Homework 1

Daniel Kiser

February 9, 2018

Problem 1

```r
# a
x <- c(9:16)
x
```

```
## [1]  9 10 11 12 13 14 15 16
```

```r
# b
last <- length(x)
x[(last - 2):last]
```

```
## [1] 14 15 16
```

```r
# c
for (i in 1:length(x)) {
    if (x[i]%%2 == 0)
        cat(x[i], " ")
}
```

```
## 10  12  14  16
```

```r
# d
for (i in length(x):1) {
    if (x[i]%%2 == 0) {
        x <- x[-i]
    }
}
x
```

```
## [1]  9 11 13 15
```

Problem 2

```r
forfunc <- function(n) {
    sum <- 0
    for (i in 1:n) {
        value <- (1/2)^i
        sum <- sum + value
    }
    return(sum)
}

whilefunc <- function(n) {
    sum <- 0
    i <- 1
    while (i <= n) {
        value <- (1/2)^i
        sum <- sum + value
        i = i + 1
    }
    return(sum)
```

```
}

analyticfunc <- function(n) {
    sum <- (1 - (1/2)^n)
    return(sum)
}

v <- seq(1:30)
compare <- function(v) {
    forresults <- unlist(lapply(v, forfunc))
    whileresults <- unlist(lapply(v, whilefunc))
    analyticresults <- unlist(lapply(v, analyticfunc))

    cat("Results of for-loop:\n", forresults, "\n")
    cat("Results of while loop:\n", whileresults, "\n")
    cat("Results of analytic formula:\n", analyticresults, "\n")
}
compare(v)
```

```
## Results of for-loop:
##  0.5 0.75 0.875 0.9375 0.96875 0.984375 0.9921875 0.9960938 0.9980469 0.9990234 0.9995117 0.9997559 (
## Results of while loop:
##  0.5 0.75 0.875 0.9375 0.96875 0.984375 0.9921875 0.9960938 0.9980469 0.9990234 0.9995117 0.9997559 (
## Results of analytic formula:
##  0.5 0.75 0.875 0.9375 0.96875 0.984375 0.9921875 0.9960938 0.9980469 0.9990234 0.9995117 0.9997559 (
```

As n gets very large, all three approaches get closer and closer to 1 until $n = 25$, when R begins to round off to 1.

Problem 3

- If $x^3 \epsilon \Theta(x^3)$ then there exists c_1, c_2, x_0 such that $c_1 g(x) \leq x^3 \leq c_2 g(x)$ for all $x \geq x_0$. $g(x) = x^3$ so the equation becomes

  $$c_1 x^3 \leq x^3 \leq c_2 x^3$$

  which can be satisfied if $c_1 \leq 1, c_2 \geq 1$ and $x_0 = 0$.
  If $x^3 \epsilon O(x^3)$ then there exists c, x_0 such that $0 \leq x^3 \leq c g(x)$ for all $x \geq x_0$. Inserting x^3 for $g(x)$ the equation becomes

  $$0 \leq x^3 \leq c x^3$$

  which can be satisfied if $c \geq 1$ and $x_0 = 0$.
  If $x^3 \epsilon \Theta(x^4)$ then there exists c_1, c_2, x_0 such that $c_1 g(x) \leq x^4 \leq c_2 g(x)$ for all $x \geq x_0$. $g(x) = x^4$ so the equation becomes

  $$c_1 x^4 \leq x^3 \leq c_2 x^4$$

  This equation cannot be satisfied for all $x \geq x_0$, since as x goes to infinity, $c_1 x^4$ will be larger than x^3 no matter how small c_1 is. We can also see that

