ps4

Hangyu Huang

10 October 2017

Contents

0.1 Partner

Partner: YUE HU

0.2 Problem 1(a)

There is only one copy that exists of the vector 1:10 during the first execution of myFun(). Based on the idea of copy on change, copies of objects are not made until one of the objects is actually modified. Since there is no change on vector 1:10, "data" points to the same memory location as the original object "x".

0.3 Problem 1(b)

The serialized object size is actually twice of the size of object "x". It seems like that the function "serialize" serialized both "x" and "data" to a raw vector as this function can not recognize that "x" and "data" point to the same memory location.

```
x <- 1:1e7
f <- function(input){
        data <- input
        g <- function(param) return(param * data)
        return(g)
        }
myFun <- f(x)
bytes <- serialize(myFun,NULL)
print(object.size(x))
```

```
## 40000040 bytes
```

```
print(object.size(bytes))
```

```
## 80007376 bytes
```

0.4 Problem 1 (c)

Based on "lazy evaluation", when R matches user input arguments to formal argument names, it does not (usually) evaluate the arguments until they are needed.

In this question, "x" is not copied until "myFun" is called at "myFun(3)", however "x" has been already removed. Therefore, R cannot find object 'x'.

0.5 Problem 1 (d)

The resulting serialized closure is 888 bytes

```
x <- 1:10
f <- function(data){
        g <- function(param) return(param * data)
        return(g)
        }
myFun <- f(1:10)
rm(x)
data <- 100
myFun(3)
```

```
##  [1]  3  6  9 12 15 18 21 24 27 30
```

```
bytes <- serialize(myFun,NULL)
print(object.size(bytes))
```

```
## 7744 bytes
```

0.6 problem 2 (a)

When an element of one of the vectors is modified, R makes the change in place without creating a new list or a new vector.

From the result, we can see that 2 vectors are in the same place.

```
library(pryr)
```

```
## Warning: package 'pryr' was built under R version 3.4.2
```

```
##
## Attaching package: 'pryr'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     f
```

```
tmp <- list(rnorm(1e7), rnorm(1e7))
.Internal(inspect(tmp))
```

```
## @0x00000000124c4208 19 VECSXP g0c2 [NAM(2)] (len=2, tl=0)
##   @0x00007ff5fade0010 14 REALSXP g1c7 [MARK] (len=10000000, tl=0) -0.907579,0.83171,-1.00542,-0.4704
```

```
##   @0x00007ff5f6190010 14 REALSXP g0c7 [] (len=10000000, tl=0) 0.115923,-0.652907,1.56478,0.412159,1.1
```

```
mem_change(tmp[[1]][1] <- 1 )
```

```
## 3.91 kB
```

```
.Internal(inspect(tmp))
```

```
## @0x00000000124c44e0 19 VECSXP g1c2 [MARK,NAM(1)] (len=2, tl=0)
##   @0x00007ff5f1540010 14 REALSXP g1c7 [MARK] (len=10000000, tl=0) 1,0.83171,-1.00542,-0.470468,-0.205
##   @0x00007ff5f6190010 14 REALSXP g1c7 [MARK,NAM(2)] (len=10000000, tl=0) 0.115923,-0.652907,1.56478,0
```

0.7 problem 2 (b)

There is no copy-on-change going on when I make a copy of the list. However, a copy of the relevant vector is made when a change is made to one of the vectors in one of the lists.

