HW_1

Carson Pedaci

Question 2

1. The data in the vector are all strings
2.

```
my_vec <- c(
    "+0.07",
    "-0.07",
    "+0.25",
    "-0.84",
    "+0.32",
    "-0.24",
    "-0.97",
    "-0.36",
    "+1.76",
    "-0.36"
)
```

```
my_vec_double <- c()
for (elem in my_vec) {
  my_vec_double <- append(my_vec_double, as.double(elem))
}
my_vec_double
```

```
[1]  0.07 -0.07  0.25 -0.84  0.32 -0.24 -0.97 -0.36  1.76 -0.36
```

```
my_vec_int <- c()
for (elem in my_vec) {
  my_vec_int <- append(my_vec_int, as.integer(elem))
}
```

```
my_vec_int
```

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

3.

```
my_vec_bool <- c()
for (elem in my_vec_double) {
  if (elem >= 0)  {my_vec_bool <- append(my_vec_bool, TRUE)}
  else if (elem <= 0) {my_vec_bool <- append(my_vec_bool, FALSE)}
}
print(my_vec_bool)
```

```
 [1]  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE
```

```
table(my_vec_bool)
```

```
my_vec_bool
FALSE  TRUE 
    6     4 
```

There are four elements greater than 0.

4.

```
sort(my_vec_double)
```

```
 [1] -0.97 -0.84 -0.36 -0.36 -0.24 -0.07  0.07  0.25  0.32  1.76
```

Question 3

1.

```
print(array(c(c(1, 4, 7), c(2, 5, 8), c(3, 6, 9)), dim = c(3,3)))
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

```
matrix <- array(c(1, 1), dim = c(2,1))
for (x in 2:100) {
  matrix <- cbind(matrix, c(x, x**2))
}
matrix
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
[1,]    1    2    3    4    5    6    7    8    9    10    11    12    13    14
[2,]    1    4    9   16   25   36   49   64   81   100   121   144   169   196
     [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
[1,]    15    16    17    18    19    20    21    22    23    24    25    26
[2,]   225   256   289   324   361   400   441   484   529   576   625   676
     [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]
[1,]    27    28    29    30    31    32    33    34    35    36    37    38
[2,]   729   784   841   900   961  1024  1089  1156  1225  1296  1369  1444
     [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50]
[1,]    39    40    41    42    43    44    45    46    47    48    49    50
[2,]  1521  1600  1681  1764  1849  1936  2025  2116  2209  2304  2401  2500
     [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61] [,62]
[1,]    51    52    53    54    55    56    57    58    59    60    61    62
[2,]  2601  2704  2809  2916  3025  3136  3249  3364  3481  3600  3721  3844
     [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74]
[1,]    63    64    65    66    67    68    69    70    71    72    73    74
[2,]  3969  4096  4225  4356  4489  4624  4761  4900  5041  5184  5329  5476
     [,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82] [,83] [,84] [,85] [,86]
[1,]    75    76    77    78    79    80    81    82    83    84    85    86
[2,]  5625  5776  5929  6084  6241  6400  6561  6724  6889  7056  7225  7396
     [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96] [,97] [,98]
[1,]    87    88    89    90    91    92    93    94    95    96    97    98
[2,]  7569  7744  7921  8100  8281  8464  8649  8836  9025  9216  9409  9604
     [,99] [,100]
[1,]    99    100
[2,]  9801  10000
```

```r
generate_matrix <- function(n){
    return(
        matrix(
            rnorm(n^2),
            nrow=n
        )
    )
}
```

2.

```r
row_wise_scan <- function(x){
    n <- nrow(x)
    m <- ncol(x)

    # Insert your code here
    count <- 0
    for(row in 1:n){
        for(col in 1:m){
            if(x[row,col] >= 0){
                count <- count + 1
            }
        }
    }

    return(count)
}
```

```r
M <- generate_matrix(3)
print(M)
```

```
           [,1]       [,2]       [,3]
[1,] -1.1990095 -0.1738400 -0.7515652
[2,] -1.8765217 -3.1524156 -1.0560054
[3,] -0.2364246 -0.6748682 -0.1682072
```

```r
row_wise_scan(M)
```

```
[1] 0
```

3.

```
col_wise_scan <- function(x){
    n <- nrow(x)
    m <- ncol(x)

    # Insert your code here
    count <- 0
    for(col in 1:m){
        for(row in 1:n){
            if(x[row, col] >= 0){
                count <- count + 1
            }
        }
    }

    return(count)
}
```

```
M <- generate_matrix(3)
print(M)
```

```
           [,1]     [,2]       [,3]
[1,] -1.0332235 2.393234 -0.6548980
[2,]  0.5707743 1.163354 -0.8454428
[3,] -1.1323271 1.356356  0.3841753
```

```
col_wise_scan(M)
```

```
[1] 5
```

4.

I believe that col_wise_scan takes shorter to run. R stores elements in matrices by columns not rows. Therefore, col_wise_scan would look at the next memory space for adjacent elements . In row_wise_scan, adjacent elements are stored at some given interval of spaces away from each other. Therefore, the memory keeps having to jump back and forth to find the next element of the matrix.

5.

```r
time_scan <- function(f, M){
    initial_time <- Sys.time()
    f(M)
    final_time <- Sys.time()

    total_time_taken <- final_time - initial_time
    return(total_time_taken)
}
```

```r
M <- generate_matrix(50)
list(
    row_wise_time = time_scan(row_wise_scan, M),
    col_wise_time = time_scan(col_wise_scan, M)
)
```

```
$row_wise_time
Time difference of 0.00100112 secs

$col_wise_time
Time difference of 0 secs
```

Col_wise_time took longer to run.

6.

```r
list(
  row_wise_time = time_scan(row_wise_scan, generate_matrix(100)),
    col_wise_time = time_scan(col_wise_scan, generate_matrix(100))
)
```

```
$row_wise_time
Time difference of 0.002453089 secs

$col_wise_time
Time difference of 0.001950026 secs
```

```r
list(
  row_wise_time = time_scan(row_wise_scan, generate_matrix(1000)),
    col_wise_time = time_scan(col_wise_scan, generate_matrix(1000))
)
```

```
$row_wise_time
Time difference of 0.1350701 secs

$col_wise_time
Time difference of 0.1372199 secs
```

```r
list(
  row_wise_time = time_scan(row_wise_scan, generate_matrix(5000)),
    col_wise_time = time_scan(col_wise_scan, generate_matrix(5000))
)
```

```
$row_wise_time
Time difference of 3.048355 secs

$col_wise_time
Time difference of 2.704967 secs
```

Row_wise_time takes less time to compute with smaller matrices, but as the dimensionality increases, col_wise time takes increasingly less time to compute compared to row_wise_time.