STAT243 - Problem Set 4 P4 (PIKK)

cpaismz

11 de octubre de 2017

Problem 4

In this problem, we will improve two functions: (1) The *PIKK()* function that returns k random values without replacement from a vector x, where $k \leq n$, and n the size of the x vector, and (2) the *FYKD()* function that returns the same output as the previous function but using an iterative shuffling algorithm in order to generate and extract the k random values from a vector x without replacement.

In order to improve them, different approaches and modifications of the original functions are implemented. The main strategies consist of eliminating extra calculations/steps as well as modifying/replacing ineficcient computations by more efficient expressions. Due to their structure and AS-IS state, both functions can be easily improved by taking into account the previous points.

a.1) Improving PIKK function

Looking at the original function, we can clearly identify some performance issues that can be tackled in order to improve the formulation: (1) Besides k values are returned as the main output of the function, n random numbers are generated, (2) Sorting function is needed in order to obtain a random permutation based on the random numbers generated. These two operations can be easily improved using a stochastic approach such that the expected number of iterations (and therefore running time) is clearly less than the original formulation.

We develop three new functions that we are going to compare with the original $PIKK()$ function in terms of running time:

i) $PIKKV1()$: Poor performance (slow) implementation using loops for comparison purposes. The idea is to show how we can develop a very bad formulation of the same problem depending on the approach used to tackle the problem.

ii) $PIKKV2()$: High performance implementation based on $runif()$ function (random number generations) transformed into a k-indexes array.

iii) $PIKKVF()$: High performance implementation based $PIKKV2()$ with an extra checking step.

In addition, we include the R *sample*() function version in our benchmarks for comparison purposes (more details in section b).

1. The logic of our first function consists of an iterative approach where each element inside the x vector has a chance of being selected equal to $\frac{k_i}{n_i}$ where k_i is equal to the remaining number of elements at iteration i that must be selected in order to get k entries and n_i is equal to the remaining number of entries inside the vector x (not yet iterated). Thus, $n_i = n - i + 1$ starting from iteration $i = 1$ and the value of k_i will be updated every iteration depending on the probability of picking that entry.

   Clearly, this function will have a poor performance due to the number of steps, calculations, and ineficient use of a for loop included. However, we are including it for comparison purposes in our analysis.

```r
# First new function: PIKKV1
PIKKV1 <- function(x, k){
  # Initialization of selected numbers array and size variable
  sel = integer(k)
  size = k

  # Variable for keeping track of the current not visited entries of vector x
  left = length(x)

  # For loop: select elements based on k_{i}/n_{i} probability, ending up with the
  # desire number of entries k
  aux = 1
  for (i in 1:length(x)){

    # Probability of selecting an element
    if (runif(1) <= size / left){
      size = size - 1
      sel[aux] <- x[i]
      aux = aux + 1
    }

    # Break if all k entries have been selected
    if (size == 0){
      break
    }

    # Update the remaining number of entries
    left = left - 1
  }

  # Return selected values (k)
  return(sel)
}
```

2. The second function $PIKKV2()$ is very simple but powerful. In this case, we are generating a set of k random numbers between 1 and the length of the x vector, such that we end up with a k size array. Then, a ceiling operation is performed to all the elements of the vector, obtaining integer values. This allows us to use the $unique()$ function in order to keep only the unique elements of the array. If the size of the resulting vector is k, we have k different indexes and the resulting vector is returned, otherwise, a new vector is generated until the k unique elements condition is satisfied.

   Clearly, this function will not have a very good performance when increasing the value of k (when it tends to $length(x)$) since the probabilities of obtaining k unique values are decreasing. However, since we are going to concentrate on values of k where k is much less than n (as indicated in the problem statement), it will reach a very good performance in comparison to the original implementation.

   Note that we use the ceiling function for showing purposes only since we would be able to skip it by remembering that non-integer values are rounded down (floor operation) automatically when they are used as indexes. In the same logic, all functions can be binarized (using the R compiler) in order to obtain an extra speed-up, however, since we want to test pure R code we are not implementing this "trick" in our analysis.

```r
# PIKK Version 2 function
PIKKV2 <- function(x, k){
```

```
  # While loop: True until condition is satisfied
  while(TRUE){

    # Indexes array (unique elements) is generated
    indexes <- unique(ceiling(runif(k, 0, length(x))))

    # If size is k: stop, otherwise repeat procedure
    if (length(indexes) == k) {
      break
    }
  }

  # Return relevant (k) entries
  return(x[indexes])
}
```

3. Based on the previous function, we define $PIKKVF()$ function with a different approach for dealing with the remaining indexes where no unique k indexes are generated. In this case, we include two hyper-parameters that will add flexibility to the proposed function: (1) A *threshold* value that determines when to generate more than k numbers based on the proportion of total elements (size of x) and k in such a way that if k is larger, then we will have a better chance of getting k unique random numbers from a larger set, and (2) *tune* parameter that determines the number of extra random numbers to be generated. Its current value (6) was obtained based on a series of experiments as can be seen in the optimizePIKKVF.R file inside the Github repository.

   Based on its structure, it would clearly have a better performance than the previous algorithms when dealing with higher values for k, thanks to the new extra checking step.

```
# PIKK VF function
PIKKVF <- function(y, k, threshold = 800, tune = 0){
  # Check if extra terms are added for increasing the probability of success
  ifelse(length(y)/k <= threshold, tune <- k/6, tune <- 0)

  # Indexes loop
  while (TRUE){
    indexes = unique(ceiling(runif(k + tune, 0, length(y)) ))
    if (length(indexes) >= k){
      return(y[indexes[1:k]])
    }
  }
}
```

4. For completeness and comparison purposes, we define a wrapper for the R *sample*() function, such that we can compare all the proposed functions and the original one provided in the statement of the problem with this internal (and very efficient) sampling function. This will be the first step before performing a detailed comparison between our best function and the default R *sample*() function in section b.1).

```
# R Sample function wrapper
RSample <- function(x, k){
  return(sample(x = x, k, replace = FALSE))
}
```

5. Finally, we declare the original function in order to be able to produce the comparisons.

```r
# Original PIKK implementation
PIKK <- function(x, k) {
  x[sort(runif(length(x)), index.return = TRUE)$ix[1:k]]
}
```

6. In order to perform the comparisons, we proceed as follows:

   i) Preliminar microbenchmark tests and plot are developed comparing all functions (including the PIKKV1 poor performance function).

   ii) A formal comparison for a series of values of k and n are performed for: $PIKK$, $PIKKV2$, $PIKKVF$, and $RSample()$ functions. Different values of n and k will be tested, focusing our analysis in both values of k that are much less than n (up to 1/100 of the maximum value) and values of k that are more similar to the size of the vector (up to 1/10 of the maximum value).

   iii) Specific comparison between our best performance function (as we will see, $PIKKVF()$) and the very efficient $RSample()$ function will be performed following the same analysis for k and n values (section b).