From the result we can find that only the place of vector [1] of "copyofTmp" is changed and the memory is increased almost 80Mb, which is the size of one vector.

```
library(pryr)
mem_change(copyofTmp <- tmp)
```

```
## 296 B
```

```
.Internal(inspect(tmp))
```

```
## @0x00000000124c44e0 19 VECSXP g1c2 [MARK,NAM(2)] (len=2, tl=0)
##   @0x00007ff5f1540010 14 REALSXP g1c7 [MARK] (len=10000000, tl=0) 1,0.83171,-1.00542,-0.470468,-0.205
##   @0x00007ff5f6190010 14 REALSXP g1c7 [MARK,NAM(2)] (len=10000000, tl=0) 0.115923,-0.652907,1.56478,0
```

```
.Internal(inspect(copyofTmp))
```

```
## @0x00000000124c44e0 19 VECSXP g1c2 [MARK,NAM(2)] (len=2, tl=0)
##   @0x00007ff5f1540010 14 REALSXP g1c7 [MARK] (len=10000000, tl=0) 1,0.83171,-1.00542,-0.470468,-0.205
##   @0x00007ff5f6190010 14 REALSXP g1c7 [MARK,NAM(2)] (len=10000000, tl=0) 0.115923,-0.652907,1.56478,0
```

```
mem_change(copyofTmp[[1]][1] <- 2 )
```

```
## 80 MB
```

```
.Internal(inspect(tmp))
```

```
## @0x00000000124c44e0 19 VECSXP g1c2 [MARK,NAM(2)] (len=2, tl=0)
##   @0x00007ff5f1540010 14 REALSXP g1c7 [MARK,NAM(2)] (len=10000000, tl=0) 1,0.83171,-1.00542,-0.470468
##   @0x00007ff5f6190010 14 REALSXP g1c7 [MARK,NAM(2)] (len=10000000, tl=0) 0.115923,-0.652907,1.56478,0
```

```
.Internal(inspect(copyofTmp))
```

```
## @0x00000000124c4470 19 VECSXP g1c2 [MARK,NAM(1)] (len=2, tl=0)
##   @0x00007ff5fade0010 14 REALSXP g1c7 [MARK] (len=10000000, tl=0) 2,0.83171,-1.00542,-0.470468,-0.205
##   @0x00007ff5f6190010 14 REALSXP g1c7 [MARK,NAM(2)] (len=10000000, tl=0) 0.115923,-0.652907,1.56478,0
```

0.8 problem 2 (c)

When the copy of a list of lists "lists", "copyofLists" is made, there is no copy-on-change going on. The two lists point to the same memory location.

