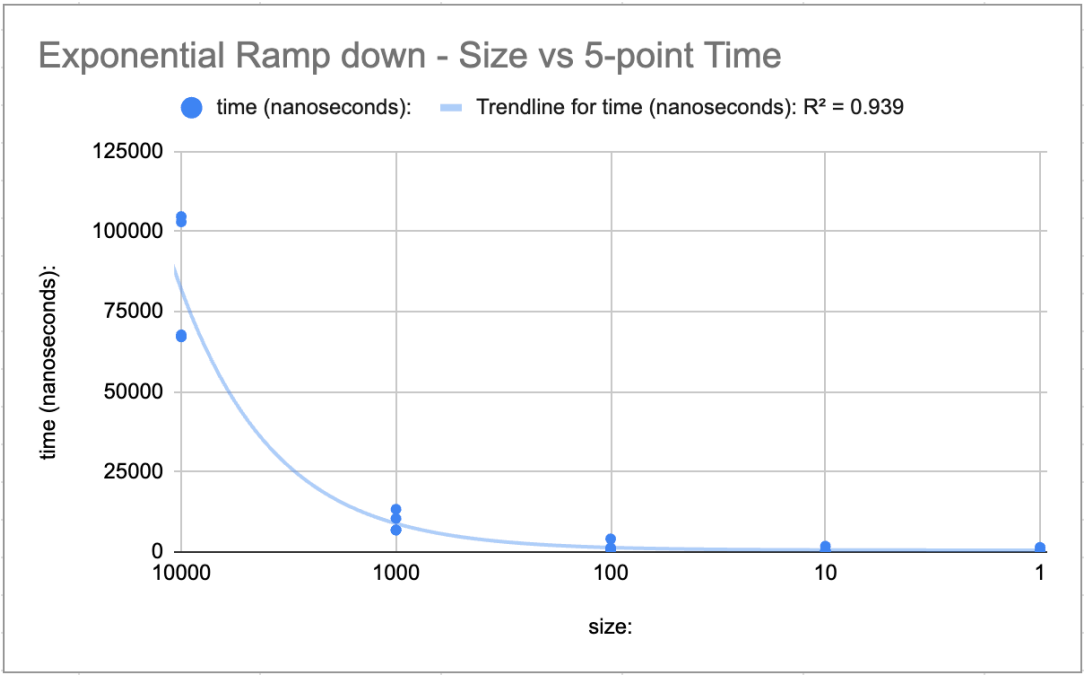
Technologies, Tools, and Resources used:
• Xcode
• Google Sheets

Tasks undertaken:

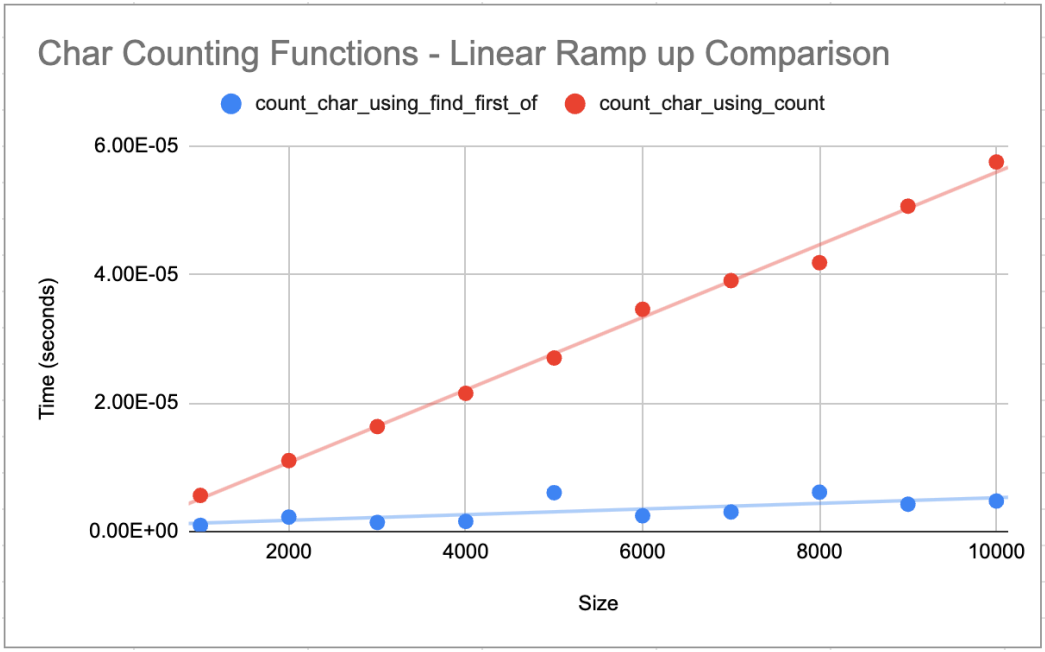
What we found out:
Measuring single tests:
We can use the steady_clock::now() function to measure the time both before and then after the function call. By setting the results to variables, we can simply subtract and find the difference in time, which will be the duration of the function.
Test Results:



