# Problem Set 4

Due Wednesday Oct. 12, 10 am

## Comments

- This covers Unit 5.
- It's due at 10 am (Pacific) on October 12, both submitted as a PDF to Gradescope as well as committed to your GitHub repository.
- Please see PS1 and the grading rubric for formatting and attribution requirements.
- I just noticed that the `pryr` package has been superseded by functionality in other packages, particularly by `lobstr` in terms of our uses of `pryr`. So I suggest you use `lobstr::obj_addr`, `lobstr::obj_size`, and `lobstr::mem_used` in place of `pryr::address`, `pryr::object_size`, and `pryr::mem_used` respectively. That said, you'll likely need `.Internal(inspect())` for your solutions for the more granular information it provides.

## Problems

1. This question explores memory use and copying with character vectors.

   a. Consider the following character vector with three strings. Modify one of the strings. Can R make the change in place? (Be careful, there are two aspects to this, so it's a bit more complicated than simply replacing an element in a numeric vector.) Also note that you may need to copy-paste output from R or RStudio as compiling the PDF may change the answer.

   ```
   vec <- c("hello there", "better luck next time", "that's not clear")
   ```

   b. Now consider this vector: `vec <- c(rep('hello friend', 1e6))`. Given each character should take 1 byte, this would seemingly use 12 million bytes. How much memory is being used? Explain what is happening and account for all major uses of memory.

   c. Compare the size of the string 'hello' with that of a single string of length 1 million characters (i.e., one where `nchar()` returns 1 million). Does each character take up 1 byte? What does this comparison suggest about short strings?

   Warning: Recall that rendering your Rmd can result in the memory allocation/address information being incorrect, so you are likely to need to paste in some results manually.

2. If I want to compute the trace of a matrix, $A = XY$, where both $X$ and $Y$ are $n \times n$ and where the trace is $\sum_{i=1}^{n} A_{ii}$, a naive implementation is `sum(diag(X%*%Y))`.

1. What is the computational complexity of that naive implementation: $O(n)$, $O(n^2)$ or $O(n^3)$? You can just count up the number of multiplications and ignore the additions.
2. Why is that naive implementation inefficient?
3. How could you (much) more efficiently compute the trace in R using vectorized operations on the matrices. Please provide R code and do not use `apply()`? What is the computational complexity of your solution?
4. Create a plot to demonstrate the quadratic vs. cubic scaling using a few values of $n$.

3. Suppose we have a matrix in which each row is a vector of probabilities that add to one, and we want to generate a categorical sample based on each row. E.g., the first row might be (0.9, 0.05, 0.05) and the second row might be (0.1, 0.85, .0.5). When we generate the first sample, it is very likely to be a 1 and the second sample is very likely to be a 2. We could do this using a for loop over the rows of the matrix, and `sample()`, but that is a lot slower than some other ways we might do it because it is a loop executing in R over many elements.

```
n <- 100000
p <- 5  ## number of categories

## way to generate a random matrix of row-normalized probabilities:
tmp <- exp(matrix(rnorm(n*p), nrow = n, ncol = p))
probs <- tmp / rowSums(tmp)

smp <- rep(0, n)

## loop by row and use sample()
set.seed(1)
system.time(
  for(i in seq_len(n))
        smp[i] <- sample(p, 1, prob = probs[i, ])
)
```

   a. Consider transposing the matrix and looping over columns. Why might I hypothesize that this could be faster? Is it faster?
   b. How can we do it much faster? (Hint: This might involve looping or not, but not in the ways described above. Think about how one can use random uniform numbers to generate from a categorical distribution.)

4. Suppose I run the code `plot(xvec, yvec)`. It's the case that `range()` is called when making the axis limits. Suppose I create a function called `range()` in my R session and then make my plot, as follows.

```
n <- 10
xvec <- rnorm(n)
yvec <- rnorm(n)
```

```
range <- function(...) print("better luck next time")
range(rnorm(3))
## [1] "better luck next time"

plot(xvec, yvec)
```

Explain **in detail** why can I still make a plot. As part of your answer, say what functions are on the call stack at the point that `range()` is called and how the "right" `range` is found.

5. (Extra credit) Consider `plot(xvec, rnorm(5))`. Explain how it is that the x-axis and y-axis labels can be assigned to be "xvec" and "rnorm(5)". The material in the optional Section 10 of Unit 5 will be useful.