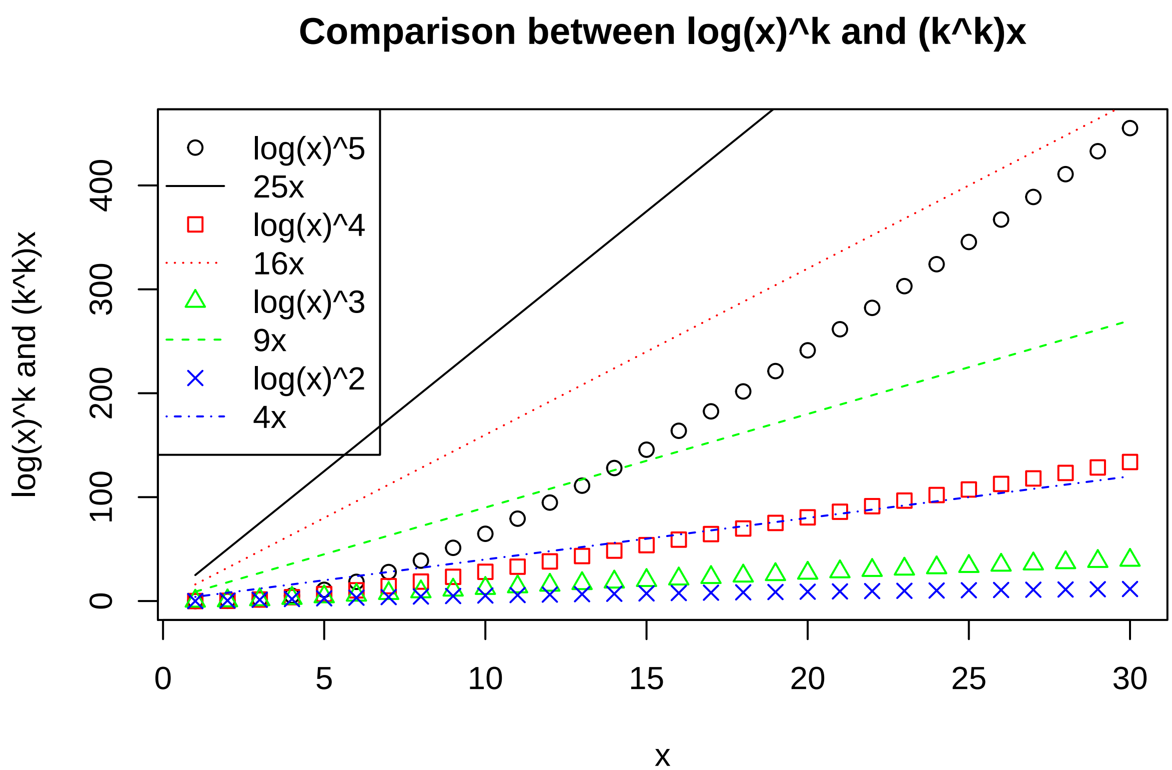
  $$\frac{f(x)}{g(x)} = \frac{x^3}{x^4}$$

  goes to zero as x increases to infinity, which also indicates that x^3 does not belong to $\Theta(x^4)$, since if $x^3 \epsilon \Theta(x^4)$, the limit of $\frac{x^3}{x^4}$ as x approaches infinity would be a constant greater than 0.

- For any real constants a and $b > 0$, $(n+a)^b = n^b + bn^{b-1}a^{b-1} + ... + a^b$. As n goes to infinity, the growth rate is determined by n^b, since it is the largest power of n in the polynomial. Thus $g(n) = n^b$. Since the limit of $\frac{f(n)}{g(n)} = \frac{n^b + bn^{b-1}a^{b-1} + ... + a^b}{n^b}$ as n approaches infinity is 1, a constant greater than 0, $(n+a)^b \epsilon \Theta(n^b)$.

- If $(log(n))^k \epsilon\ O(n)$, then there exists c, n_0 such that $0 \leq (log(n))^k \leq cn$ for all $n \geq n_0$. One solution is $n_0 = 0, c = k^k$, as demonstrated by the plot below. Thus, $(log(n))^k \epsilon O(n)$.

```
# Plot to demonstrate that log(x)^k always has (k^k)x as an
# upper bound
x <- seq(1, 30)
plot(x, log(x)^5, xlab = "x", ylab = "log(x)^k and (k^k)x", main = "Comparison between log(x)^k and (k^k
lines(x, 25 * x)
lines(x, log(x)^4, type = "p", col = "red", pch = 0)
lines(x, 16 * x, col = "red", lty = 3)
lines(x, log(x)^3, type = "p", col = "green", pch = 2)
lines(x, 9 * x, col = "green", lty = 2)
lines(x, log(x)^2, type = "p", col = "blue", pch = 4)
lines(x, 4 * x, col = "blue", lty = 4)
legend("topleft", c("log(x)^5", "25x", "log(x)^4", "16x", "log(x)^3",
    "9x", "log(x)^2", "4x"), col = c("black", "black", "red",
    "red", "green", "green", "blue", "blue"), lty = c(0, 1, 0,
    3, 0, 2, 0, 4), pch = c(1, NA, 0, NA, 2, NA, 4, NA))
```



- Since $\frac{n}{n+1} = 1 + \frac{1}{n}$, we can write

$$T(\frac{n}{n+1}) = 1 + T(\frac{1}{n})$$

We know that $T(1/n) = O(1/n)$ since the equation $0 \leq \frac{1}{n} \leq c\frac{1}{n}, n \geq n_0$ is satisfied by $c \geq 1$ and $n_0 = 0$. Thus, $\frac{n}{n+1} = 1 + O(\frac{1}{n})$.