When an element is added to the second list, only one copy of the new vector is made for "copyofLists$B", there is no change for the rest of the two lists. 30Mb memory change takes place, which is the size of the added element.

```
library(pryr)
lists <- list(A = list(a = rnorm(1e7), b = rnorm(1e7)),
              B = list(a = rnorm(1e6)))
mem_change(copyofLists <- lists)
```

```
## 768 B
```

```
.Internal(inspect(lists))
```

```
## @0x00000000124c4278 19 VECSXP g1c2 [MARK,NAM(2),ATT] (len=2, tl=0)
##   @0x00000000124c4208 19 VECSXP g1c2 [MARK,ATT] (len=2, tl=0)
##     @0x00007ff5ec8f0010 14 REALSXP g1c7 [MARK] (len=10000000, tl=0) 1.47197,0.912267,2.20222,-1.75208
##     @0x00007ff5e7ca0010 14 REALSXP g1c7 [MARK] (len=10000000, tl=0) -0.802089,0.0725343,-0.252489,-0
##   ATTRIB:
##     @0x0000000012509418 02 LISTSXP g1c0 [MARK]
##       TAG: @0x0000000011da2630 01 SYMSXP g1c0 [MARK,NAM(2),LCK,gp=0x4000] "names" (has value)
##       @0x00000000124c4240 16 STRSXP g1c2 [MARK] (len=2, tl=0)
##  @0x0000000013ce3378 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "a"
##  @0x000000001407f270 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "b"
##   @0x0000000012508438 19 VECSXP g1c1 [MARK,ATT] (len=1, tl=0)
##     @0x00007ff5e74f0010 14 REALSXP g1c7 [MARK] (len=1000000, tl=0) 1.05788,1.20887,-0.816647,-0.0454
##   ATTRIB:
##     @0x0000000012509728 02 LISTSXP g1c0 [MARK]
##       TAG: @0x0000000011da2630 01 SYMSXP g1c0 [MARK,NAM(2),LCK,gp=0x4000] "names" (has value)
##       @0x0000000012508468 16 STRSXP g1c1 [MARK] (len=1, tl=0)
##  @0x0000000013ce3378 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "a"
## ATTRIB:
##   @0x0000000012509798 02 LISTSXP g1c0 [MARK]
##     TAG: @0x0000000011da2630 01 SYMSXP g1c0 [MARK,NAM(2),LCK,gp=0x4000] "names" (has value)
##     @0x00000000124c42b0 16 STRSXP g1c2 [MARK] (len=2, tl=0)
##       @0x00000000160b8678 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "A"
##       @0x00000000124f00c0 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "B"
```

```
.Internal(inspect(copyofLists))
```

```
## @0x00000000124c4278 19 VECSXP g1c2 [MARK,NAM(2),ATT] (len=2, tl=0)
##   @0x00000000124c4208 19 VECSXP g1c2 [MARK,ATT] (len=2, tl=0)
##     @0x00007ff5ec8f0010 14 REALSXP g1c7 [MARK] (len=10000000, tl=0) 1.47197,0.912267,2.20222,-1.75208
##     @0x00007ff5e7ca0010 14 REALSXP g1c7 [MARK] (len=10000000, tl=0) -0.802089,0.0725343,-0.252489,-0
##   ATTRIB:
##     @0x0000000012509418 02 LISTSXP g1c0 [MARK]
##       TAG: @0x0000000011da2630 01 SYMSXP g1c0 [MARK,NAM(2),LCK,gp=0x4000] "names" (has value)
##       @0x00000000124c4240 16 STRSXP g1c2 [MARK] (len=2, tl=0)
##  @0x0000000013ce3378 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "a"
##  @0x000000001407f270 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "b"
##   @0x0000000012508438 19 VECSXP g1c1 [MARK,ATT] (len=1, tl=0)
##     @0x00007ff5e74f0010 14 REALSXP g1c7 [MARK] (len=1000000, tl=0) 1.05788,1.20887,-0.816647,-0.0454
##   ATTRIB:
##     @0x0000000012509728 02 LISTSXP g1c0 [MARK]
##       TAG: @0x0000000011da2630 01 SYMSXP g1c0 [MARK,NAM(2),LCK,gp=0x4000] "names" (has value)
##       @0x0000000012508468 16 STRSXP g1c1 [MARK] (len=1, tl=0)
##  @0x0000000013ce3378 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "a"
```

```
## ATTRIB:
##   @0x0000000012509798 02 LISTSXP g1c0 [MARK]
##     TAG: @0x0000000011da2630 01 SYMSXP g1c0 [MARK,NAM(2),LCK,gp=0x4000] "names" (has value)
##     @0x00000000124c42b0 16 STRSXP g1c2 [MARK] (len=2, tl=0)
##       @0x00000000160b8678 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "A"
##       @0x00000000124f00c0 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "B"
```

```
mem_change(copyofLists$B <- c(copyofLists$B, rnorm(1e6)))
```

```
## 80 MB
```

```
.Internal(inspect(lists))
```

```
