Valentine_C_3B

Clint Valentine

January 24, 2017

Load the data file

```
require(lubridate)
```

```
if (exists('bird.strikes') == F) {
  unzip('Bird Strikes.zip')
  # Escape sep = ',' in quotations
  # Impute NA as empty strings
  # Do not consider strings factors
  bird.strikes <- read.table(
    file = 'Bird Strikes.csv',
    fill = T,
    header = T,
    na.strings = '',
    quote = '\"',
    sep = ',',
    stringsAsFactors = F
  )

  # Convert `Reported..Date` column to POSIX objects
  bird.strikes <- within(bird.strikes, Reported..Date <- mdy_hm(Reported..Date))
}
```

How many bird strikes did not have a `Reported: Date` assigned, i.e., for where there is no value for `Reported: Date`?

```
with(bird.strikes, sum(is.na(Reported..Date)))
```

```
## [1] 72564
```

Which year had the most bird strikes? Write a function to calculate.

```
yearMostStrikes <- function(bird.strikes) {
  # Return the year with the most bird strikes.
  #
  # Args:
  #   bird.strikes: Bird Strikes data.
  #
  # Returns:
  #   Numeric longform year with the most bird strikes.

  # Tally the number of bird strikes per year
  year.tally <- table(with(bird.strikes, year(Reported..Date)))

  # Sort the binned bird strikes by year, by frequency, then take the
  # name of the last element (greatest frequency) and return as.numeric
  return(as.numeric(names(tail(sort(year.tally), 1))))
}

yearMostStrikes(bird.strikes)
```

```
## [1] 2002
```

How many bird strikes were there for each year? Place the result into a data frame. —

```
yearStrikes <- data.frame(sort(table(with(bird.strikes, year(Reported..Date))),
                               decreasing = T))
colnames(yearStrikes) <- c("Year", "Frequency")

yearStrikes
```

```
##    Year Frequency
## 1  2002      4930
## 2  2003      3782
## 3  2004      3086
## 4  2005      2999
## 5  2001      2919
## 6  2000      2155
## 7  2006      1748
## 8  2007      1259
## 9  2008      1169
## 10 2009      1167
## 11 2011       791
## 12 2010       774
## 13 2012        61
```

Write a function that calculates the number of birds strikes per airline. Write another function that returns the airline that has the most bird strikes.

```
airlineStrikesAll <- function(x) {
  # Return a tally of bird strikes by airline
  #
  # Args:
  #   x: Bird Strikes data.frame.
  #
  # Returns:
  #   data.frame of bird strikes by airline.

  # Tally the number of bird strikes per airline, sort, and return
  x <- data.frame(sort(table(with(x,
                                  Aircraft..Airline.Operator)),
                  decreasing = T))
  colnames(x) <- c("Airline", "Frequency")
  return(x)
}

airlineMostStrikes <- function(x) {
  # Return the airline with the most bird strikes.
  #
  # Args:
  #   x: data.frame of bird strikes by airline
  #
  # Returns:
  #   Airline with the most bird strikes.

  # Return the name of the second to last element (greatest frequency)
  # because the last element is usually UNKNOWN
  return(as.character(with(AirlineStrikes, tail(head(Airline, 2), 1))))

}

AirlineStrikes <- airlineStrikesAll(bird.strikes)
airlineMostStrikes(AirlineStrikes)
```

```
## [1] "MILITARY"
```

Comment on the time and space complexity of your functions.

List of steps and their respective complexities in time and space:

1. All airlines need to be tallied. This can occur in $O(n)$ complexity, where n is equal to the number of observations, as we loop through each element and increment a counter for each unique object.
2. In order to determine the greatest or second greatest value in a container we must first sort the container. Sorting, in its worst case, involves pairwise comparison of all elements which occurs in $O(n^2)$ (n is equal to elements in list to be sorted). Sorting, in its best case, can sort in $O(n)$ complexity depending on the algorithm and initial container ordering.
3. Column names are change in linear time at $O(n)$ where n is equal to the number of columns to be labelled.
4. After sorting, slicing of the container is needed to extract the second to last element. Slicing is achieved in $O(n)$ complexity where n is often the length of the list to be sliced and not the slice size.

Overall our implementation runs in linear time ($O(n)$) in the best case and quadratic time ($O(n^2)$) if the `sort` routine in R is poorly optimized.

What would happen if the data set were 2x, 10x, 100x, 1000x bigger than it is now? How would that affect memory use and run time of your code?

For $O(n)$ complexity the size of the input data and both the memory and time are dependently related with an approximately linear fit.

