Performance of the bit package

Dr. Jens Oehlschlägel

2019-12-10

Contents

A performance example

Before we measure performance of the main functionality of the package, note that something simple as '(a:b)[-i]' can and has been accelerated in this package:

```
a <- 1L
b <- 1e7L
i <- sample(a:b,1e3)
x <- c(
  R = median(microbenchmark((a:b)[-i], times=times)$time)
, bit = median(microbenchmark(bit_rangediff(c(a,b), i), times=times)$time)
, merge = median(microbenchmark(merge_rangediff(c(a,b), bit_sort(i)), times=times)$time)
)
knitr::kable(as.data.frame(as.list(x/x["R"]*100)), caption="% of time relative to R", digits=1)
```

Table 1: % of time relative to R

R	bit	merge
100	11.7	9.1

Boolean data types

> "A designer knows he has achieved perfection not when there is nothing left to add, but when there is nothing left to take away."
> "Il semble que la perfection soit atteinte non quand il n'y a plus rien à ajouter, mais quand il n'y a plus rien à retrancher"
> (Antoine de St. Exupery, Terre des Hommes (Gallimard, 1939), p. 60.)

We compare memory consumption (n=1000) and runtime (median of 5 replications) of the different `booltypes` for the following filter scenarios:

Table 2: selection characteristic

coin	often	rare	chunk
random 50%	random 99%	random 1%	contiguous chunk of 5%

There are substantial savings in skewed filter situations:

Even in non-skewed situations the new booltypes are competitive:

Detailed tables follow.

% memory consumption of filter

Table 3: % bytes of logical

	coin	often	rare	chunk
logical	100.0	100.0	100.0	100.0
bit	40.5	40.5	40.5	40.5
bitwhich	62.3	18.2	18.2	22.1
which	57.9	110.3	13.8	17.8
ri	NA	NA	NA	8.9

% time extracting

Table 4: % bytes of logical

	coin	often	rare	chunk
logical	0.8	2.7	1.0	NA
bit	48.5	14.9	13.6	NA
bitwhich	33.9	31.0	27.7	NA
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