## @0x00000000124c4278 19 VECSXP g1c2 [MARK,NAM(2),ATT] (len=2, tl=0)
##   @0x00000000124c4208 19 VECSXP g1c2 [MARK,NAM(2),ATT] (len=2, tl=0)
##     @0x00007ff5ec8f0010 14 REALSXP g1c7 [MARK] (len=10000000, tl=0) 1.47197,0.912267,2.20222,-1.7520
##     @0x00007ff5e7ca0010 14 REALSXP g1c7 [MARK] (len=10000000, tl=0) -0.802089,0.0725343,-0.252489,-0
##   ATTRIB:
##     @0x0000000012509418 02 LISTSXP g1c0 [MARK]
##       TAG: @0x0000000011da2630 01 SYMSXP g1c0 [MARK,NAM(2),LCK,gp=0x4000] "names" (has value)
##       @0x00000000124c4240 16 STRSXP g1c2 [MARK] (len=2, tl=0)
##  @0x0000000013ce3378 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "a"
##  @0x000000001407f270 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "b"
##   @0x0000000012508438 19 VECSXP g1c1 [MARK,NAM(2),ATT] (len=1, tl=0)
##     @0x00007ff5e74f0010 14 REALSXP g1c7 [MARK,NAM(2)] (len=1000000, tl=0) 1.05788,1.20887,-0.816647,
##   ATTRIB:
##     @0x0000000012509728 02 LISTSXP g1c0 [MARK]
##       TAG: @0x0000000011da2630 01 SYMSXP g1c0 [MARK,NAM(2),LCK,gp=0x4000] "names" (has value)
##       @0x0000000012508468 16 STRSXP g1c1 [MARK,NAM(2)] (len=1, tl=0)
##  @0x0000000013ce3378 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "a"
## ATTRIB:
##   @0x0000000012509798 02 LISTSXP g1c0 [MARK]
##     TAG: @0x0000000011da2630 01 SYMSXP g1c0 [MARK,NAM(2),LCK,gp=0x4000] "names" (has value)
##     @0x00000000124c42b0 16 STRSXP g1c2 [MARK,NAM(2)] (len=2, tl=0)
##       @0x00000000160b8678 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "A"
##       @0x00000000124f00c0 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "B"
```

```
.Internal(inspect(copyofLists))
```

```
## @0x00000000124c42e8 19 VECSXP g1c2 [MARK,NAM(1),ATT] (len=2, tl=0)
##   @0x00000000124c4208 19 VECSXP g1c2 [MARK,NAM(2),ATT] (len=2, tl=0)
##     @0x00007ff5ec8f0010 14 REALSXP g1c7 [MARK] (len=10000000, tl=0) 1.47197,0.912267,2.20222,-1.7520
##     @0x00007ff5e7ca0010 14 REALSXP g1c7 [MARK] (len=10000000, tl=0) -0.802089,0.0725343,-0.252489,-0
##   ATTRIB:
##     @0x0000000012509418 02 LISTSXP g1c0 [MARK]
##       TAG: @0x0000000011da2630 01 SYMSXP g1c0 [MARK,NAM(2),LCK,gp=0x4000] "names" (has value)
##       @0x00000000124c4240 16 STRSXP g1c2 [MARK] (len=2, tl=0)
##  @0x0000000013ce3378 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "a"
##  @0x000000001407f270 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "b"
##   @0x00007ff5e6590010 19 VECSXP g1c7 [MARK,NAM(1),ATT] (len=1000001, tl=0)
##     @0x00007ff5e74f0010 14 REALSXP g1c7 [MARK,NAM(2)] (len=1000000, tl=0) 1.05788,1.20887,-0.816647,
##     @0x0000000012507e60 14 REALSXP g1c1 [MARK] (len=1, tl=0) -0.681086
##     @0x0000000012507e90 14 REALSXP g1c1 [MARK] (len=1, tl=0) -2.08421
##     @0x0000000012507ec0 14 REALSXP g1c1 [MARK] (len=1, tl=0) -0.328865
##     @0x0000000012507ef0 14 REALSXP g1c1 [MARK] (len=1, tl=0) -1.04247
##     ...
```

```
##   ATTRIB:
##     @0x00000000125093e0 02 LISTSXP g1c0 [MARK]
##       TAG: @0x0000000011da2630 01 SYMSXP g1c0 [MARK,NAM(2),LCK,gp=0x4000] "names" (has value)
##       @0x00007ff5e5de0010 16 STRSXP g1c7 [MARK] (len=1000001, tl=0)
##  @0x0000000013ce3378 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "a"
##  @0x0000000011da0e90 09 CHARSXP g1c1 [MARK,gp=0x60] [ASCII] [cached] ""
##  @0x0000000011da0e90 09 CHARSXP g1c1 [MARK,gp=0x60] [ASCII] [cached] ""
##  @0x0000000011da0e90 09 CHARSXP g1c1 [MARK,gp=0x60] [ASCII] [cached] ""
##  @0x0000000011da0e90 09 CHARSXP g1c1 [MARK,gp=0x60] [ASCII] [cached] ""
##  ...
## ATTRIB:
##   @0x0000000012509488 02 LISTSXP g1c0 [MARK]
##     TAG: @0x0000000011da2630 01 SYMSXP g1c0 [MARK,NAM(2),LCK,gp=0x4000] "names" (has value)
##     @0x00000000124c42b0 16 STRSXP g1c2 [MARK,NAM(2)] (len=2, tl=0)
##       @0x00000000160b8678 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "A"
##       @0x00000000124f00c0 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "B"
```

