STAT 33B Lec Workbook WK 9

Won Shil Park (3033452021)

Mar 14, 2021

This workbook is due **Mar 17, 2021** 11:59pm

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 homework, quizzes, and later lectures.

Below are the code snippets from the preallocation video.

The first loop does not preallocate.

```
n = 20
x = c()
      for (i in 1:n) {
        x = c(x, i * 2)
      }
x
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
```

```
length(x)
```

```
## [1] 20
```

The next loop does preallocate.

```
n = 20
x = numeric(n)
for (i in seq_len(n)) {
  x[i] = i * 2
}

x
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
```

Step 1

If you haven't done so already, install the microbenchmark package. Examine the signature of the `microbenchmark()` function. Notice that the first parameter is `...` for supplying the expressions to benchmark.

Call `mcirobenchmark()` with the two versions of the code. Repeat the evaluation of the code 50 times and ask for the results to be in microseconds, i.e., set the `unit` parameter value to "ms". Note that you need only call `microbenchmark()` once, passing in the two versions of the code. Make sure to wrap each version in curly braces. Also, there's no need to supply `n` in the call. It can be set once outside the call.

```
library(microbenchmark)
n = 20
microbenchmark({x = c()
      for (i in 1:n) {
        x = c(x, i * 2)
      }},
{x = numeric(n)
for (i in seq_len(n)) {
  x[i] = i * 2
}},
times = 50L, unit = "ms")
```

```
## Unit: milliseconds
##                                                                                 expr
##             {     x = c()     for (i in 1:n) {         x = c(x, i * 2)     } }
##  {     x = numeric(n)     for (i in seq_len(n)) {         x[i] = i * 2     } }
##       min       lq     mean   median       uq      max neval
##  1.543925 1.697814 2.042377 1.856960 2.216119 4.397630    50
##  1.552324 1.681021 1.898261 1.735228 1.995713 3.608888    50
```

Step 2

Wrap your call to `microbenchmark()` into a function called `benchit()`. The parameters to this function should be

- n - the length of the vector x; make this a required argument
- times - the number of times the expressions are run; set the default to 50L

```
benchit = function(n, rep = 50L) {
  microbenchmark({x = c()
      for (i in 1:n) {
        x = c(x, i * 2)
      }},
{x = numeric(n)
for (i in 1:n) {
  x[i] = i * 2
}},
times = rep, unit = "ms")
}
```

Call `benchit()` with `n = 20` and assign the return value to `b`. Use `class()` to confirm that `b` is a data frame. What is its dimension? Variable names?

```
b = benchit(n = 20)

class(b)
```

```
## [1] "microbenchmark" "data.frame"
```

```
names(b)
```

```
## [1] "expr" "time"
```

```
sapply(b, class)
```

```
##      expr      time
##  "factor" "numeric"
```

```
dim(b)
```

```
## [1] 100   2
```

```
head(b$time)
```

```
## [1] 21617  8351  8729 19107  8639 14051
```

```
tail(b$time)
```

```
## [1] 11515  8118 12758 11494 11699 12574
```

```
head(b)
```

```
## Unit: milliseconds
##                                                                      expr
##      {     x = c()     for (i in 1:n) {         x = c(x, i * 2)     } }
##  {     x = numeric(n)     for (i in 1:n) {         x[i] = i * 2     } }
##       min       lq     mean   median       uq      max neval
##  0.008351 0.008495 0.011834 0.008684 0.015173 0.021617     4
##  0.014051 0.014051 0.016579 0.016579 0.019107 0.019107     2
```

Step 3

Use `tapply()` with `b` to compute the median time for each expression.

```
tapply(b$time, b$expr, median)
```

```
##     {     x = c()     for (i in 1:n) {         x = c(x, i * 2)     } }
##                                                                  7977.0
## {     x = numeric(n)     for (i in 1:n) {         x[i] = i * 2     } }
##                                                                 12093.5
```

Step 4

Now let's benchmark for vectors of length 10, 100, 1000, and 10000. If 10000 is too slow for your computer then you can reduce the value.

- Create a vector called `n` with these 4 values.
- Preallocate a vector twice the length of n to store the median times for the two approaches. Call this vector `meds`.
- Write a for loop to call `benchit()` for each `n` and assign the median times for the two approaches into `meds`.

Hint: We recommend using `seq_along` in the for loop.

```
n = c(10, 100, 1000, 10000)
meds = numeric(2 * length(n))

for (i in seq_along(n)) {
  b = benchit(n = n[i])
  meds[c((2 * i - 1), (2*i))] = c(tapply(b$time, b$expr, median)[1],
                      tapply(b$time, b$expr, median)[2])

}
meds
```

```
## [1]      3821.0      6390.0     64768.0     56447.0   2042313.5    610787.5
## [7] 304112036.0   6160836.5
```

How much faster is preallocation when `n` is 10000? How do the two methods compare for the different values of n? To make this comparison divide the median times for the first strategy by the preallocation strategy (e.g., divide the first element of `meds` by the second, the third by the fourth, and so on). Note this can be done as a vectorized calculation.

```
meds[seq(1, 8, by = 2)] / meds[seq(2, 8, by = 2)]
```

```
## [1]  0.5979656  1.1474126  3.3437382 49.3621339
```