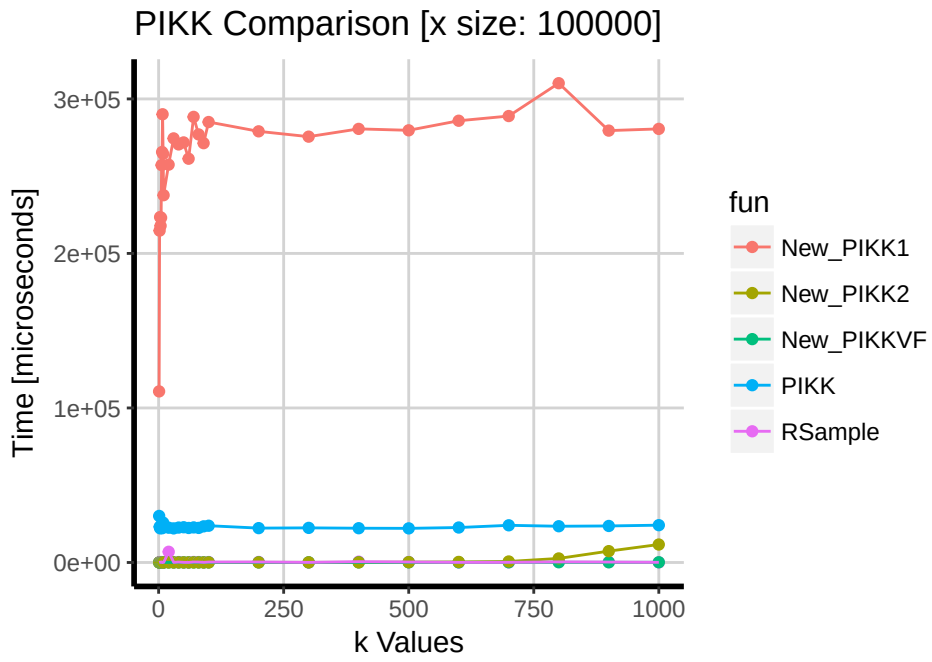
   Thus, we will declare and load a series of plotting functions (for comparison purposes) that are available inside the file plotPIKKVF.R in the Github repository. These functions can be found in the Appendix section. For completeness, we also include a microbenchmark wrapper function that can be easily used with a $lapply()$ command for generating a series of comparisons (check appendix for more details).

```r
# Load relevant plot functions
options(warn=-1)
setwd("C:/Users/chile/Desktop/Stats243/HW/HW4/Code/Submitting/")
source("plotPIKKVF.R")

# Microbenchmark wrapper for comparisons: inputs(vector, number of elements,
# time unit string, times)
MicroExp <- function(x, k, tu = NULL, ntimes = 5){
  if (!is.null(tu)){
    # Print results using specific time unit
    print(microbenchmark(PIKK(x, k), PIKKV1(x, k), PIKKV2(x, k),
                         PIKKVF(x, k), RSample(x, k), unit = tu,
                         times = ntimes))
  }

  # Print results using default time unit
  else{
    print(microbenchmark(PIKK(x, k), PIKKV1(x, k), PIKKV2(x, k),
                         PIKKVF(x, k), RSample(x, k), times = ntimes))
  }
}
```

7. Preliminary comparisons are performed following the values recommendation from the problem statement. Plots are generated for visualization purposes.

```r
# Loading libraries
library("microbenchmark")
library("ggplot2")
library("tidyr")
```

```r
## Preliminar Test
#Testing Data: x size 1e5
x <- 1:1e5

# Up to 1/100 of the total size
ks  <- c(seq(1, 10, 1), seq(20, 100, 10), seq(200, length(x) / 100, 100))

# All values plot & microbenchmark comparison for the largest k
plotComparisonPIKKAll_A(x, ks, ntimes = 5)
```




```r
MicroExp(x, ks[length(ks)])
```

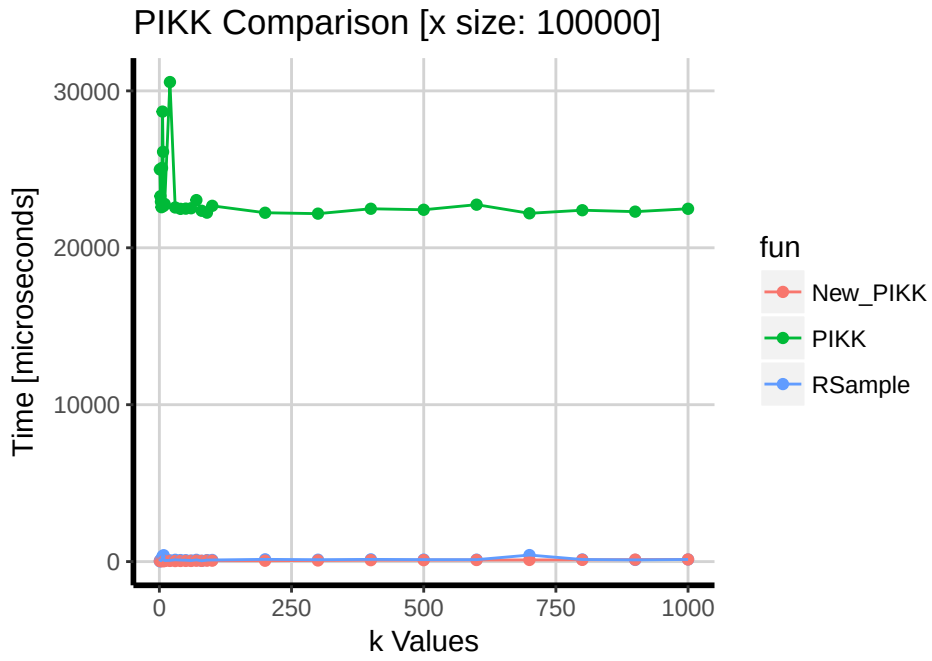
```
## Unit: microseconds
##           expr        min         lq        mean     median         uq
##     PIKK(x, k)  21954.845  22321.343  23200.8542  22347.002  23245.929
##   PIKKV1(x, k) 265031.037 266186.984 283384.0178 269224.601 306720.352
##   PIKKV2(x, k)    848.036   2747.243  10808.0724   4843.598  21160.694
##   PIKKVF(x, k)    119.316    124.020    137.5340    131.290    145.403
##  RSample(x, k)     94.939    102.637    432.5286    172.344    172.345
##         max neval
##   26135.152     5
##  309757.115     5
##   24440.791     5
##     167.641     5
##    1620.378     5
```

