STAT 33B Workbook 7

Ming Fong (3035619833)

Oct 15, 2020

This workbook is due **Oct 15, 2020** by 11:59pm PT.

The workbook is organized into sections that correspond to the lecture videos for the week. Watch a video, then do the corresponding exercises *before* moving on to the next video.

Workbooks are graded for completeness, so as long as you make a clear effort to solve each problem, you'll get full credit. That said, make sure you understand the concepts here, because they're likely to reappear in homeworks, quizzes, and later lectures.

As you work, write your answers in this notebook. Answer questions with complete sentences, and put code in code chunks. You can make as many new code chunks as you like.

In the notebook, you can run the line of code where the cursor is by pressing `Ctrl` + `Enter` on Windows or `Cmd` + `Enter` on Mac OS X. You can run an entire code chunk by clicking on the green arrow in the upper right corner of the code chunk.

Please do not delete the exercises already in this notebook, because it may interfere with our grading tools.

You need to submit your work in two places:

- Submit this Rmd file with your edits on bCourses.
- Knit and submit the generated PDF file on Gradescope.

If you have any last-minute trouble knitting, **DON'T PANIC**. Submit your Rmd file on time and follow up in office hours or on Piazza to sort out the PDF.

For-loops

Watch the "For-loops" lecture video.

No exercises for this section.

Loop Indices

Watch the "Loop Indices" lecture video.

No exercises for this section.

While-loops

Watch the "While-loops" lecture video.

No exercises for this section.

Preallocation

Watch the “Preallocation” lecture video.

Exercise 1

Use the microbenchmark package to benchmark the “BAD” and “GOOD” example from the lecture video.

Benchmark with three different values of `n` (testing both the “BAD” and “GOOD” example for each value). About how much faster is the “GOOD” example?

YOUR ANSWER GOES HERE:

The “GOOD” example runs about 10-100 times faster than the “BAD” example.

```r
library(microbenchmark)

# BAD, NO PREALLOCATION:
bad = function(n) {
   x = c()

   for (i in 1:n) {
   x = c(x, i * 2)
   }
   x
}

# GOOD:
good = function(n) {
   x = numeric(n)
   for (i in seq_len(n)) {
   x[i] = i * 2
   }
}
microbenchmark(bad(5e2))
```

```
## Unit: microseconds
##      expr   min    lq    mean median    uq    max neval
##  bad(500) 345.1 372.7 532.069  389.8 423.5 5704.6   100
```

```r
microbenchmark(good(5e2))
```

```
## Unit: microseconds
##       expr  min   lq   mean median   uq    max neval
##  good(500) 32.7 35.1 83.694   35.4 35.9 4720.9   100
```

```r
microbenchmark(bad(1e3))
```

```
## Unit: milliseconds
##       expr    min      lq     mean  median      uq    max neval
##  bad(1000) 1.1613 1.23485 1.590201 1.29205 1.36555 5.8847   100
```

```r
microbenchmark(good(1e3))
```

```
## Unit: microseconds
##        expr  min   lq   mean median   uq   max neval
##  good(1000) 61.4 61.9 66.334   62.7 68.2 101.2   100
```

```
microbenchmark(bad(5e3))
```

```
## Unit: milliseconds
##       expr     min       lq     mean  median       uq     max neval
##  bad(5000) 29.0487 32.18795 33.87132 32.8926 33.96935 93.6655   100
```

```
microbenchmark(good(5e3))
```

```
## Unit: microseconds
##        expr   min    lq    mean median    uq   max neval
##  good(5000) 309.9 330.7 352.415 340.95 349.8 632.5   100
```

Loops Example

Watch the "Loops Example" lecture video.

Exercise 2

Write a function that returns the first `n + 1` positions of a 3-dimensional discrete random walk. Return the `x`, `y`, and `z` coordinates in a data frame with columns `x`, `y`, and `z`. Your function should have a parameter `n` that controls the number of steps.

Hint: For efficiency, use vectors for x, y, and z. Wait to combine them into a data frame until the very last line of your function.

YOUR ANSWER GOES HERE:

```
random_walk = function(n) {
   x = numeric(n + 1)
   y = numeric(n + 1)
   z = numeric(n + 1)

   xyz = sample(c(0, 1, 2), n, replace = TRUE)
   move = sample(c(-1, 1), n, replace = TRUE)

   for (i in seq_len(n)) {
      if (xyz[i] == 0) { # x
         x[i + 1] = x[i] + move[i]
         y[i + 1] = y[i]
         z[i + 1] = z[i]
   } else if (xyz[i] == 1){ # y
         x[i + 1] = x[i]
         y[i + 1] = y[i] + move[i]
         z[i + 1] = z[i]
   } else { # z
         x[i + 1] = x[i]
         y[i + 1] = y[i]
         z[i + 1] = z[i] + move[i]
   }
   }
   data.frame(x, y, z)
}
random_walk(10)
```

```
##     x  y  z
## 1   0  0  0
## 2   0  0  1
## 3   0 -1  1
## 4   0 -1  0
## 5   0 -2  0
## 6   0 -1  0
## 7  -1 -1  0
## 8  -1 -1 -1
## 9  -1 -2 -1
## 10 -2 -2 -1
## 11 -2 -1 -1
```

Recursion

Watch the "Recursion" lecture video.

