DSSA 5001
Problem Set 1
Due 28 October 2024

Efficient code is not necessary when analyzing small data sets (and the working definition of small is going up as computer hardware continues to improve). At some point, however, code written without an eye toward efficiency will become impractically slow, either during testing or in production. Many inefficiencies result from poor coding practices, so learning best practices from the outset reduces inefficiencies with little extra effort.

Terms used when measuring code performance include *profiling* and *benchmarking*. RStudio has profiling tools that can be used to identify bottlenecks in a script that runs too slowly. For timing code snippets (smaller sections of a script), the microbenchmark package is recommended.

(1) You have seen that there is usually more than one way to do a task in R. This is true even of something as apparently simple as taking the square root of a number. The built-in function `sqrt` is available. Alternatively, the exponentiation operator with 0.5 as the exponent can be used. For example, both `sqrt(2)` and `2^0.5` return the square root of 2.

The two methods can be compared using microbenchmark. Download `PS02_square_root.R` from the `Problem Sets/PS02` folder on Blackboard. Running the script loads microbenchmark and executes it to compare the two methods for taking the square root of 2.

The output of microbenchmark is displayed in the Console pane because it is not assigned to a variable. By default microbenchmark executes each expression 100 times to produce statistically meaningful results. The results are

- the expression(s) evaluated (`expr`);
- the minimum execution time(s) (`min`);
- the lower quartile(s) of the distribution of execution times (`lq`);
- the mean(s) of the distribution of execution times (`mean`);
- the median(s) of the distribution of execution times (`median`);
- the upper quartile(s) of the distribution of execution times (`uq`);
- the maximum execution time(s) (`max`);
  and
- the number of times the expression was evaluated (`neval`).

(a) There is quite a lot of scatter in the distribution of execution times. The presence of this scatter is noteworthy even if its causes are beyond the scope of this question. Add a comment to `PS02_square_root.R` in which you briefly discuss how you used the output of microbenchmark to get a sense of the scatter in execution times.

(b) Add a second comment to `PS02_square_root.R` in which you identify the more efficient of the two methods and briefly explain how you reached your conclusion. Extend your comment by estimating the number of times a script would need to calculate a square root for the difference in execution times to become noticeable. Briefly explain how you made your estimate.

(2) Determining how an algorithm performs as the size of the data set increases is a second benchmarking application. You will build on the example from our last class in this problem.

To start, download PS02_linear_search.R from the Problem Sets/PS02 folder on Blackboard. Running the script loads microbenchmark and defines the linear search algorithm. Next, use sample to create a test array named synth containing a shuffled list of the whole numbers from 1 to 8192.

(a) Enter the commands

```
target <- synth[1]
timing_data <- microbenchmark(linear_search(synth, target))
print(timing_data)
```

Record the median time, including its units, as a comment following the code in PS02_linear_search.R.

Examine timing_data. The first execution is often the slowest. Record which of the 100 executions of linear search took the most time, briefly describe how you identified it, and say whether the first execution was slowest as a comment in PS02_linear_search.R.

(b) Enter the commands

```
target <- tail(synth, n = 1)
timing_data <- microbenchmark(linear_search(synth, target))
print(timing_data)
```

Record the median time, including its units, as a comment following the code in PS02_linear_search.R.

Examine timing_data. Is the execution you identified as slowest in part (a) again the slowest? If not, where does it rank (e.g., third slowest)? Include this information, along with a brief description of how you determined the rank, in PS02_linear_search.R as a comment.

(c) Searching for an element that occurs near the start of the list, as in part (a), should be considerably faster than searching for an element near the end of the list, as in part (b). Do your results of parts (a) and (b) support the statement that it is faster to locate an element near the start of a list than near the end of a list? Referring to linear_search, briefly explain why this should be so in terms of the steps executed during a search as a comment in PS02_linear_search.R.