Looking at the results, we can easily check our initial thought about our $PIKKV1()$ function: it is by far the less efficient of all the tested ones. Important is to notice that both $PIKKV2()$ and $PIKKVF()$ reached a far better performance than the original $PIKK()$ function. In addition, we can clearly see that these two functions follow a similar pattern to the one showed by the $RSample()$ function, telling us that we should focus our analysis on them.

The microbenchmark analysis are very clear: (1) $PIKKV1()$ is the worst function (in terms of performance) by 1 order of magnitude in comparison to the original $PIKK()$ function, (2) the original function is the second worst in terms of running time, reaching mean values around 24 milliseconds while (3) $PIKKV2()$ needs around 11 milliseconds and (4) $PIKKVF()$ only needs around 140 microseconds, pretty similar to the performance reached by the very efficient $RSample()$ function.

Thanks to this preliminary analysis, we are able to illustrate how important is the structure of a function and the approach for solving the problem under study in order to reach the best possible performance. Now, we will perform the rest of our analysis without taking $PIKKV1()$ function into account, due to its poor performance. Under the same logic, since $PIKKVF()$ is an enhanced version of $PIKKV2()$, we will also drop this last function from the rest of our analysis.

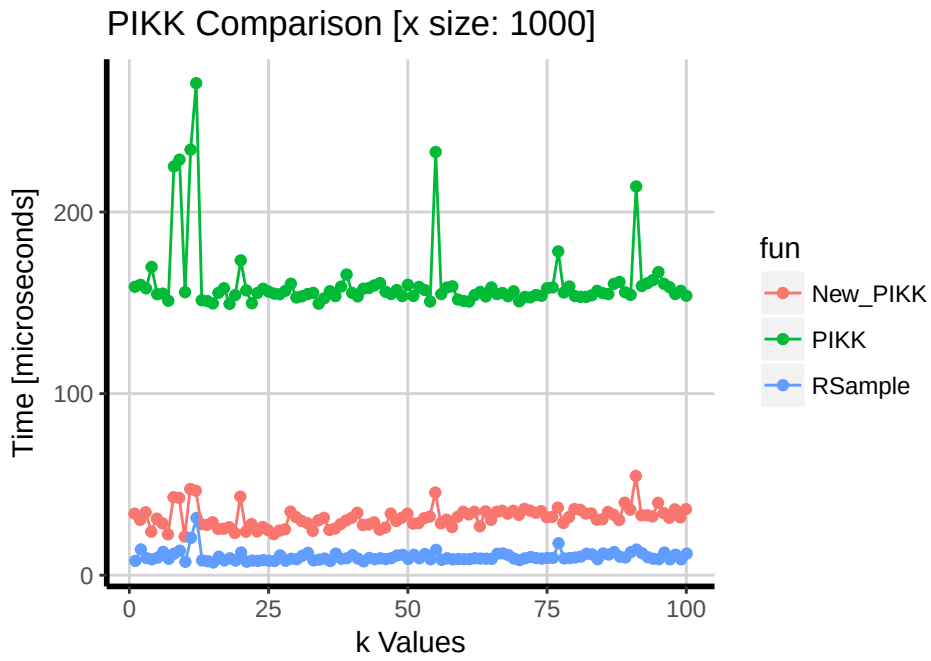
```
# Focus on cases up to 1/100 of the total size without PIKKV1()
ks  <- c(seq(1, 10, 1), seq(20, 100, 10), seq(200, length(x) / 100, 100))

# All values plot & microbenchmark comparison for the largest k
plotComparisonAll(x, ks, ntimes = 5)
```




```
microbenchmark(PIKK(x, ks[length(ks)]), RSample(x, ks[length(ks)]),
               PIKKVF(x, ks[length(ks)]), times = 5)
```

```
## Unit: microseconds
##                         expr       min        lq       mean    median
##     PIKK(x, ks[length(ks)]) 22378.649 22693.402 23077.6906 22895.681
##  RSample(x, ks[length(ks)])    98.361    99.644   128.9808   117.605
##   PIKKVF(x, ks[length(ks)])   115.467   121.454   155.0676   160.371
##         uq       max neval
##  23050.492 24370.229     5
##    156.522   172.772     5
##    183.891   194.155     5
```

