HW1-Darshan Patel-3:30-5:30PM

Darshan Patel

9/23/2018

Question 1:

The vectors `state.name`, `state.area`, and `state.region` are pre-loaded in *R* and contain US state names, area (in square miles) and region respectively.

Solution:

(a) Identify the data type for `state.name`, `state.area`, and `state.region`.

The data type for `state.name` is

```
class(state.name)
```

```
## [1] "character"
```

The data type for `state.area` is

```
class(state.area)
```

```
## [1] "numeric"
```

The data type for `state.region` is

```
class(state.region)
```

```
## [1] "factor"
```

(b) What is the longest state name (including spaces)? How long is it?

```
# Set the longest state name to be the first value in the vector
longest_state <- state.name[1]

# For each value in the state.name vector, if the length of the state name is greater than
# the longest state name determined, replace the longest state variable
for(i in 1:length(state.name)){
  temp_length <- nchar(state.name[i])
  if(temp_length > nchar(longest_state)) longest_state <- state.name[i]
}

# Print result
paste("The longest state name is", longest_state,"and it is", nchar(longest_state), "characters long.",
```

```
## [1] "The longest state name is North Carolina and it is 14 characters long."
```

(c) Compute the average area of the states which contain the word "New" at the start of the state name. Use the function `substr()`.

```
# Create an empty vector to store areas that fulfull the requirement
new_states <- c()

# For each state name, if it contains the word New at the beginning,
# store its name in the above vector by concatement
for(i in 1:length(state.name)){
  if(substr(state.name[i], start = 1, stop = 3) == "New")
```

```
    new_states <- c(new_states, i)
}

# Default area = 0
area <- 0

# For each state name in the vector, add its area to the aggregating variable
for(i in 1:length(new_states)) area <- area + state.area[new_states[i]]

# Compute average
avg_area <- area / length(new_states)

# Print result
paste("The average area of the states which contain the word New at the start of the state name is", avg
```

```
## [1] "The average area of the states which contain the word New at the start of the state name is 4709
```

(d) Use the function `table()` to determine how many states are in each region. Use the function `kable()` to include the table in your solutions.

```
# Import knitr
library(knitr)

# Use kable() to print table of number of states per region in an aesthetic manner
kable(table(state.region), caption="Number of States per Region")
```

Table 1: Number of States per Region

state.region	Freq
Northeast	9
South	16
North Central	12
West	13

Question 2:

Perfect numbers are those where the sum of the proper divisors (i.e., divisors other than the number itself) add up to the number. For example, 6 is a perfect number because its divisors, 1, 2, and 3, when summed, equal 6.

Solution:

(a) The following code was written to find the first 2 perfect numbers: 6 and 28; however, there are some errors in the code and the programmer forgot to add comments for readability. Debug and add comments to the following:

```
# Create counter for number of perfect numbers to find
nums.perfect <- 2
count <- 0
iter <- 2

# Keep running until the designated number of perfect numbers is found
while(count < nums.perfect){
  # Set 1 to be the starting divisor
  divisor <- 1

  # Fom 2 onwards, look for possible divisors of a number and add it to a vector if so
  # Stop when it approaches the number minus 1
  for(i in 2:(iter-1)){
    if(iter%%i==0) divisor <- c(divisor, i)
  } # end for loop

  # If the sum of the divisors is equal to the number being tested, it is a perfect number
  # Print the value and then increment number of perfect numbers found
  if(sum(divisor)==iter){
    print(paste(iter, " is a perfect number", sep=""))
    count <- count + 1
  } # end if

  # Increment to the next number to be tested
  iter <- iter + 1
} #end while loop
```

```
## [1] "6 is a perfect number"
## [1] "28 is a perfect number"
```

(b) Use the function `date()` at the start and at the end of your amended code. Then compute how long the program approximately takes to run. Find the run time when you set `num.perfect` to 1, 2, 3 and 4. Create a table of your results. What are the first four perfect numbers?

```
# A function that looks for a variable number of perfect numbers and store them in a vector.
perfect_numbers <- function(x){

  # Create counter for number of perfect numbers to find
  nums.perfect <- x
  count <- 0
  iter <- 2

  # Create vector to store perfect numbers
  perfect_nums <- c()

  # Keep running until the designated number of perfect numbers is found
  while(count < nums.perfect){

    # Set 1 to be the starting divisor
    divisor <- 1

    # From 2 onwards, look for possible divisors of a number and add it to a vector if so
    # Stop when it approaches the number minus 1
    for(i in 2:(iter-1)){
      if(iter%%i==0) divisor <- c(divisor, i)
    } # end for loop

    # If the sum of the divisors is equal to the number being tested, it is a perfect number
    # Append to list of perfect numbers
    if(sum(divisor)==iter){
      perfect_nums <- c(perfect_nums, iter)
      count <- count + 1
    } # end if

    # Increment to the next number to be tested
    iter <- iter + 1
  } #end while loop
  perfect_nums
}

# Store time durations in vector
duration <- rep(0,4)

# Get time duration for 4 runs of perfect_numbers from nums.perfect = 1 to 4
for(i in 1:4){
  start_time <- Sys.time()
  perfect_numbers(i)
  end_time <- Sys.time()
  duration[i] <- round(end_time - start_time, digits = 3)
}
```

Answer: Table of Run Times:

```
# Bind vector of time duration with the respective column of
# perfect numbers to find into a nice table
run_times <- cbind(seq(1,4,1), duration)
colnames(run_times) <- c("num.perfect", "duration in seconds")
run_times
```

```
##      num.perfect duration in seconds
## [1,]           1               0.014
## [2,]           2               0.000
## [3,]           3               0.028
## [4,]           4               3.481
```