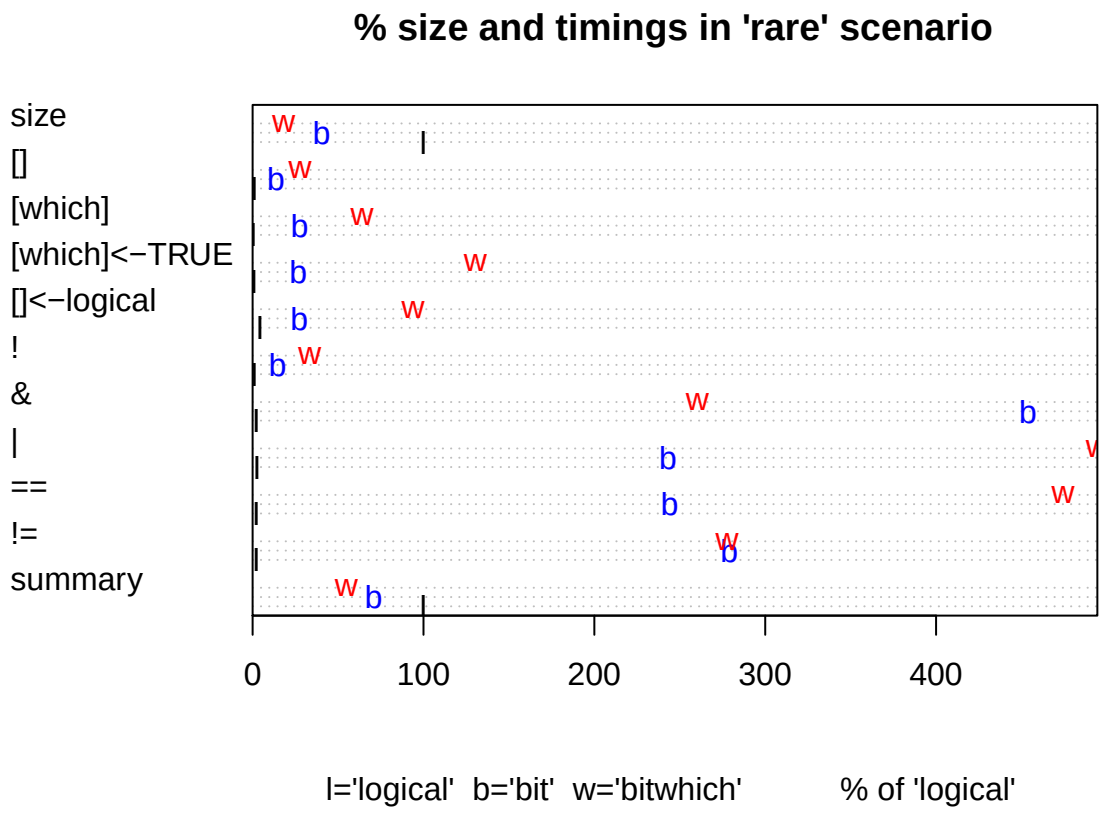

Figure 1: % size and execution time for bit (b) and bitwhich (w) relative to logical in the 'rare' scenario

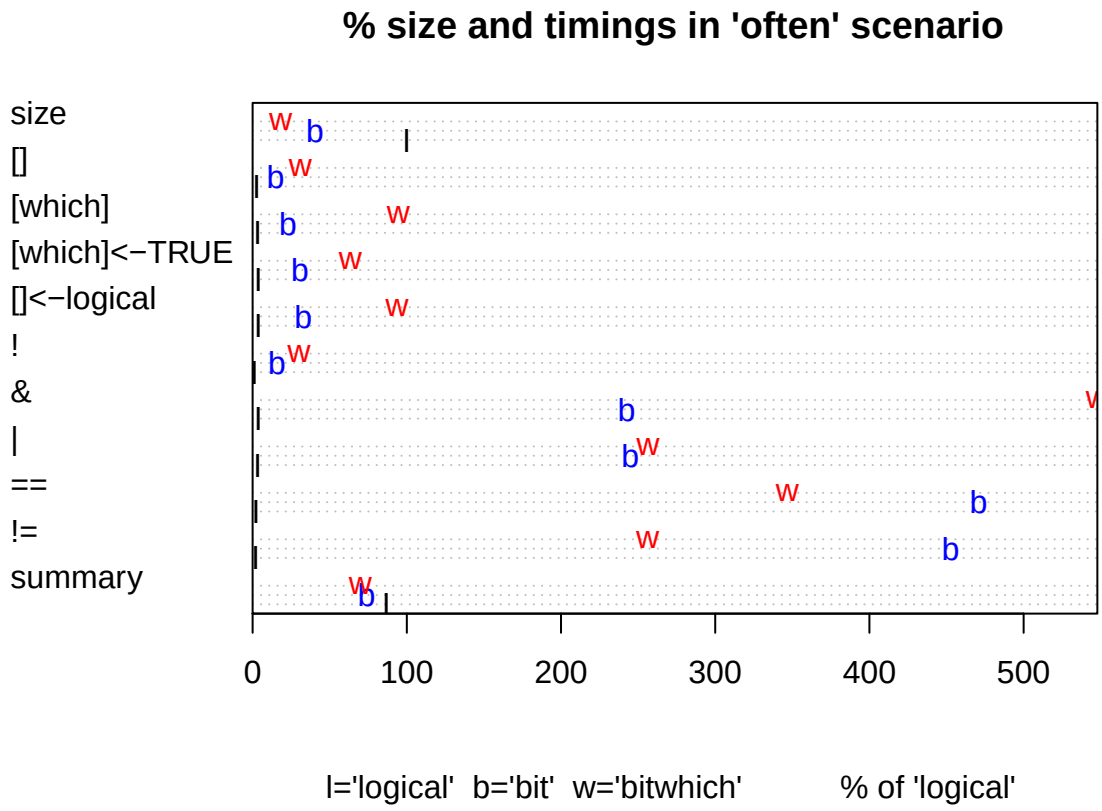

Figure 2: % size and execution time for bit (b) and bitwhich (w) relative to logical in the 'coin' scenario