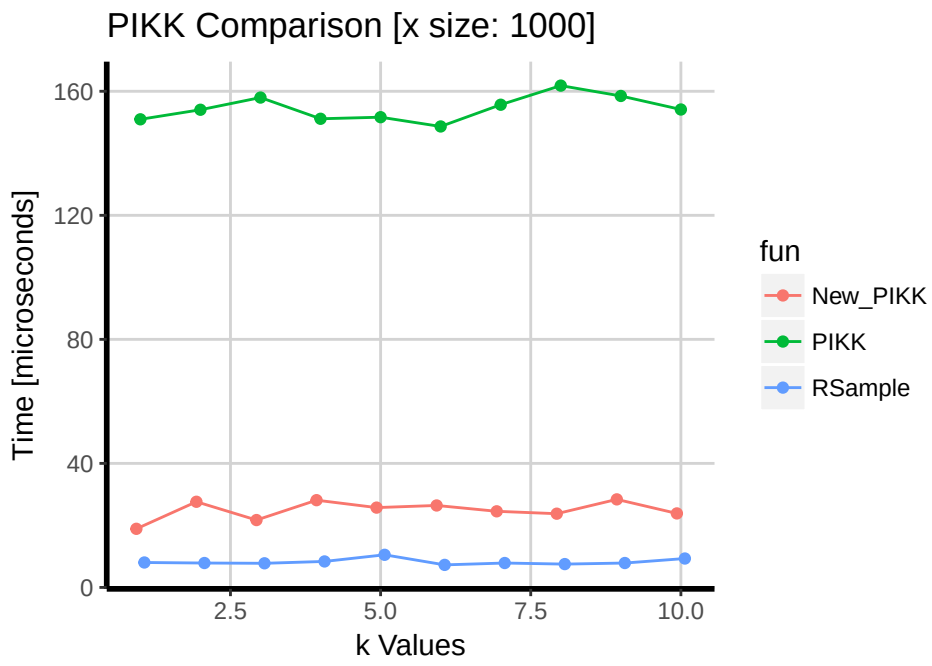
```r
# Testing Data: x size 1e3
x <- 1:1e3

# Up to 1/10 of the total size
ks <- c(seq(1, length(x) / 10, 1))

# All values plot & microbenchmark comparison for the largest k
plotComparisonAll(x, ks, ntimes = 5)
```




```r
microbenchmark(PIKK(x, ks[length(ks)]), RSample(x, ks[length(ks)]),
               PIKKVF(x, ks[length(ks)]), times = 5)
```

```
## Unit: microseconds
##                         expr     min      lq     mean  median      uq
##     PIKK(x, ks[length(ks)]) 144.547 154.383 163.7916 157.377 180.043
##  RSample(x, ks[length(ks)])   8.126   9.409  10.5208   9.836  11.547
##   PIKKVF(x, ks[length(ks)])  29.508  35.068  44.3054  35.496  51.747
##      max neval
##  182.608     5
##   13.686     5
##   69.708     5
```

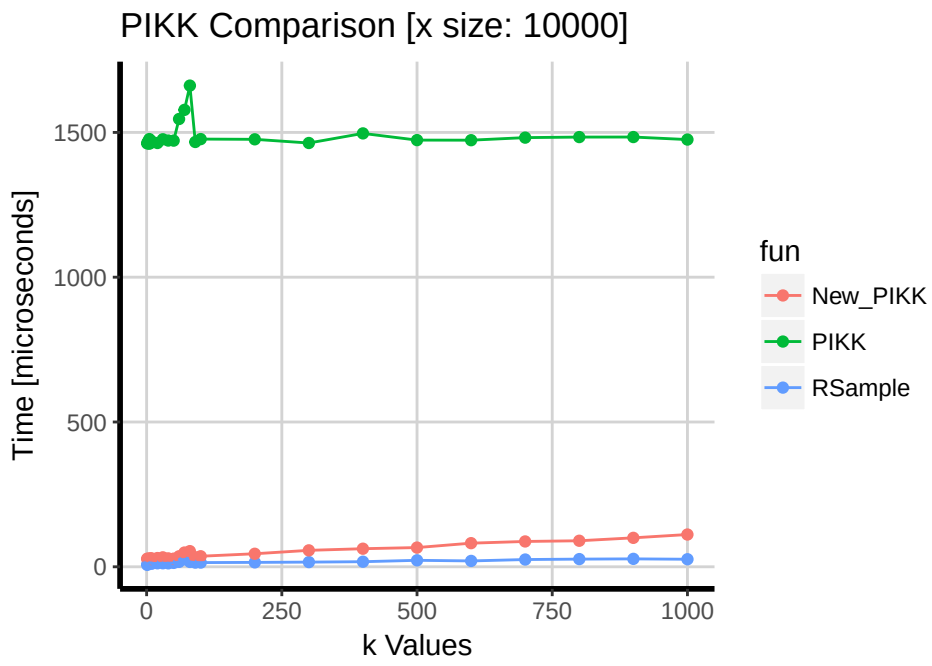
```r
# Focus on cases up to 1/100 of the total size
ks  <- c(seq(1, length(x) / 100, 1))

# All values plot & microbenchmark comparison for the largest k
plotComparisonAll(x, ks, ntimes = 5)
```




```
microbenchmark(PIKK(x, ks[length(ks)]), RSample(x, ks[length(ks)]),
               PIKKVF(x, ks[length(ks)]), times = 5)
```

```
## Unit: microseconds
##                         expr     min      lq     mean  median      uq
##     PIKK(x, ks[length(ks)]) 140.271 140.698 151.5610 141.554 144.975
##  RSample(x, ks[length(ks)])   6.415   7.271  11.4616   8.126   8.126
##   PIKKVF(x, ks[length(ks)])  15.824  19.245  27.7980  23.521  38.917
##      max neval
##  190.307     5
##   27.370     5
##   41.483     5
```