0.9 problem 2 (d)

We find that the two vectors in the list point to the memory location. so the real memory used is ~80Mb as can be seen with gc().

object.size():This function merely provides a rough indication: it should be reasonably accurate for atomic vectors, but does not detect if elements of a list are shared, for example.

Therefore, object.size() count the memory of both elements in the list, which is two times of the real size.

```
gc()
```

```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  1494187  79.8    2637877 140.9  1565418  83.7
## Vcells 54901523 418.9   88519968 675.4 55901278 426.5
```

```
tmp <- list()
x <- rnorm(1e7)
tmp[[1]] <- x
tmp[[2]] <- x
.Internal(inspect(tmp))
```

```
## @0x00000000124b6098 19 VECSXP g0c2 [NAM(1)] (len=2, tl=0)
##   @0x00007ff5e1190010 14 REALSXP g0c7 [NAM(2)] (len=10000000, tl=0) -1.3483,0.345942,-0.0897337,1.39
##   @0x00007ff5e1190010 14 REALSXP g0c7 [NAM(2)] (len=10000000, tl=0) -1.3483,0.345942,-0.0897337,1.39
```

```
object.size(tmp)
```

```
## 160000136 bytes
```

```
gc()
```

```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  1494259  79.9    2637877 140.9  1565418  83.7
## Vcells 54901650 418.9   88519968 675.4 64973892 495.8
```

0.10 problem 3 original code

To present the time the original code takes.

```
library(rbenchmark)
load("C:/Users/crypress/Desktop/243/ps4prob3.Rda") # provides values for A, k, n
ll <- function(Theta, A) {
  sum.ind <- which(A==1, arr.ind=T)
  logLik <- sum(log(Theta[sum.ind])) - sum(Theta)
  return(logLik)
}
oneUpdate <- function(A, n, K, theta.old, thresh = 0.1) {
  theta.old1 <- theta.old
  Theta.old <- theta.old %*% t(theta.old)
  L.old <- ll(Theta.old, A)
  q <- array(0, dim = c(n, n, K))
  for (i in 1:n) {
    for (j in 1:n) {
      for (z in 1:K) {
        if (theta.old[i, z]*theta.old[j, z] == 0){
          q[i, j, z] <- 0
        } else {
          q[i, j, z] <- theta.old[i, z]*theta.old[j, z] /
            Theta.old[i, j]
        }
      }
    }
  }
  theta.new <- theta.old
  for (z in 1:K) {
    theta.new[,z] <- rowSums(A*q[,,z])/sqrt(sum(A*q[,,z]))
  }
  Theta.new <- theta.new %*% t(theta.new)
  L.new <- ll(Theta.new, A)
  converge.check <- abs(L.new - L.old) < thresh
  theta.new <- theta.new/rowSums(theta.new)
  return(list(theta = theta.new, loglik = L.new,
              converged = converge.check))
}
# initialize the parameters at random starting values
temp <- matrix(runif(n*K), n, K)
theta.init <- temp/rowSums(temp)
# do single update and check the time
print(system.time(out <- oneUpdate(A, n, K, theta.init)))
```

```
##    user  system elapsed
##    7.46    0.16    7.61
```

0.11 problem 3 improved code

To present the time the improved code takes.

```
load("C:/Users/crypress/Desktop/243/ps4prob3.Rda") # provides values for A, k, n
# change the name of the function from "ll" to "getlogLik" to show the application of this function
getlogLik <- function(Theta, A) {
  sum.ind <- which(A==1, arr.ind=T)
  logLik <- sum(log(Theta[sum.ind])) - sum(Theta)
```

```
  return(logLik)
}
oneUpdate <- function(A, n, K, theta.old, thresh = 0.1) {
  # theta.old1 is removed as it is not used in the funtion
  Theta.old <- theta.old %*% t(theta.old)
  L.old <- getlogLik(Theta.old, A)
  # instead of 3 dimentional array, a  2-D matrix is used to for q
  q <- matrix(0, nrow = n,ncol = n)
  theta.new <- theta.old
  # instead of using a for loop, apply() is applied here
  theta.new<- apply(matrix(1:K), 1, FUN = function(z) {
    q <- theta.old[,z]%*% t(theta.old[,z])/Theta.old
    theta.new[,z] <- rowSums(A*q)/sqrt(sum(A*q))
  })
  Theta.new <- theta.new %*% t(theta.new)
  L.new <- getlogLik(Theta.new, A)
  converge.check <- abs(L.new - L.old) < thresh
  theta.new <- theta.new/rowSums(theta.new)
  return(list(theta = theta.new, loglik = L.new,
              converged = converge.check))
}
# initialize the parameters at random starting values
temp <- matrix(runif(n*K), n, K)
theta.init <- temp/rowSums(temp)
# do single update and check the time
out <- oneUpdate(A, n, K, theta.init)
print(system.time(out <- oneUpdate(A, n, K, theta.init)))
```

```
##    user  system elapsed
##    0.21    0.28    0.50
```

