Performance of the bit package

Dr. Jens Oehlschlägel

2022-11-13

Contents

A performance example . . . 1
Boolean data types . . . 2
% memory consumption of filter . . . 6
% time extracting . . . 6
% time assigning . . . 6
% time subscripting with 'which' . . . 6
% time assigning with 'which' . . . 6
% time Boolean NOT . . . 7
% time Boolean AND . . . 7
% time Boolean OR . . . 7
% time Boolean EQUALITY . . . 7
% time Boolean XOR . . . 8
% time Boolean SUMMARY . . . 8
Fast methods for `integer` set operations . . . 8
% time for sorting . . . 11
% time for unique . . . 11
% time for duplicated . . . 11
% time for anyDuplicated . . . 12
% time for sumDuplicated . . . 12
% time for match . . . 12
% time for in . . . 13
% time for notin . . . 13
% time for union . . . 13
% time for intersect . . . 14
% time for setdiff . . . 14
% time for symdiff . . . 14
% time for setequal . . . 15
% time for setearly . . . 15

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	19.4	21.5

The vignette is compiled with the following performance settings: 5 replications with domain size small 1000 and big 10^6, sample size small 1000 and big 10^6.

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=1e+06) and runtime (median of 5 replications) of the different `booltype`s 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.

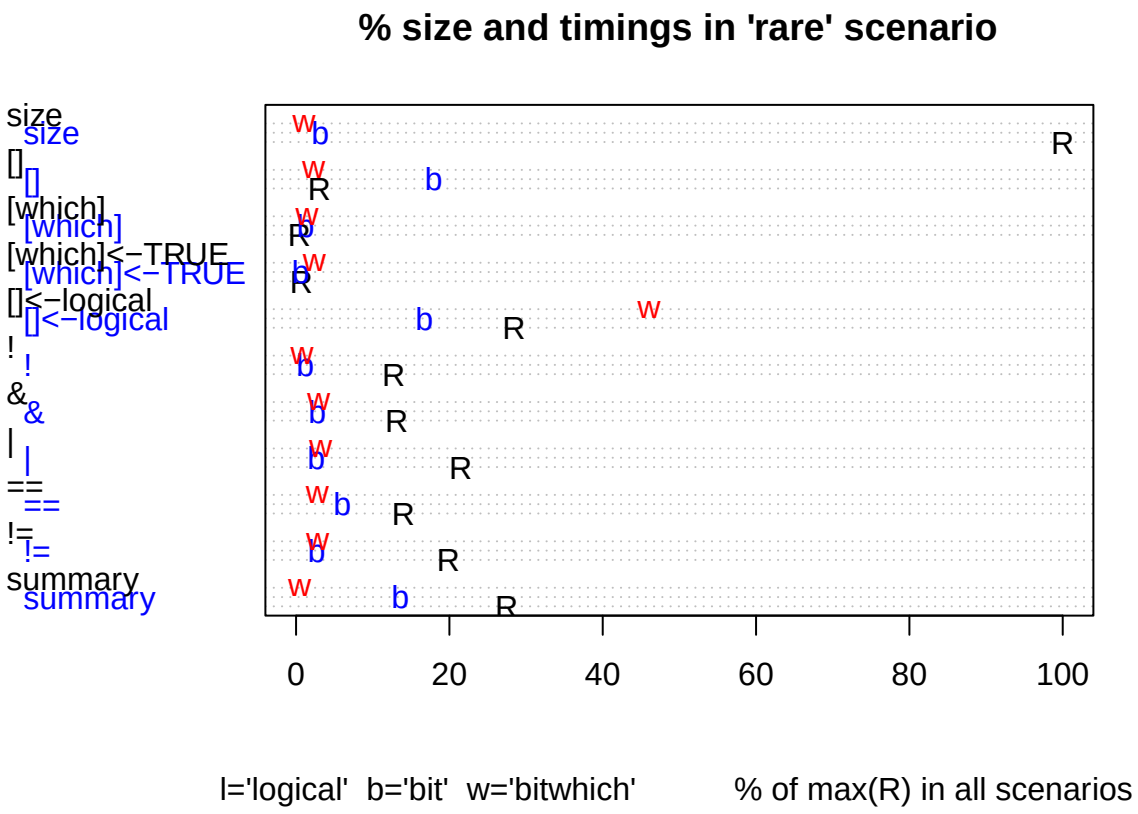

Figure 1: % size and execution time for bit (b) and bitwhich (w) relative to logical (R) in the 'rare' scenario