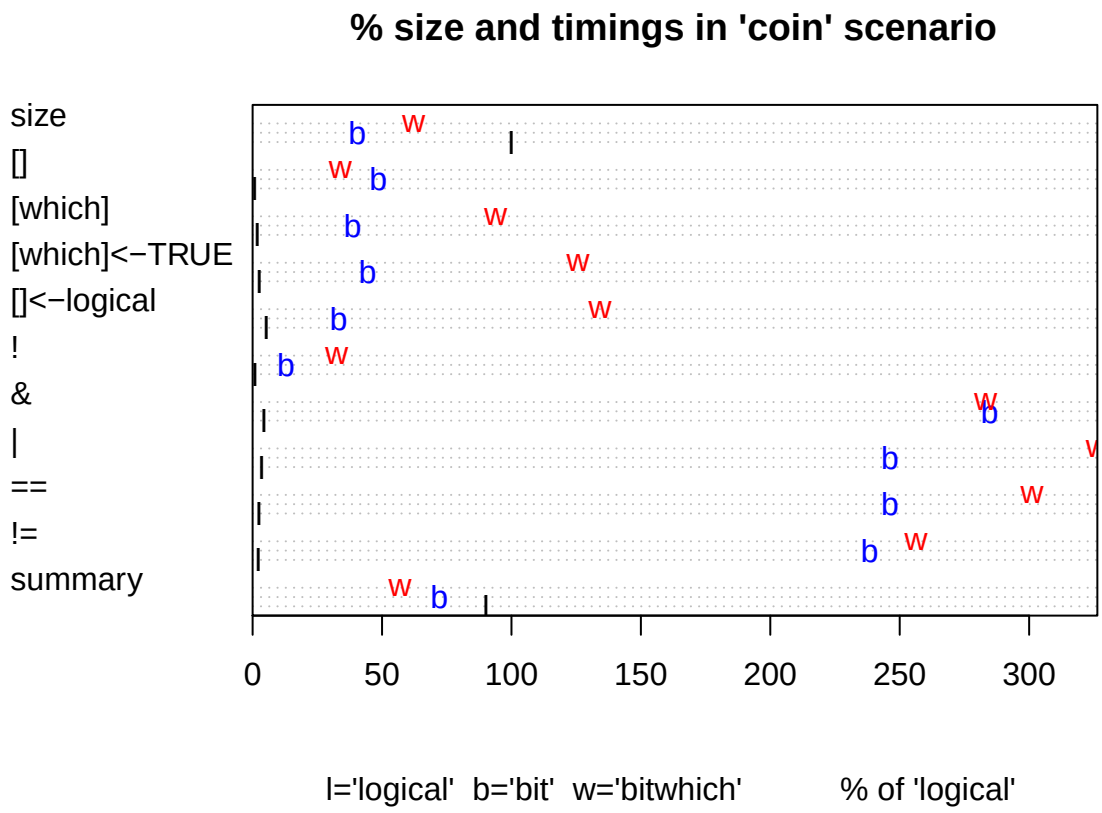

Figure 3: % size and execution time for bit (b) and bitwhich (w) relative to logical in the 'coin' scenario

% time assigning

Table 5: % bytes of logical

	coin	often	rare	chunk
logical	5.3	3.7	4.4	NA
bit	33.3	32.9	27.3	NA
bitwhich	134.3	93.8	93.9	NA
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time subscripting with 'which'

Table 6: % bytes of logical

	coin	often	rare	chunk
logical	1.9	3.3	0.4	NA
bit	38.6	23.0	27.6	NA
bitwhich	93.9	94.5	64.1	NA
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time assigning with 'which'

Table 7: % bytes of logical

	coin	often	rare	chunk
logical	2.6	3.8	0.9	NA
bit	44.4	30.7	26.8	NA
bitwhich	125.7	63.3	130.5	NA
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean NOT

Table 8: % time for Boolean NOT

	coin	often	rare	chunk
logical	1.0	1.2	1.0	1.2
bit	12.9	15.8	14.8	14.7
bitwhich	32.5	30.2	33.4	78.3
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean AND

Table 9: % time for Boolean &

	coin	often	rare	chunk
logical	4.5	3.7	2.1	3.1
bit	284.8	242.6	453.8	242.3
bitwhich	283.3	547.8	260.4	699.5
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean OR

Table 10: % time for Boolean |

	coin	often	rare	chunk
logical	3.5	3.3	2.7	2.3
bit	246.2	244.9	243.1	498.8
bitwhich	326.4	256.4	494.4	376.2
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean EQUALITY

Table 11: % time for Boolean ==

	coin	often	rare	chunk
logical	2.5	2.2	2.2	2.1
bit	246.2	470.8	244.2	259.6
bitwhich	301.2	346.8	474.4	330.2
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean XOR

Table 12: % time for Boolean !=

	coin	often	rare	chunk
logical	2.2	2.0	2.2	2.0
bit	238.5	452.6	279.0	276.8
bitwhich	256.4	256.3	277.6	292.1
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean SUMMARY