0.12 problem 4 (a) PIKK algorithm

```
# PIKK algorithm
library(microbenchmark)
```

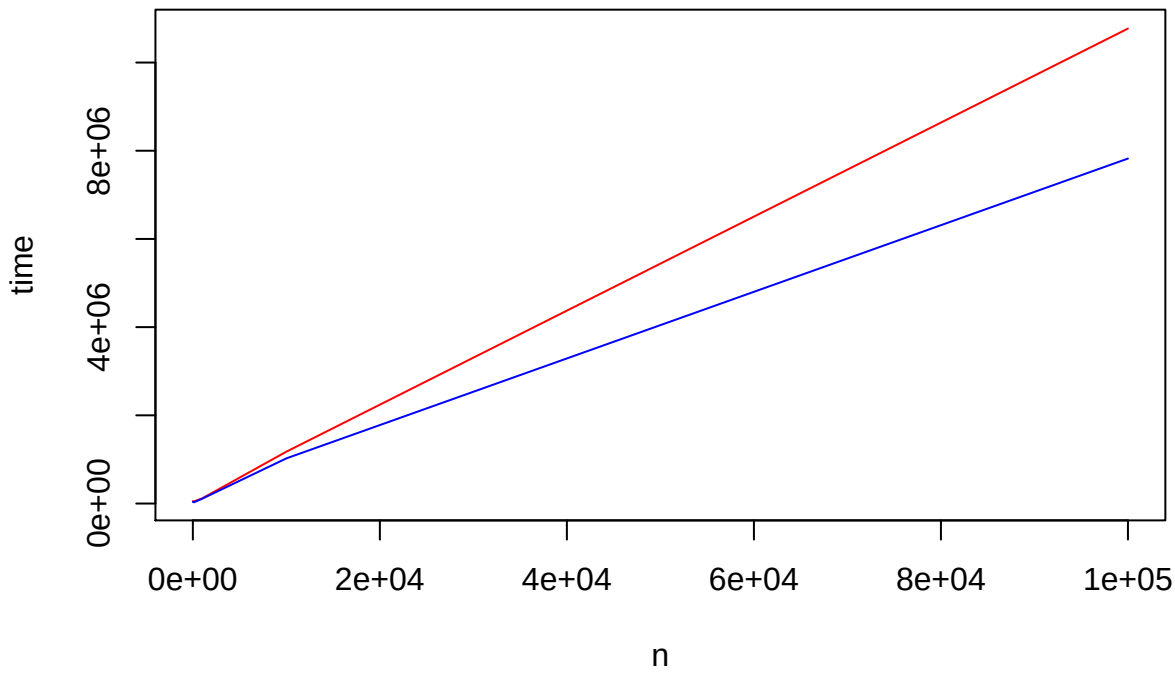
```
## Warning: package 'microbenchmark' was built under R version 3.4.2
```