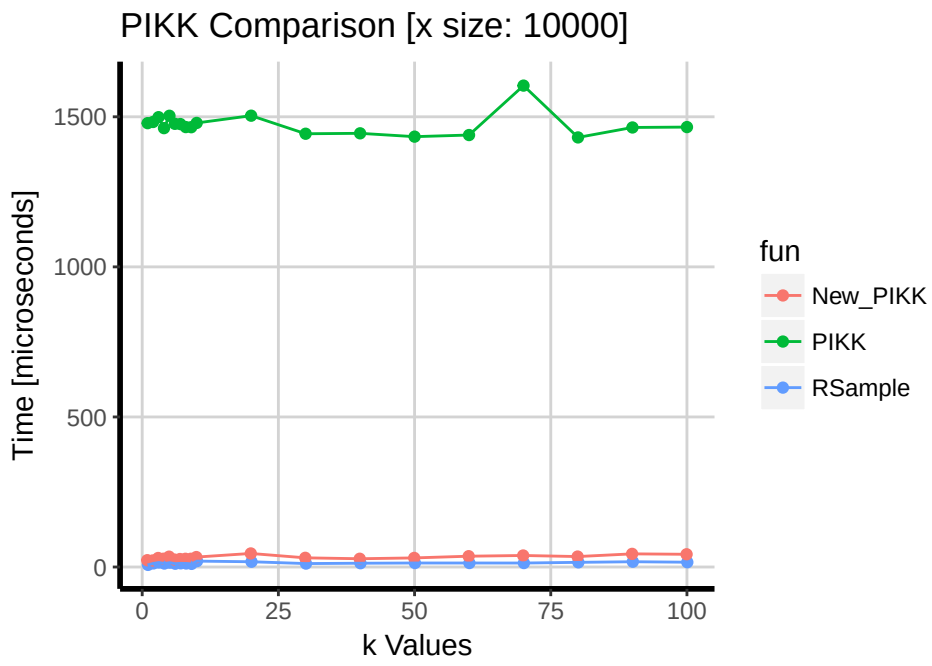
```
# Testing Data: x size 1e4
x <- 1:1e4

# Up to 1/10 of the total size
ks  <- c(seq(1, 10, 1), seq(20, 100, 10), seq(200, length(x) / 10, 100))

# All values plot & microbenchmark comparison for the largest k
plotComparisonAll(x, ks, ntimes = 5)
```




```
microbenchmark(PIKK(x, ks[length(ks)]), RSample(x, ks[length(ks)]),
               PIKKVF(x, ks[length(ks)]), times = 5)
```

```
## Unit: microseconds
##                         expr      min       lq      mean   median       uq
##     PIKK(x, ks[length(ks)]) 1473.693 1474.547 1523.3002 1479.252 1522.017
##  RSample(x, ks[length(ks)])   22.666   23.949   26.7718   26.516   29.936
##   PIKKVF(x, ks[length(ks)])   97.078  100.072  109.4798  102.637  103.065
##       max neval
##  1666.992     5
##    30.792     5
##   144.547     5
```

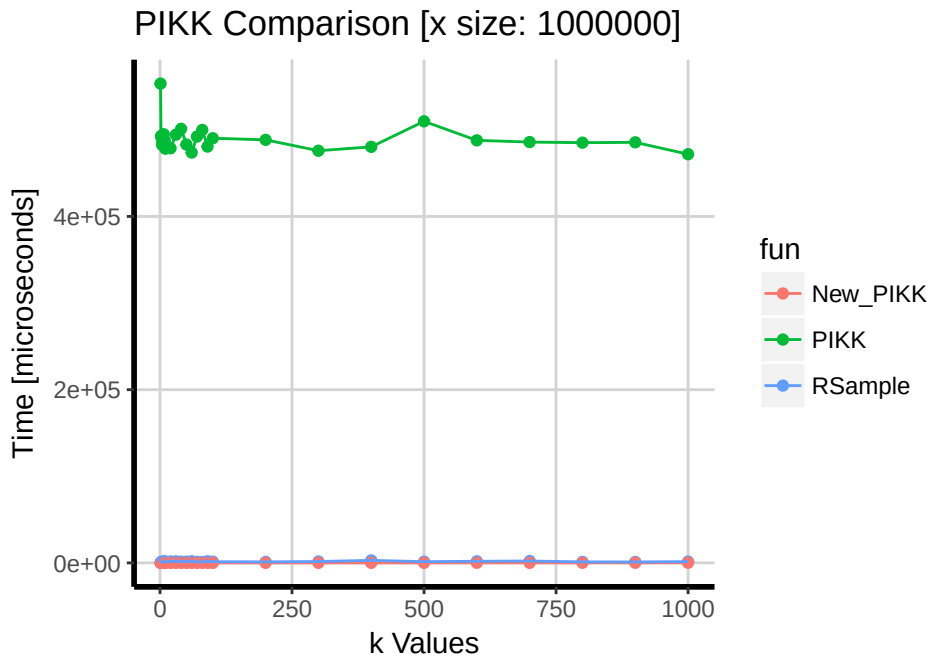
```
# Focus on cases up to 1/100 of the total size
ks  <- c(seq(1, 10, 1), seq(20, length(x) / 100, 10))

# All values plot & microbenchmark comparison for the largest k
plotComparisonAll(x, ks, ntimes = 5)
```




```
microbenchmark(PIKK(x, ks[length(ks)]), RSample(x, ks[length(ks)]),
               PIKKVF(x, ks[length(ks)]), times = 5)
```

```
## Unit: microseconds
##                         expr      min       lq      mean   median       uq
##     PIKK(x, ks[length(ks)]) 1408.689 1464.712 1466.5934 1470.271 1480.535
##  RSample(x, ks[length(ks)])   10.264   13.258   15.9094   14.969   15.824
##   PIKKVF(x, ks[length(ks)])   26.515   27.370   42.9366   34.640   50.463
##       max neval
##  1508.760     5
##    25.232     5
##    75.695     5
```

```
## Testing Data: x size 1e6
x <- 1:1e6

# Up to k = 1000
ks <- c(seq(1, 10, 1), seq(20, 100, 10), seq(200, 1000, 100))

# All values plot & microbenchmark comparison for the largest k
plotComparisonAll(x, ks, ntimes = 5)
```




```
microbenchmark(PIKK(x, ks[length(ks)]), RSample(x, ks[length(ks)]),
               PIKKVF(x, ks[length(ks)]), times = 5)
```