Exercise 3

1. Use the microbenchmark package to benchmark `find_fib()` and `find_fib2()` for `n` equal to 1 through 30.
2. Collect the median timings for each into a data frame with a columns `time`, `n`, and `function`. The data frame should have 60 rows (30 for each function).
3. Use ggplot2 to make a line plot of `n` versus `time`, with a separate line for each `function`.
4. Comment on the the shapes of the lines. Does the computation time grow at the same rate (as `n` increases) for both functions?

YOUR ANSWER GOES HERE:

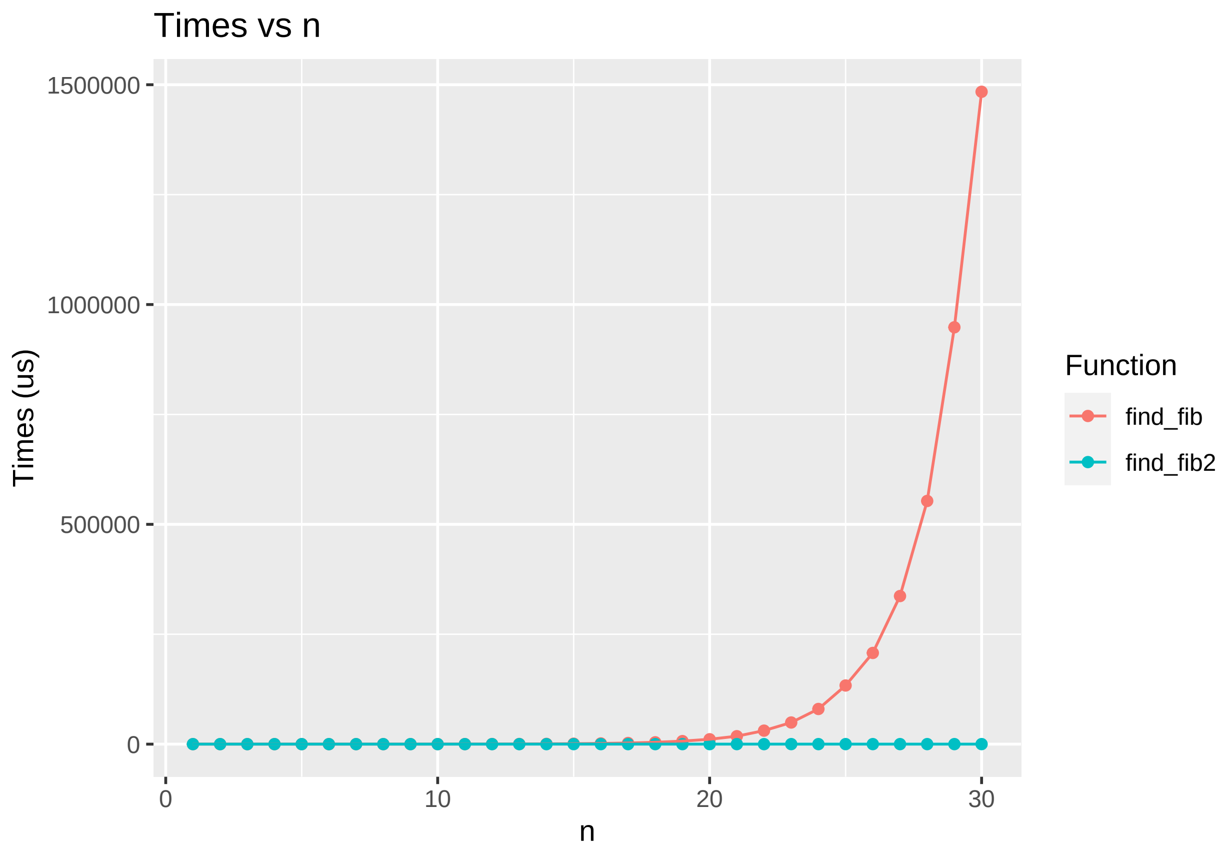
1. Combined with part 2 below
2.

```r
find_fib = function(n) {
  if (n == 1 | n == 2)
    return (1)

  find_fib(n - 2) + find_fib(n - 1)
}

find_fib2 = function(n, fib = c(1, 1)) {
  len = length(fib)
  if (n <= len)
    return (fib[n])

  fib = c(fib, fib[len - 1] + fib[len])
  find_fib2(n, fib)
}

times = numeric(60)
n = numeric(60)
fun = character(60)
for (i in 1:30){
   m1 = microbenchmark(find_fib(i), unit = "us", times = 30L)
   times[i] = summary(m1)["median"]
   n[i] = i
```

```r
    fun[i] = "find_fib"

    m2 = microbenchmark(find_fib2(i), unit = "us", times = 30L)
    times[i + 30] = summary(m2)["median"]
    n[i + 30] = i
    fun[i + 30] = "find_fib2"
}
times = unlist(times)
df = data.frame(times, n, fun)
```

3.

```r
library(ggplot2)
ggplot(df, aes(x = n, y = times, color = fun)) + geom_point() + geom_line() +
    labs(title = "Times vs n", color = "Function") +
    xlab("n") + ylab("Times (us)")
```



4. The recursive approach is **much** slower than the second appraoch. With recursion, the time to run grew exponentially with `n`, while the second method seemed linear.

Developing Iterative Code

Watch the "Developing Iterative Code" lecture video.

No exercises for this section. All done!