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

# improved PIKK algorithm
newPIKK <- function(x, k) {
  order(runif(length(x)))[1:k]
}

# test and compare
orgTime <- list()
newTime <- list()
n <- c(0,1e2,1e3,1e4, 1e5)
for (i in n){
```

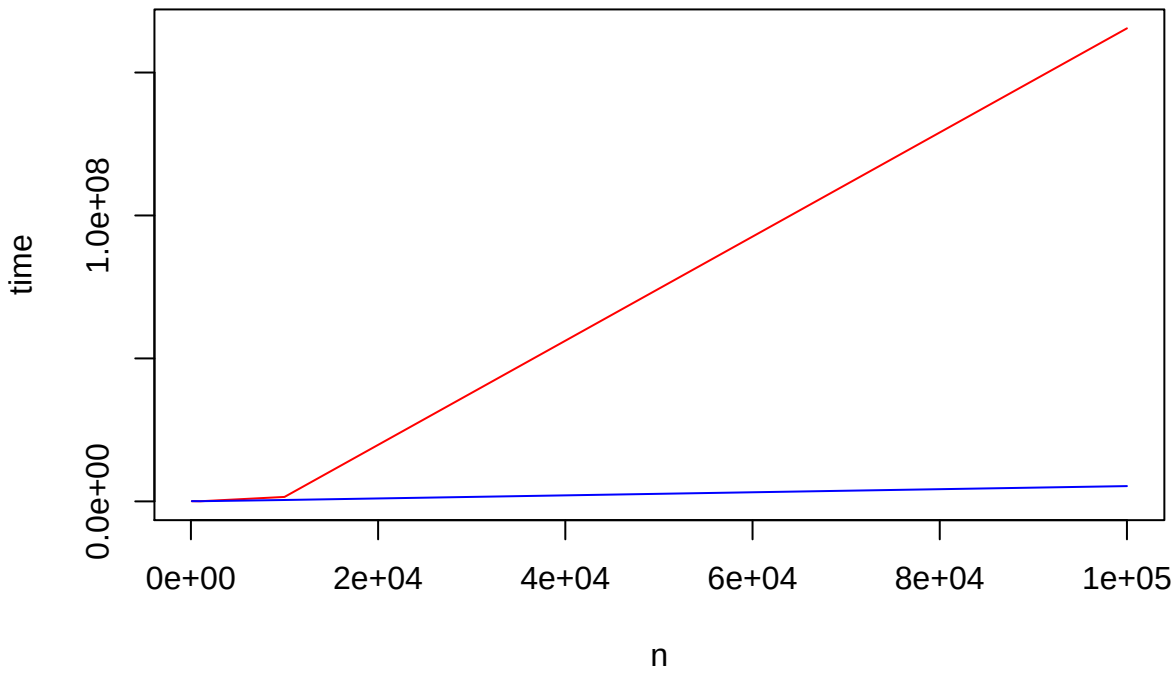
```
  x <- c(1:i)
  k <- i/100
  orgBenchm <- microbenchmark(PIKK(x,k))
  newBenchm <- microbenchmark(newPIKK(x,k))
  orgTime <- c(orgTime, mean(orgBenchm$time))
  newTime<- c(newTime, mean(newBenchm$time))
}
plot(n,orgTime, type = "l", col = "red",
     xlab = "n", ylab ="time")
lines(n, newTime, col = "blue")
```



0.13 Problem 4 (b) PYKD algorithm

```
# original FYKD algorithm
FYKD <- function(x, k) {
  n <- length(x)
  for(i in 1:k) {
    j = sample(i:n, 1)
    tmp <- x[i]
    x[i] <- x[j]
    x[j] <- tmp
  }
  return(x[1:k])
}
```

```r
# Improved FYKD algorithm
newFYKD <- function(x, k) {
  n <- length(x)
  y <- c(1:k)
  idx <- sapply(y, function(x){
    sample(i:n,1)})
  for(i in 1:k) {
    tmp <- x[i]
    x[i] <- x[idx[i]]
    x[idx[i]] <- tmp
  }
  return(x)
}
# test and compare
orgTime <- list()
newTime <- list()
n <- c(1e2,1e3,1e4, 1e5)
for (i in n){
  x <- c(1:i)
  k <- i/100
  orgBenchm <- microbenchmark(FYKD(x,k))
  newBenchm <- microbenchmark(newFYKD(x,k))
  orgTime <- c(orgTime, mean(orgBenchm$time))
  newTime<- c(newTime, mean(newBenchm$time))
}
plot(n,orgTime, type = "l", col = "red",
     xlab = "n", ylab ="time")
lines(n, newTime, col = "blue")
```


time
0.0e+00
1.0e+08
0e+00
2e+04
4e+04
6e+04
8e+04
1e+05
n