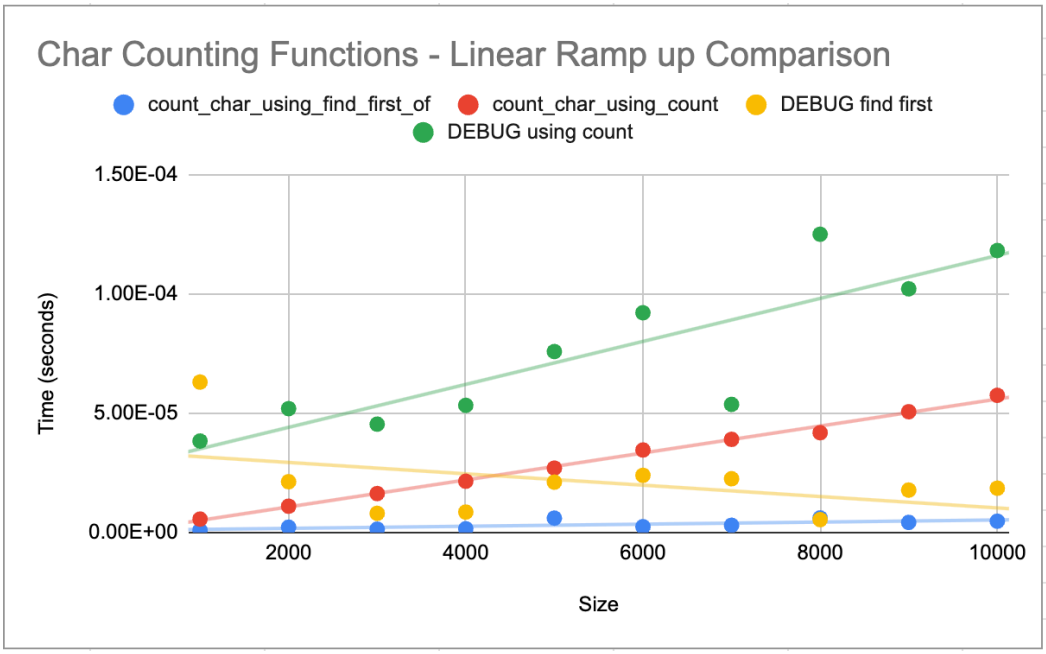




The initial test results for linear ramp up, exponential ramp up, and exponential ramp down are fairly easy to understand - the bigger the size, the longer the time. Even on the exponential tests, the relationship is actually linear; for sake of clarity a logarithm is applied on the horizontal axis. In terms of **repeatability**, it is very good. Each point was measured five times, in order to measure the variation. The data only seems to vary between 3 - 8% as shown by the R^2 value.



Here is the comparison between both char in string counting functions that count the letter s in a given string. Using randomly generated strings increasing in size, we can see a linear relationship for both functions with the length of the string. Furthermore, we can see the count_char_using_find_first_of() function is considerably less taxing on performance.



With the addition of debug performance measurements, we can see how performance is lessened. While there does seem to be a downward trend with the debug count_char_using_find_first_of() that would suggest a performance increase, the first data point on the line seems to be an outlier of sorts, and is skewing the data. By default, the debug compiler for Xcode disables optimisations, while the release/build compiler selects the fastest compiler.

To change the optimisation level on Xcode, the settings can be found under the Apple Clang - code generation heading in the build settings tab of your project. From there, the optimisation level for both release and debug builds can be set.