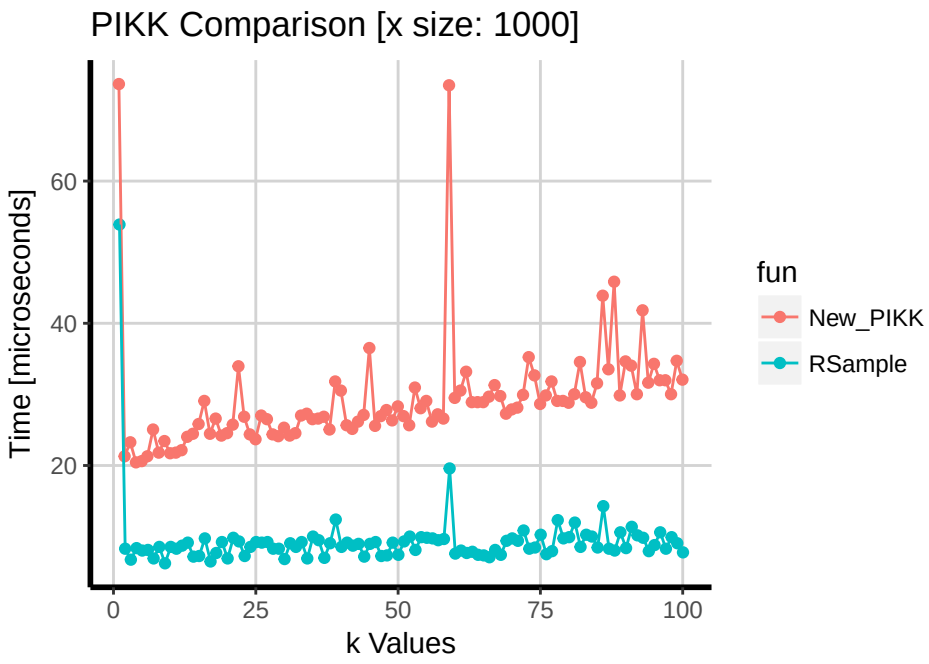
```
## Unit: microseconds
##                         expr        min         lq        mean     median
##     PIKK(x, ks[length(ks)]) 462041.744 464282.217 488550.2360 500899.996
##  RSample(x, ks[length(ks)])   1036.204   1046.895   1109.4178   1086.666
##   PIKKVF(x, ks[length(ks)])    145.403    152.245    163.6206    161.653
##          uq        max neval
##  505714.085 509813.138     5
##    1185.454   1191.870     5
##     162.081    196.721     5
```

Based on all the previous results, we can conclude that the $PIKKVF()$ function is far better in terms of running time performance in comparison to the original $PIKK()$ function. Furthermore, we see that it follows a very similar performance pattern to the one obtained by the $RSample()$ function, allowing us to perform a specific analysis of these two functions in the next section, where we will check that $PIKKVF()$ will never take longer than 3-4 times the time of the $RSample()$ function when solving instances where k is much less than n as expected in the problem statement.

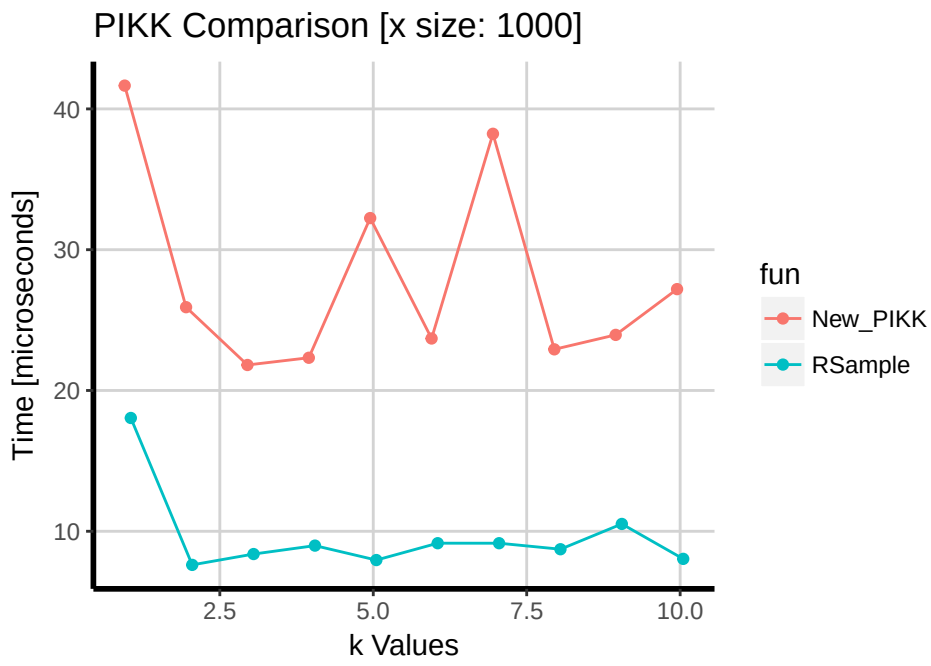
b.1) PIKKVF vs RSample function

In order to perform a specific comparison analysis between these two functions, we perform similar experiments to the ones shown above including new values for n and k. A series of plots are included for simplicity instead of multiple microbenchmark outputs.

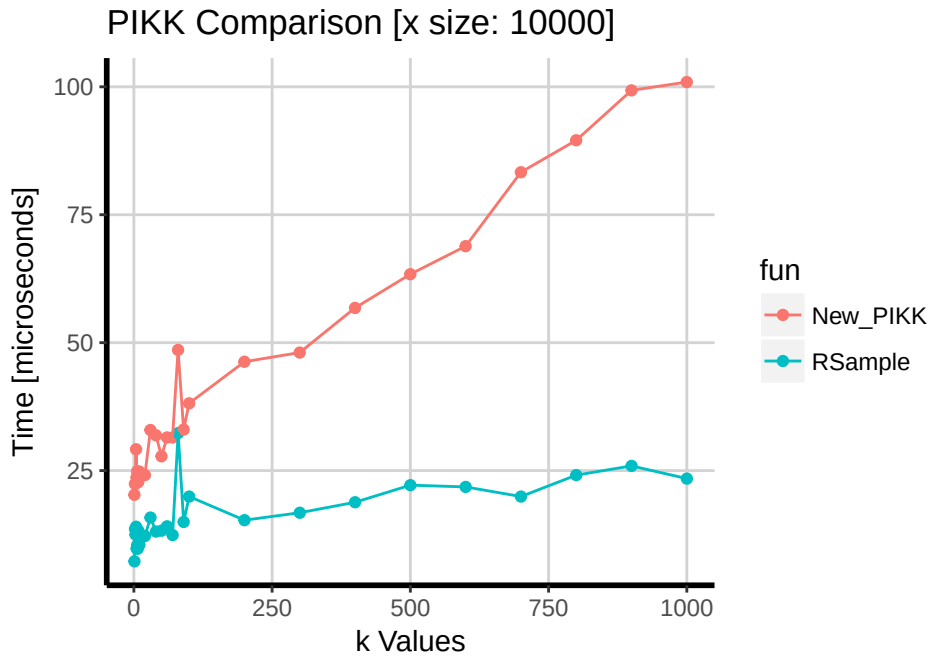
```
# Testing Data: x size 1e3
x <- 1:1e3

# Up to 1/10 of the total size
ks <- c(seq(1, length(x) / 10, 1))

# Plot comparison vs RSample
plotComparison(x, ks, ntimes = 5)
```