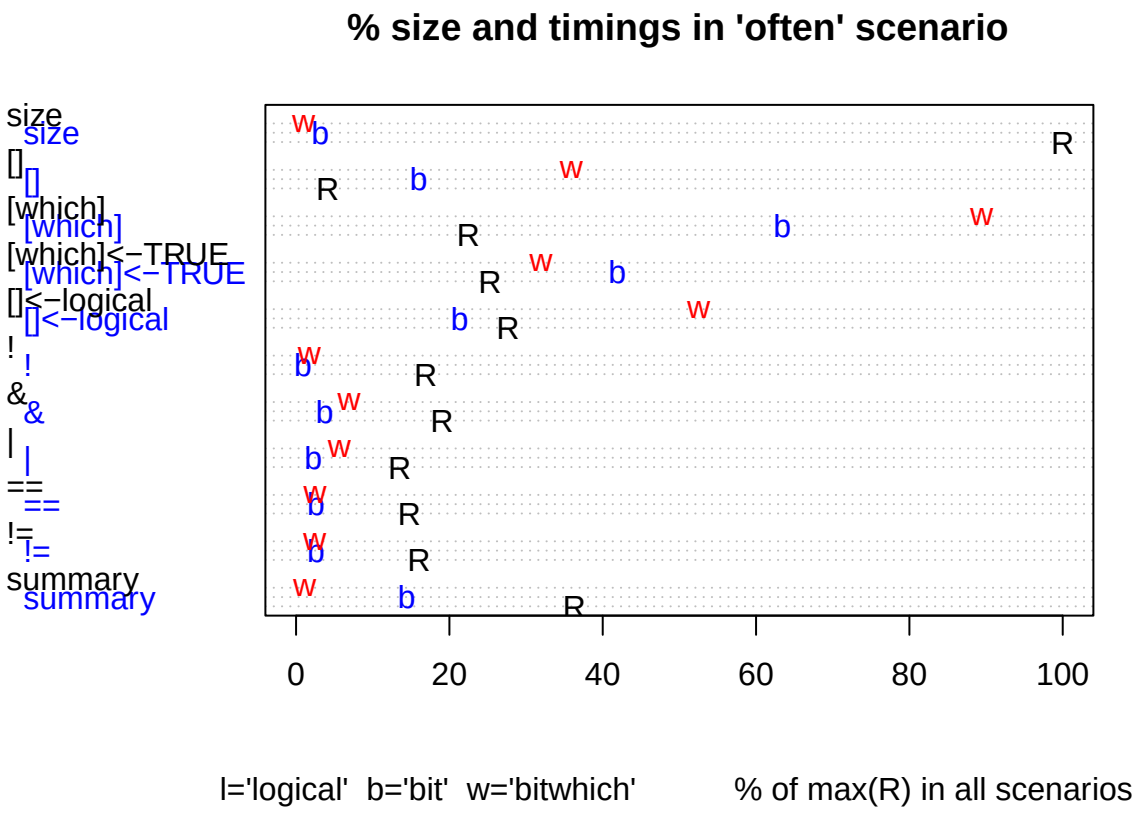

Figure 2: % size and execution time for bit (b) and bitwhich (w) relative to logical (R) in the 'often' scenario

