Exceedingly Simple Sorting with Indices

Jan de Leeuw

Version 01, March 22, 2017

Abstract

We use the system `qsort` to write a routine that produces both the sort an the order of a vector of doubles.

Contents

1 Introduction **1**

2 Example **1**

3 C code **2**

4 R code **3**

Note: This is a working paper which will be expanded/updated frequently. All suggestions for improvement are welcome. The directory gifi.stat.ucla.edu/mysort has a pdf version, the complete Rmd file with the code chunks, and the R source code.

1 Introduction

This may not be of much use to anyone else, but I'll make it available anyway. For a larger project I needed something simple in C to sort a vector and return both the sorted vector and the corresponding index vector. Thus it combines `sort` and `order` in R. It creates an array of structures in C, where each structure has a value member and an index member. It then sorts the structures using the system `qsort`, by comparison of the values. Finally it splits the sorted array of structures into values and indices and returns those to R in a two-element list.

2 Example

```
set.seed (12345)
x <- rnorm (10)
mySort (x)
```

```
## $values
##  [1] -1.8179559677 -0.9193220025 -0.4534971735 -0.2841597439 -0.2761841052
##  [6] -0.1093033147  0.5855288178  0.6058874558  0.6300985511  0.7094660175
##
## $indices
##  [1]  6 10  4  9  8  3  1  5  7  2
```

as compared to

```
list (values = sort (x), indices = order (x))
```

```
## $values
##  [1] -1.8179559677 -0.9193220025 -0.4534971735 -0.2841597439 -0.2761841052
##  [6] -0.1093033147  0.5855288178  0.6058874558  0.6300985511  0.7094660175
##
## $indices
##  [1]  6 10  4  9  8  3  1  5  7  2
```

A quick comparison shows very little timing difference.

```
f <- function () {
  x <- rnorm(10000)
  h <- mySort (x)
}
g <- function () {
  x <- rnorm (10000)
  h <- list (values = sort (x), indices = order (x))
}
microbenchmark (f, g, times = 1000)
```

```
## Unit: nanoseconds
##  expr min lq   mean median uq  max neval cld
##     f  32 33 34.889     34 36  416  1000   a
##     g  30 31 34.769     32 33 2690  1000   a
```

3 C code

```
#include <math.h>
#include <stdlib.h>

int myComp(const void *, const void *);
void mySort(double *, int *, int *);

struct couple {
    double value;
```

```
    int index;
};

int myComp(const void *px, const void *py) {
    double x = ((struct couple *)px)->value;
    double y = ((struct couple *)py)->value;
    return (int)copysign(1.0, x - y);
}

void mySort(double *x, int *k, int *n) {
    int nn = *n;
    struct couple *xi =
        (struct couple *)calloc((size_t)nn, (size_t)sizeof(struct couple));
    for (int i = 0; i < nn; i++) {
        xi[i].value = x[i];
        xi[i].index = i + 1;
    }
    (void)qsort(xi, (size_t)nn, (size_t)sizeof(struct couple), myComp);
    for (int i = 0; i < nn; i++) {
        x[i] = xi[i].value;
        k[i] = xi[i].index;
    }
    free(xi);
}
```

4 R code

```
dyn.load("mySort.so")

mySort <- function (x) {
  h <-
    .C(
      "mySort",
      values = as.double (x),
      indices = as.integer(1:length(x)),
      as.integer(length(x))
    )
  return (list (values = h$values, indices = h$indices))
}
```