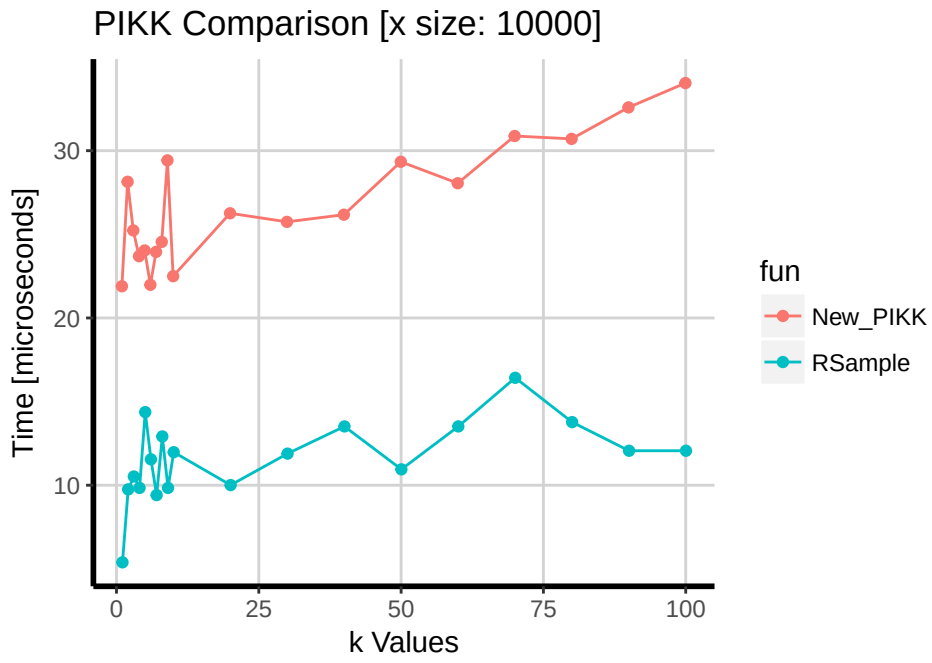

```
# Focus on cases up to 1/100 of the total size
ks  <- c(seq(1, length(x) / 100, 1))

# Plot comparison vs RSample
plotComparison(x, ks, ntimes = 5)
```



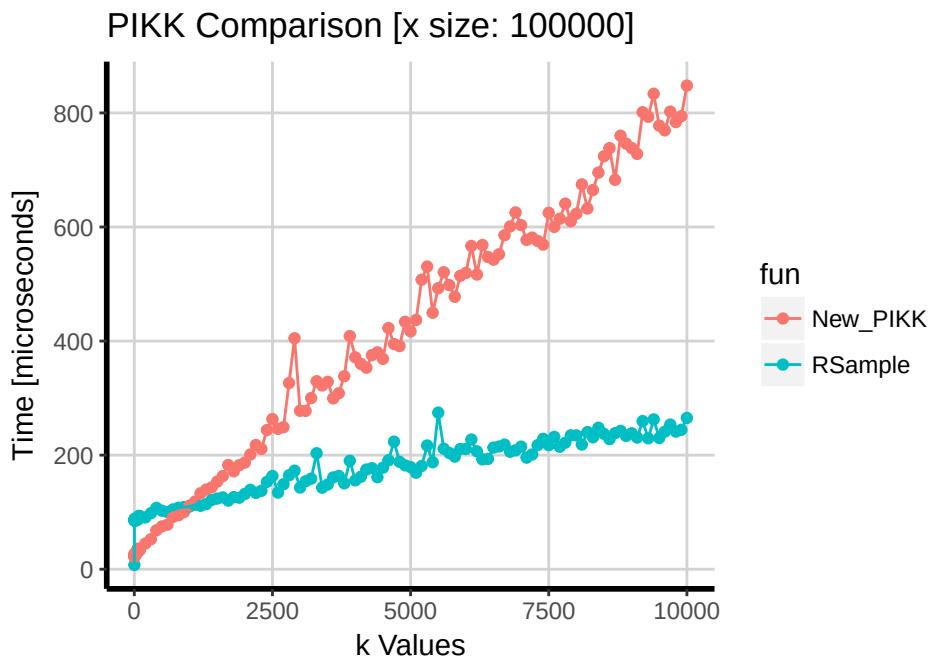

```r
# Testing Data: x size 1e4
x <- 1:1e4

# Up to 1/10 of the total size
ks  <- c(seq(1, 10, 1), seq(20, 100, 10), seq(200, length(x) / 10, 100))

# Plot & microbenchmark comparisons
plotComparison(x, ks, ntimes = 5)
```



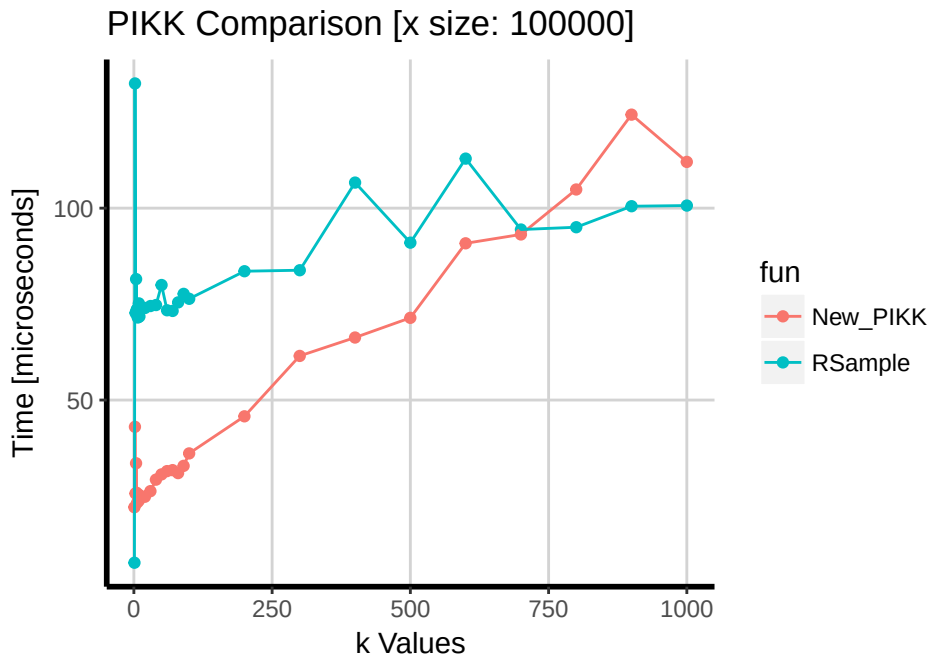
```r
# Focus on cases up to 1/100 of the total size
ks  <- c(seq(1, 10, 1), seq(20, length(x) / 100, 10))

# Plot & microbenchmark comparisons
plotComparison(x, ks, ntimes = 5)
```




```r
# Testing Data: x size 1e5
x <- 1:1e5

# Up to 1/10 of the total size
ks  <- c(seq(1, 10, 1), seq(20, 100, 10), seq(200, length(x) / 10, 100))

# Plot & microbenchmark comparisons
plotComparison(x, ks, ntimes = 5)
```