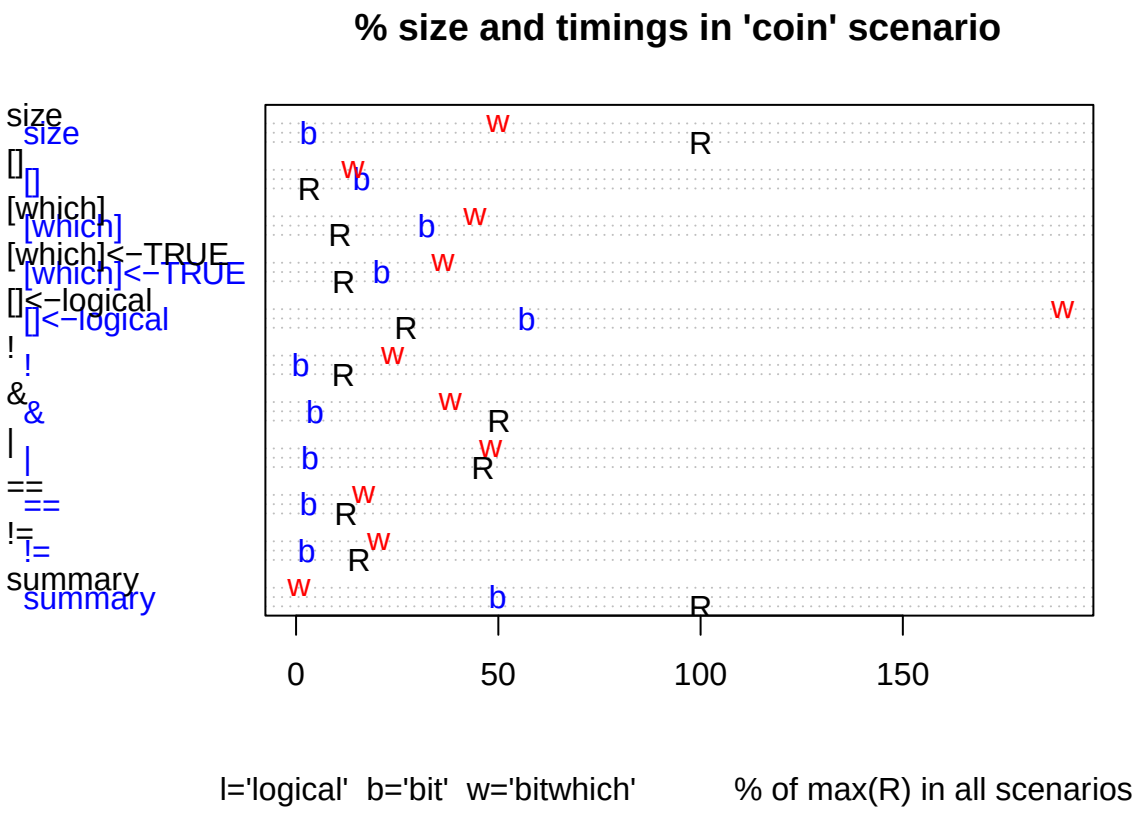

Figure 3: % size and execution time for bit (b) and bitwhich (w) relative to logical (R) in the 'coin' scenario

% memory consumption of filter

Table 3: % bytes of logical

	coin	often	rare	chunk
logical	100.0	100.0	100.0	100.0
bit	3.2	3.2	3.2	3.2
bitwhich	50.0	1.0	1.0	5.0
which	50.0	99.0	1.0	5.0
ri	NA	NA	NA	0.0

% time extracting

Table 4: % time of logical

	coin	often	rare	chunk
logical	3.2	4.1	3.1	NA
bit	16.2	16.0	17.9	NA
bitwhich	14.1	35.9	2.3	NA
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time assigning

Table 5: % time of logical

	coin	often	rare	chunk
logical	27.2	27.7	28.4	NA
bit	57.0	21.4	16.7	NA
bitwhich	189.5	52.5	46.0	NA
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time subscripting with 'which'