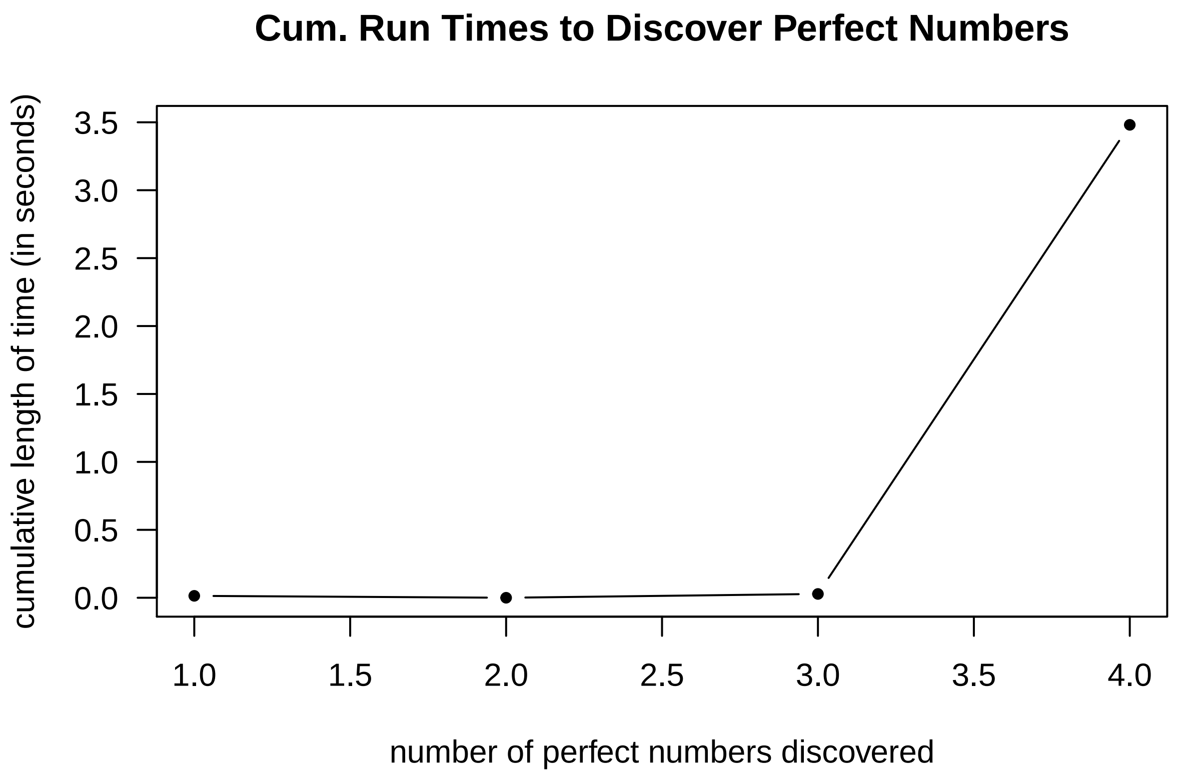
The first four perfect numbers are:

```
perfect_numbers(4)
```

```
## [1]    6   28  496 8128
```

(c) Let `x <- 1:4` and define `y` to be the vector of run times. Plot `y` vs `x` using the code below. Is the relationship between the discovery of perfect numbers and run times on your computer linear? Justify your answer.

```
x <- 1:4
y <- duration
plot(x, y, pch=20, type="b",
     xlab="number of perfect numbers discovered",
     ylab="cumulative length of time (in seconds)",
     main="Cum. Run Times to Discover Perfect Numbers",
     las=TRUE)
```



Answer: The relationship between the discovery of perfect numbers and run times is not linear. As the number of perfect numbers to be found increased, the value of the perfect number itself was found further and further away from the previous value.

Question 3:

The geometric mean of a numeric vector x is computed as follows:

$$\tilde{x} = \left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}}$$

Solution:

(a) Using a `for` loop, write code to compute the geometric mean of the numeric vector `x <- c(4, 67, 3)`. Make sure your code (i) removes any `NA` values and (ii) prints an error message if there are any non-positive values in `x`.

```r
# Function to conpute the geometric mean of a numeric vector
geometric_mean <- function(v){

  # Removes all NA values from given vector
  v <- v[!is.na(v)]

  # Initialize the product to be 1
  g_prod <- 1

  # For each value in the vector, check if less than 0
  # If yes, then return error message
  # Otherwise, multiple its value by the product variable
  for(i in v){

    if(i < 0) return(paste("Not computable because there are non-positives values."))

    else g_prod <- g_prod * i
  }

  # Return geometric mean
  g_prod^(1/length(v))
}
```

(b) Test your code on the following cases and show the output: (i) `{NA, 4, 67, 3}`, (ii) `{0, NA, 6}`, (iii) `{67, 3, Inf}` and (iv) `{-Inf, 67, 3}`

```r
# Test cases for geometric mean function

geometric_mean(c(NA, 4, 67, 3))
```

```
## [1] 9.298624
```

```r
geometric_mean(c(0, NA, 6))
```

```
## [1] 0
```

```r
geometric_mean(c(67, 3, Inf))
```

```
## [1] Inf
```

```r
geometric_mean(c(-Inf, 67, 3))
```

```
## [1] "Not computable because there are non-positives values."
```