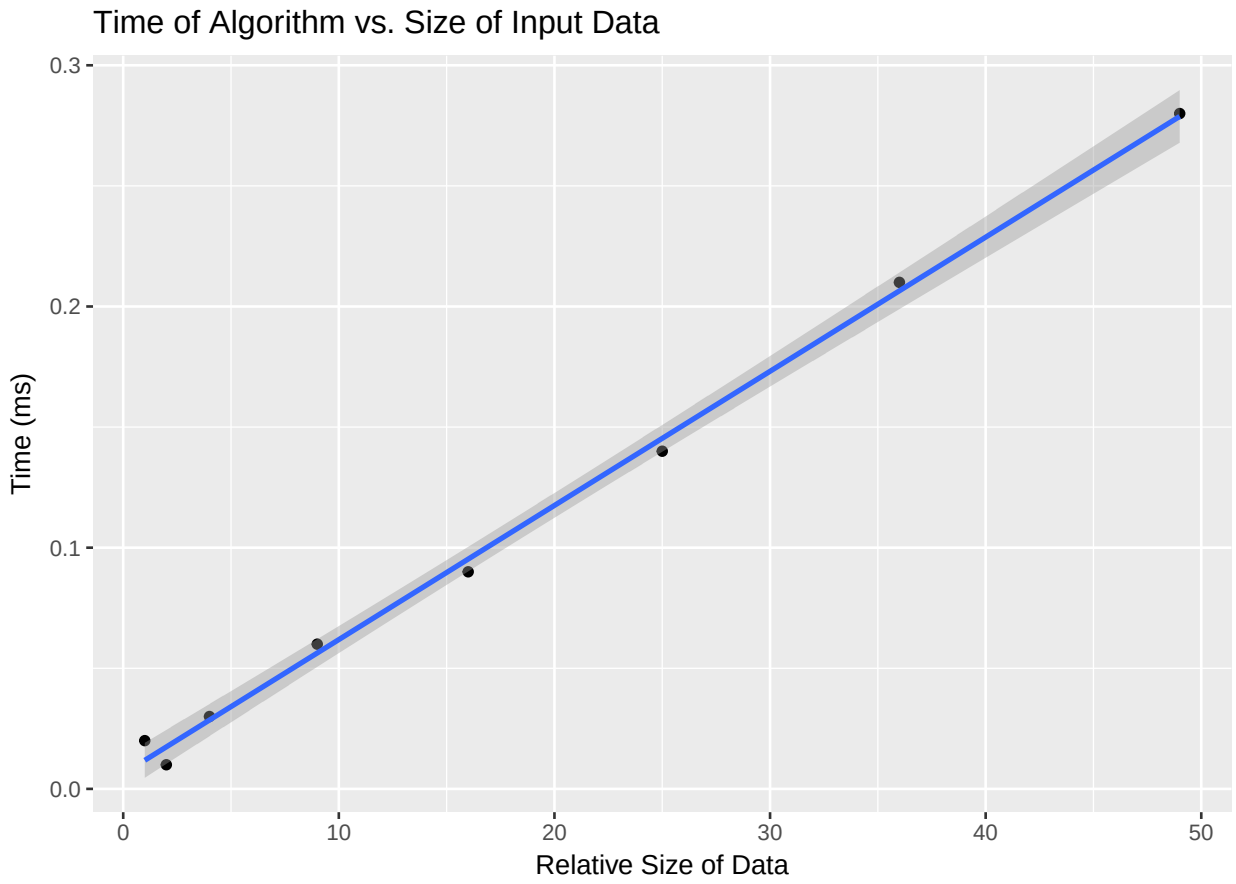
Test the execution time for increased data size.

Our hypothesis is that the complexity of our function `airlineStrikesAll` is $O(n)$ at best where n is equal to the number of observations. More data points were selected than in the assignment to fit a proper linear model. The perfect linear fit to the timing results show that our algorithm is indeed $O(n)$ over 1.5x orders of magnitude.

```
magnitudes <- c(1, 2, sapply(2:7, function(x) x^2 ))

time <- rep(NA, length(magnitudes))

for (i in 1:length(magnitudes)) {
    test.data <- do.call("rbind", replicate(magnitudes[i],
                                            bird.strikes,
                                            simplify=F))
    time[i] <- system.time(airlineStrikesAll(test.data))[['elapsed']]
}
```

```
require(ggplot2)
```

```
results <- data.frame(list(magnitudes=magnitudes, time=time))

ggplot(results, aes(magnitudes, time)) +
  geom_point() +
  geom_smooth(method = 'lm') +
  labs(title = 'Time of Algorithm vs. Size of Input Data',
       x = 'Relative Size of Data',
       y = 'Time (ms)')
```




```
print(summary(lm(magnitudes ~ time, data=results)))
```

```
##
## Call:
## lm(formula = magnitudes ~ time, data = results)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.5134 -0.5996 -0.2127  0.9481  1.2791
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -1.072      0.567   -1.89    0.108
## time         179.254      4.071   44.03 9.19e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.054 on 6 degrees of freedom
## Multiple R-squared:  0.9969, Adjusted R-squared:  0.9964
## F-statistic:  1939 on 1 and 6 DF,  p-value: 9.188e-09
```