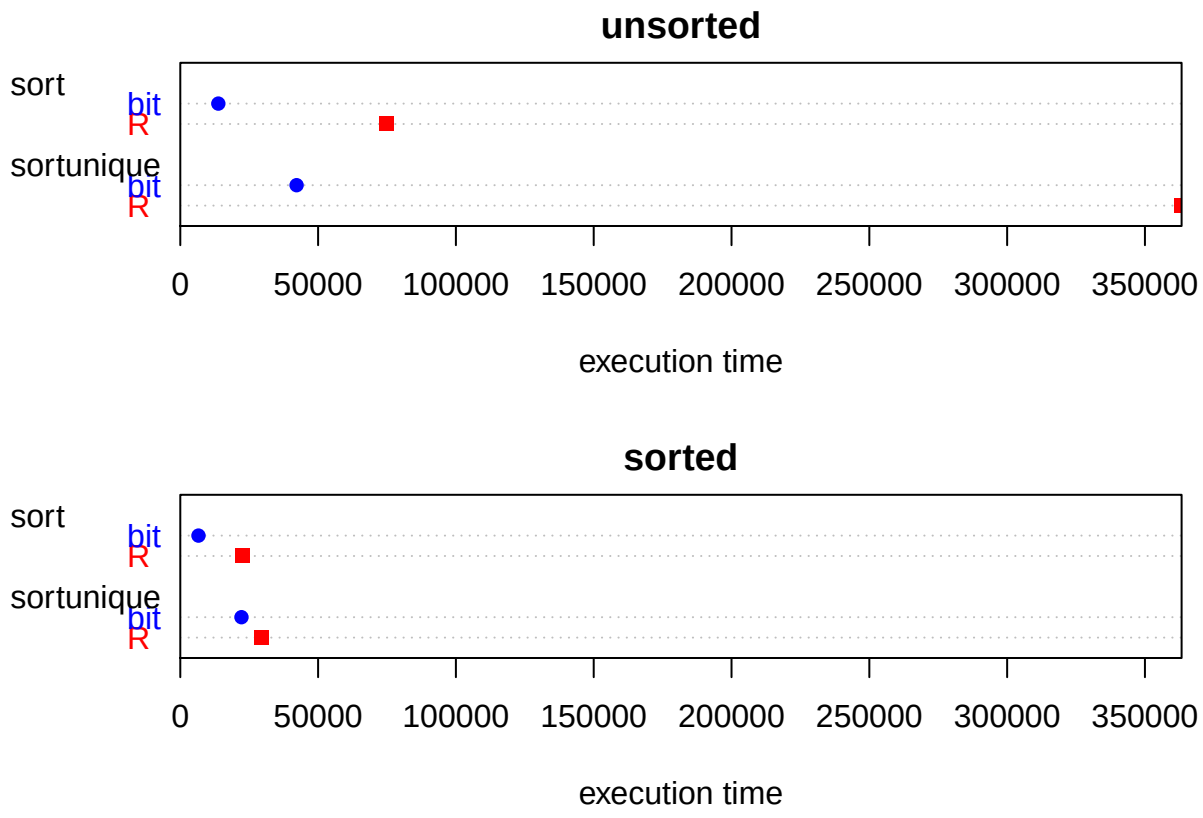

Figure 4: Execution time for R (R) and bit (b)

Table 13: % time for Boolean summary

	coin	often
logical	90.2	86.7
bit	72.1	74.1

Fast methods for `integer` set operations

> "The space-efficient structure of bitmaps dramatically reduced the run time of sorting"
> (Jon Bently, Programming Pearls, Cracking the oyster, p. 7)