Table 6: % time of logical

	coin	often	rare	chunk
logical	10.9	22.5	0.4	NA
bit	32.3	63.5	1.3	NA
bitwhich	44.3	89.5	1.4	NA
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time assigning with 'which'

Table 7: % time of logical

	coin	often	rare	chunk
logical	11.8	25.3	0.7	NA
bit	21.2	41.9	0.6	NA
bitwhich	36.3	31.9	2.4	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	11.7	16.9	12.7	14.6
bit	1.1	0.9	1.2	1.0
bitwhich	23.9	1.7	0.8	2.8
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean AND

Table 9: % time for Boolean &

	coin	often	rare	chunk
logical	50.1	19.0	13.1	15.9
bit	4.7	3.7	2.8	4.8
bitwhich	38.1	6.9	3.0	5.6
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean OR

Table 10: % time for Boolean |

	coin	often	rare	chunk
logical	46.2	13.5	21.5	24.7
bit	3.4	2.3	2.6	2.5
bitwhich	48.1	5.7	3.2	6.3
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean EQUALITY

Table 11: % time for Boolean ==

	coin	often	rare	chunk
logical	12.4	14.8	14.0	12.9
bit	3.2	2.6	6.1	2.9

	coin	often	rare	chunk
bitwhich	16.7	2.5	2.8	6.2
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean XOR

Table 12: % time for Boolean !=

	coin	often	rare	chunk
logical	15.6	16.0	19.9	16.4
bit	2.7	2.6	2.7	3.4
bitwhich	20.4	2.4	2.8	7.8
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean SUMMARY

Table 13: % time for Boolean summary

	coin	often
logical	100.0	36.3
bit	49.9	14.4

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)