- We know that $\sum_{i=0}^{[log_2(n)]} 2^i = 1 + 2 + 4 + 16 + ... + n$. Since

$$\frac{f(n)}{g(n)} = \frac{1 + 2 + ... + n}{n} > 0$$

as n approaches infinity, we know that $\sum_{i=0}^{[log_2(n)]} 2^i \epsilon \Theta(n)$.

Problem 4

```
bubblesort <- function(x) {
    n <- length(x)
    swapped <- TRUE
    while (swapped == TRUE) {
        swapped <- FALSE
        for (i in 1:(n - 1)) {
            if (x[i] > x[i + 1]) {
                temp <- x[i]
                x[i] <- x[i + 1]
                x[i + 1] <- temp
                swapped <- TRUE
            }
        }
    }
    return(x)
}

mergearrays <- function(x, y) {
    m = length(x)
    n = length(y)
    if (m == 0) {
        return(z = y)
    }
    if (n == 0) {
        return(z = x)
    }
    if (x[1] <= y[1]) {
        return(z = c(x[1], mergearrays(x[-1], y)))
    } else {
        return(z = c(y[1], mergearrays(x, y[-1])))
    }
}

mergesort <- function(x) {
    n = length(x)
    mid = floor(n/2)
    if (n > 1) {
        return(mergearrays(mergesort(x[1:mid]), mergesort(x[(mid +
```

```
            1):n])))
    } else {
        return(x)
    }
}

set.seed(12345)
reps <- 100
time1 <- time2 <- rep(0, reps)
for (i in 1:reps) {
    x <- rnorm(1000, 0, 1)
    ptm <- proc.time()
    mergesort(x)
    t1 <- proc.time() - ptm
    time1[i] = t1[["elapsed"]]
    ptm <- proc.time()
    bubblesort(x)
    t2 <- proc.time() - ptm
    time2[i] = t2[["elapsed"]]
}

cat("merge sort average time:", mean(time1))
```

```
## merge sort average time: 0.07563
```

```
cat("merge sort variance:", var(time1))
```

```
## merge sort variance: 0.001187003
```

```
cat("bubble sort average time:", mean(time2))
```

```
## bubble sort average time: 0.19298
```

```
cat("bubble sort variance:", var(time2))
```

```
## bubble sort variance: 0.0002543228
```

The mean time for merge sort was near 0.07592 seconds while the mean time for bubble sort was near 0.20083 seconds. However, the variance for merge sort (near 0.001056377) was about 4 to 5 times as much as the variance for bubble sort (near 0.0003222637). (The reported values differ from the output of my R code, since the values changed slightly with each run when I modified this document.) These results agree with my expectations, since merge sort is $O(nlog_2n)$ while bubble sort is $O(n^2)$. Thus, as n increases we would expect the worst case for bubble sort to increase faster than the worst case for merge sort.

Bonus Problem

```
find_kth_smallest <- function(x, k) {
    smallest <- x[1]
    index <- 1
    if (length(x) > 1) {
        for (i in 2:length(x)) {
            if (x[i] < smallest) {
                smallest <- x[i]
                index <- i
```

```
            }
        }
        x <- x[-index]
    }
    if (k == 1) {
        print(smallest)
    } else find_kth_smallest(x, k - 1)
}

y <- c(4, 2, 8, 1, -4)
find_kth_smallest(y, 3)
```

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

In order to find the k-th smallest element of the array, the algorithm loops through an n-length array k times, removing the largest element each time. The k-th element removed from the array is returned as the k-th smallest element.

Since the algorithm loops k times through an array that starts at length n and is reduced by 1 each time, the number of computational steps in the algorithm can be written as:

$$f(n) = n + (n - 1) + (n - 2) + ... + (n - k)$$

This can be rewritten as:

$$f(n) = kn - \sum_{i=1}^{k} i$$

We can find c and n_0 such that $0 \leq kn - \sum_{i=1}^{k} i \leq cn$ where $n > n_0$. $n_0 = 0$ and c can be any number greater than k. Thus, $f(n) \epsilon O(n)$.