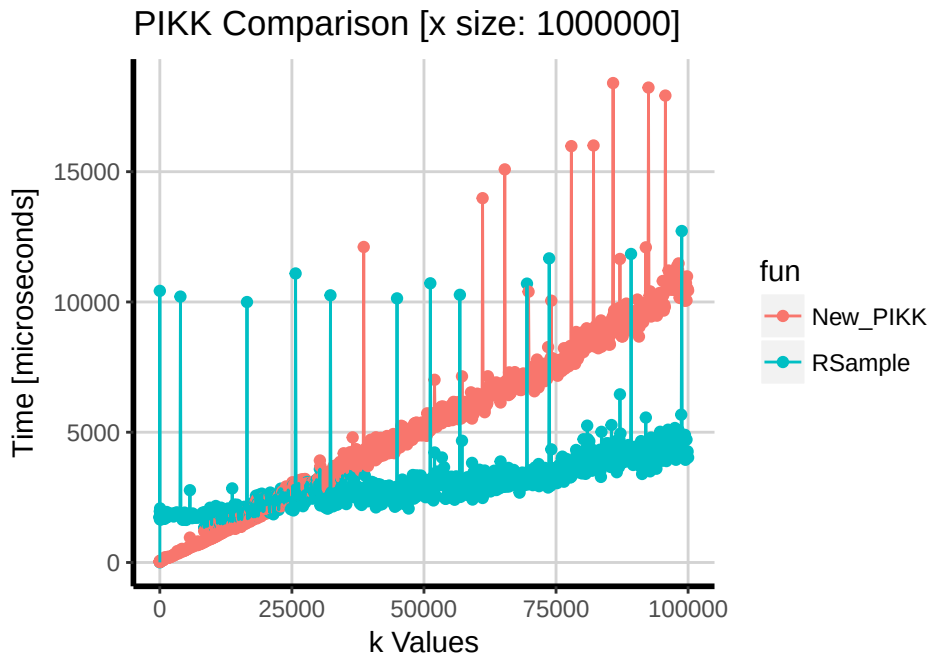



```
# Focus on cases up to 1/100 of the total size
ks  <- c(seq(1, 10, 1), seq(20, 100, 10), seq(200, length(x) / 100, 100))

# Plot & microbenchmark comparisons
plotComparison(x, ks, ntimes = 5)
```

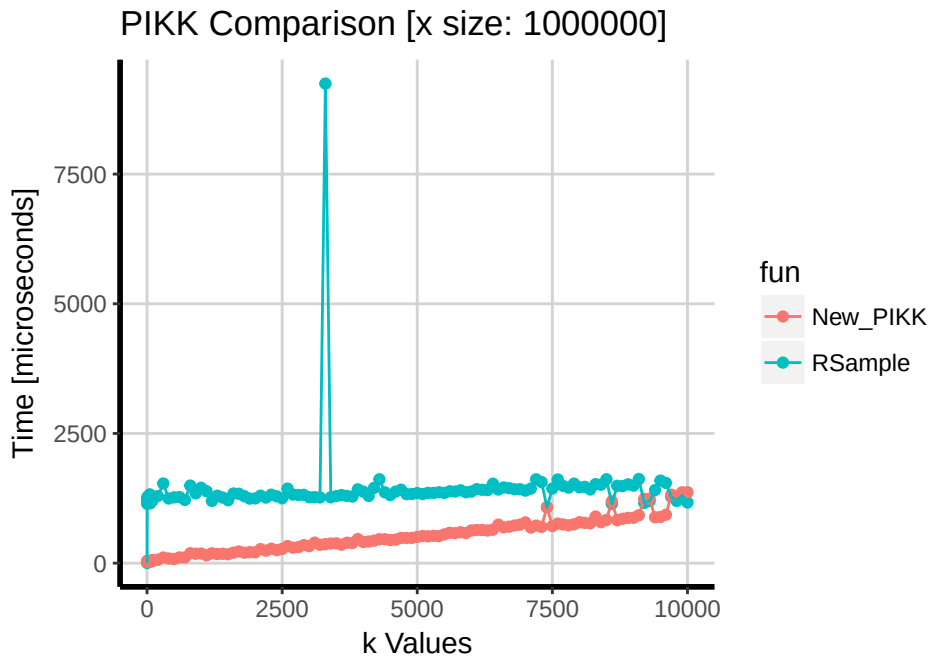



```
## Testing Data: x size 1e6
x <- 1:1e6
```

```r
# Up to 1/10 of the total size
ks <- c(seq(1, 10, 1), seq(20, 100, 10), seq(200, length(x) / 10, 100))

# Plot & microbenchmark comparisons
plotComparison(x, ks, ntimes = 5)
```




```r
# Focus on cases up to 1/100 of the total size
ks  <- c(seq(1, 10, 1), seq(20, 100, 10), seq(200, length(x) / 100, 100))

# Plot & microbenchmark comparisons
plotComparison(x, ks, ntimes = 5)
```



Based on the previous results, it is clear that our $PIKKVF()$ function is able to "defeat" the $RSample()$ function for some instances and it is always below 3-4 times the running time of this very efficient function for instances where k is much less than n, as asked and expected.