% time for sorting

Table 14: sorted data relative to R's sort

	small	big
sort	45.4	29.3
sortunique	115.7	75.5

Table 15: unsorted data relative to R's sort

	small	big

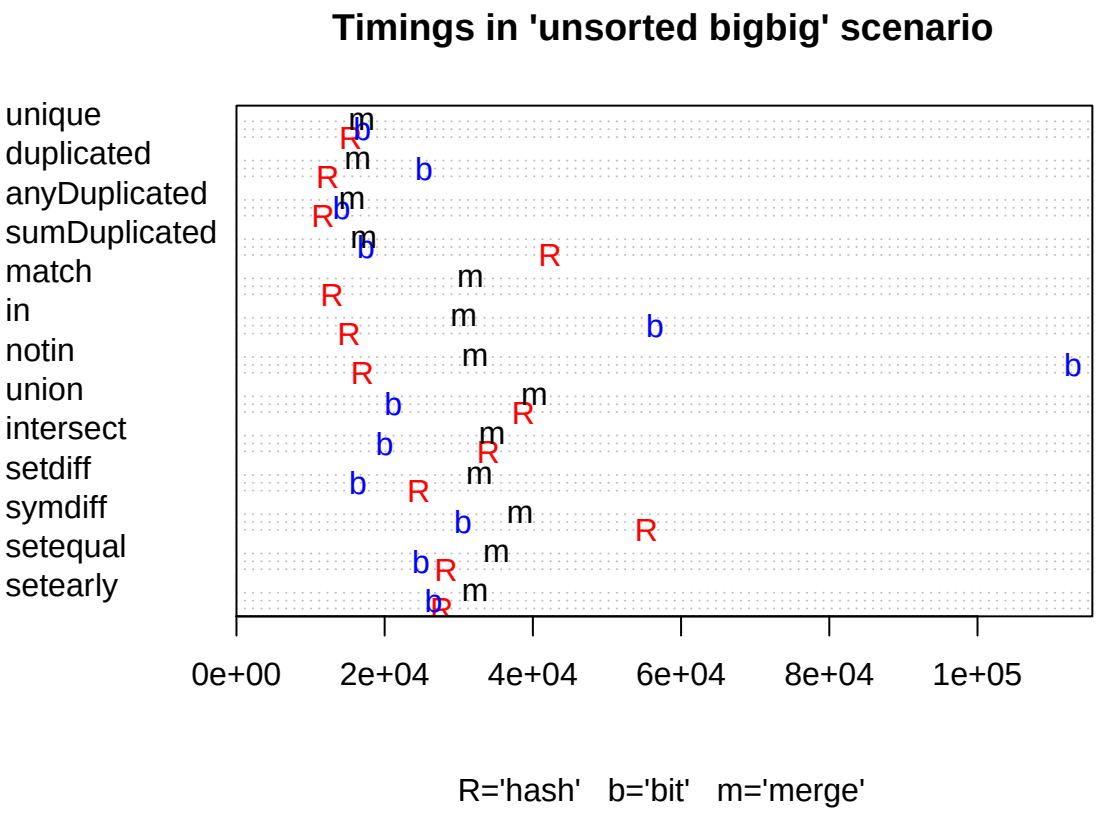

Figure 5: Execution time for R, bit and merge relative to most expensive R in 'unsorted bigbig' scenario

