Homerwork 6th benchmarks

This documents shows benchmarks of functions created for 6th homework of IPI PAN Advanced R course.

Exercise 06.01 - mode

Mode is the most frequently occurring value in an integer vector

- mode - Rcpp function
- mode2 - function written in R for reference

```
x<-as.integer(runif(100)*100)
microbenchmark(mode(x), mode2(x))
```

```
## Unit: microseconds
##      expr     min      lq      mean   median      uq     max neval
##   mode(x)  12.571  13.968  16.33727  17.0410  17.879  37.714   100
##  mode2(x) 124.596 127.670 135.17366 129.2065 138.984 292.495   100
```

```
x<-as.integer(runif(10000)*100)
microbenchmark(mode(x), mode2(x))
```

```
## Unit: microseconds
##      expr      min       lq      mean   median       uq      max neval
##   mode(x)  533.588  542.946  606.6553  561.803  600.775 2882.768   100
##  mode2(x) 2172.063 2219.695 2353.0309 2271.936 2422.794 4854.527   100
```

```
x<-as.integer(runif(100000)*10000)
microbenchmark(mode(x), mode2(x))
```

```
## Unit: milliseconds
##      expr      min       lq      mean    median       uq      max neval
##   mode(x)  9.00282  9.27115  9.567319  9.468801  9.81242 11.47045   100
##  mode2(x) 29.84402 30.70670 32.679161 31.523978 32.94148 99.16456   100
```

As we can see, Rcp is faster than R, but difference is smaller and smaller when data size grows.

Exercise 06.02 - simplify2array2

This function converts list into matrix if possible.

- simplify2array2 - Rcpp function
- simplify2array - function from R base

```
L<-list()
for (i in 1:10) L[[i]]<-as.integer(runif(10)*100)
microbenchmark(simplify2array2(L), simplify2array(L))
```

```
## Unit: microseconds
##                expr    min      lq     mean median      uq     max neval
##  simplify2array2(L) 15.644 16.6220 18.44368 17.600 19.6955  46.095   100
##   simplify2array(L) 67.048 68.5845 75.72197 71.378 73.7525 188.571   100
```

```
L<-list()
for (i in 1:100) L[[i]]<-as.integer(runif(100)*100)
microbenchmark(simplify2array2(L), simplify2array(L))
```

```
## Unit: microseconds
##                expr     min       lq     mean   median       uq      max
##  simplify2array2(L) 246.120 259.2510 297.1550 262.4635 268.8885 3190.908
##   simplify2array(L) 253.664 260.6475 296.9875 268.4700 289.7020  906.819
##  neval
##    100
##    100
```

```
L<-list()
for (i in 1:1000) L[[i]]<-as.integer(runif(1000)*100)
microbenchmark(simplify2array2(L), simplify2array(L))
```

```
## Unit: milliseconds
##                expr      min       lq     mean   median       uq      max
##  simplify2array2(L) 26.20696 29.04406 36.53416 30.27983 32.33092 94.88580
##   simplify2array(L) 10.49798 10.89384 15.72433 12.55062 13.31468 75.38779
##  neval
##    100
##    100
```

For this function, Rcpp i faster only for small lists. For lists with about 10000 elements times are similar, for more elements base R function is faster. This happens because Rcpp solution is just iterating through vectors from beginning to end while R is optimized to operations on whole vector at the same time.

Exercise 06.03 - ass

This function generates all possible assignments of survey participants.

- ass - Rcpp function
- ass2 - function written in R for reference

```r
microbenchmark(ass(1), ass2(1))
```

```
## Unit: microseconds
##     expr     min       lq      mean   median       uq     max neval
##   ass(1)   9.499  11.1750  14.59748  15.0870  15.9245  64.813   100
##  ass2(1) 222.934 227.6835 243.00362 238.9975 244.7250 457.880   100
```

```r
microbenchmark(ass(2), ass2(2))
```

```
## Unit: microseconds
##     expr     min       lq      mean  median       uq      max neval
##   ass(2)   9.498  10.8950  15.70838  12.571  18.5775   64.533   100
##  ass2(2) 813.232 833.4855 859.69276 839.492 851.0855 1201.549   100
```

```r
microbenchmark(ass(3), ass2(3))
```

```
## Unit: microseconds
##     expr       min         lq        mean    median         uq       max
##   ass(3)    10.896    12.8505    27.91692    37.994    41.2065    46.375
##  ass2(3) 18688.688 18980.7650 19762.53125 19223.113 20269.3360 23342.352
##  neval
##    100
##    100
```

As we can see, Rcpp function is O(log(n)), while R function is O(n). This happens because this Rcpp function uses STL algorithm which is highly optimized, while R uses own permutation function written in R.