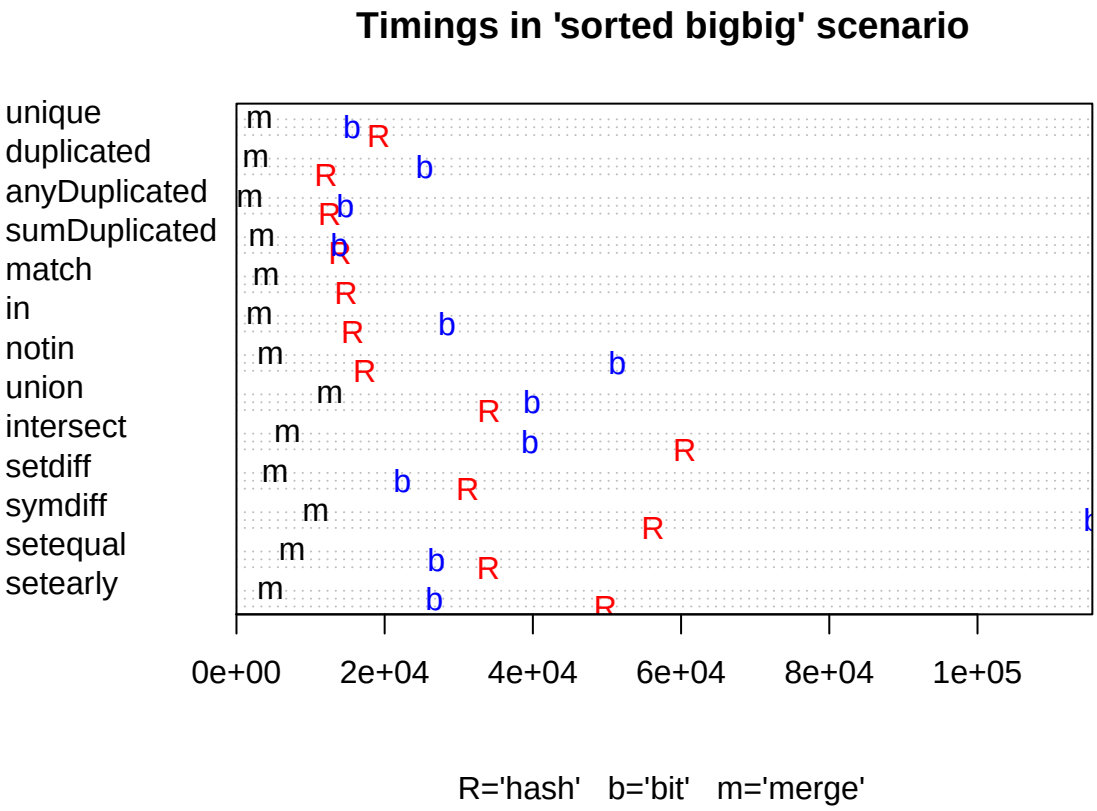

Figure 6: Execution time for R, bit and merge in 'sorted bigbig' scenario

Table 18: sorted data relative to R

	small	big
bit	485.7	209.9
merge	30.5	21.5
sort	0.0	0.0

Table 19: unsorted data relative to R

	small	big
bit	838.2	205.7
merge	270.9	133.3
sort	212.7	112.2

% time for anyDuplicated

Table 20: sorted data relative to R

	small	big
bit	218.0	116.7
merge	26.2	14.3
sort	0.0	0.0

Table 21: unsorted data relative to R

	small	big
bit	997.9	121.4
merge	277.1	133.3
sort	243.8	117.9

% time for sumDuplicated

Table 22: sorted data relative to R

	small	big
bit	152.1	99.3
merge	20.5	24.3
sort	0.0	0.0

Table 23: unsorted data relative to R

	small	big
bit	122.9	41.4
merge	159.0	40.7
sort	141.0	32.6

% time for match

Table 24: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	NA	NA	NA	NA
merge	96.2	19.8	51.5	27
sort	0.0	0.0	0.0	0

Table 25: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	NA	NA	NA	NA
merge	784.8	165.1	194.3	245
sort	709.1	153.6	161.4	214

% time for in

Table 26: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	611.3	254.2	215.9	180.9
merge	54.7	20.8	19.0	19.7
sort	0.0	0.0	0.0	0.0

Table 27: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	1565.5	264.9	153.4	371.7
merge	906.9	283.5	136.8	202.0
sort	806.9	262.9	125.0	181.6

% time for notin

Table 28: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	995.2	388.8	327.5	297.1
merge	50.0	31.8	19.7	26.6
sort	0.0	0.0	0.0	0.0

Table 29: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	538.1	435.1	263.0	664.1

	smallsmall	smallbig	bigsmall	bigbig
merge	303.6	253.5	171.5	189.4
sort	278.6	223.7	154.5	162.4

% time for union

Table 30: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	136.2	59.7	28.7	117
merge	63.1	29.0	21.7	37
sort	0.0	0.0	0.0	0

Table 31: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	127.3	115.9	118.4	54.8
merge	298.2	158.4	189.9	103.9
sort	212.7	112.8	117.5	71.3

% time for intersect

Table 32: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	109.2	91.0	44.0	65.5
merge	20.0	14.6	8.1	11.4
sort	0.0	0.0	0.0	0.0

Table 33: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	131.1	77.0	41.8	58.8
merge	118.7	126.6	65.8	101.5
sort	106.8	114.9	58.2	81.2

% time for setdiff

Table 34: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	229.7	69.7	54.9	71.8
merge	20.7	12.8	9.7	16.7
sort	0.0	0.0	0.0	0.0

Table 35: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	117.4	65.2	43.8	66.7
merge	235.8	138.8	81.9	133.3
sort	214.7	126.9	72.0	112.2

% time for symdiff

Table 36: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	65.7	39.0	81.1	205.5
merge	7.5	4.2	4.6	19.0
sort	0.0	0.0	0.0	0.0

Table 37: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	70.3	80.5	44.4	55.3
merge	88.6	41.5	41.5	69.3
sort	80.7	37.1	36.9	49.9

% time for setequal

Table 38: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	392.5	524.1	111.1	79.4
merge	52.2	32.9	15.4	22.1
sort	0.0	0.0	0.0	0.0

Table 39: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	472.1	293.2	96.9	88.0
merge	312.8	384.9	92.5	124.0
sort	272.1	349.3	79.2	97.5

% time for setearly

Table 40: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	185.0	26.3	38.0	53.6

	smallsmall	smallbig	bigsmall	bigbig
merge	33.3	14.6	18.2	9.2
sort	0.0	0.0	0.0	0.0

Table 41: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	184.6	28.8	30.0	96.0
merge	201.5	134.0	175.0	116.2
sort	172.1	120.3	159.4	99.6