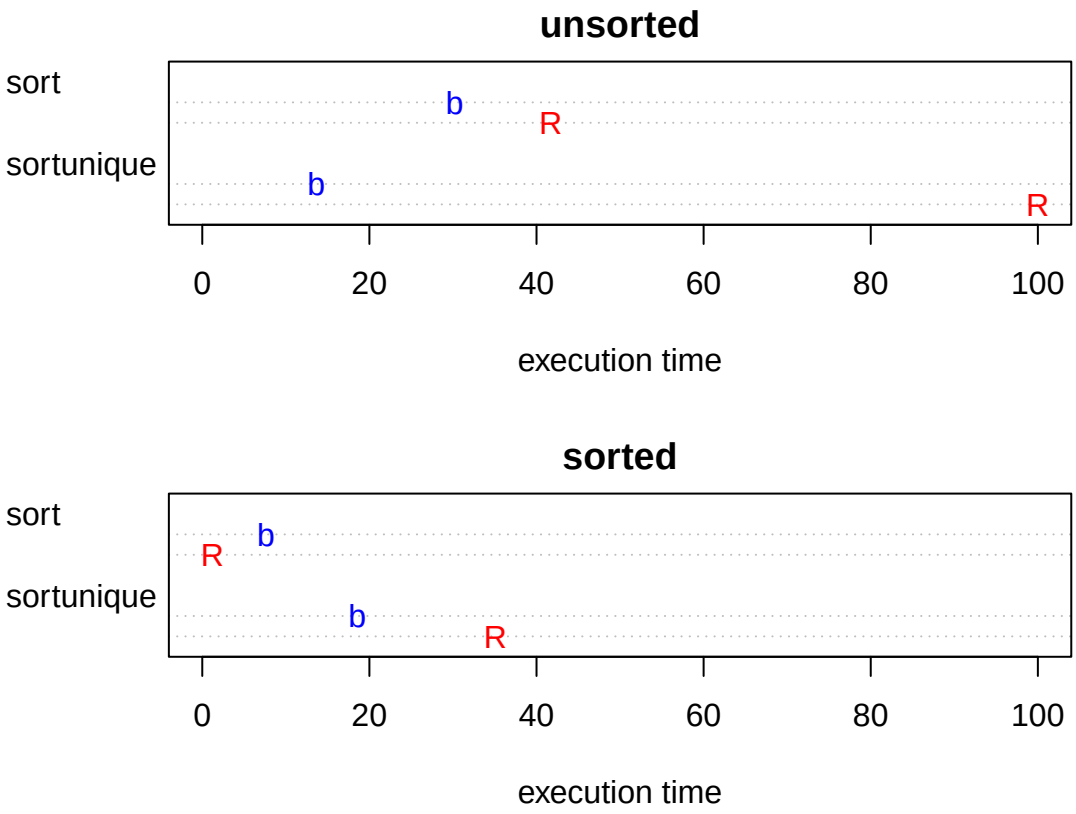

Figure 4: Execution time for R (R) and bit (b)

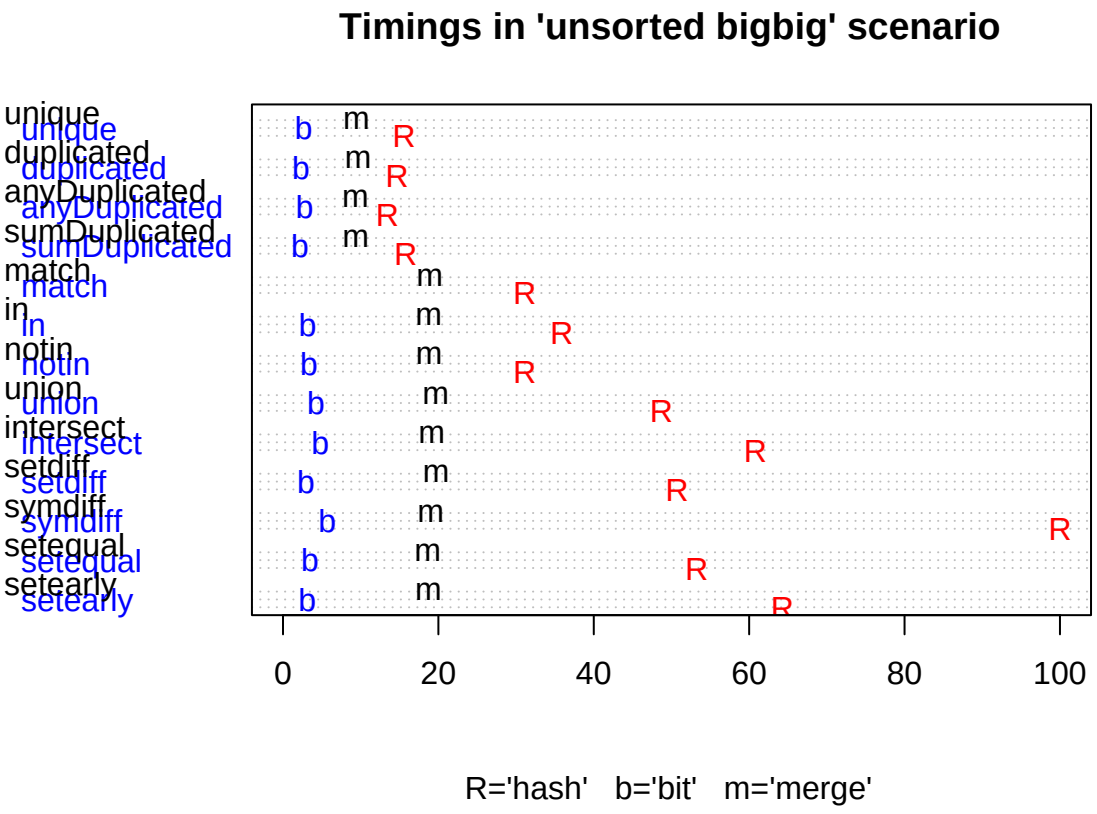

Figure 5: Execution time for R, bit and merge relative to most expensive R in 'unsorted bigbig' scenario

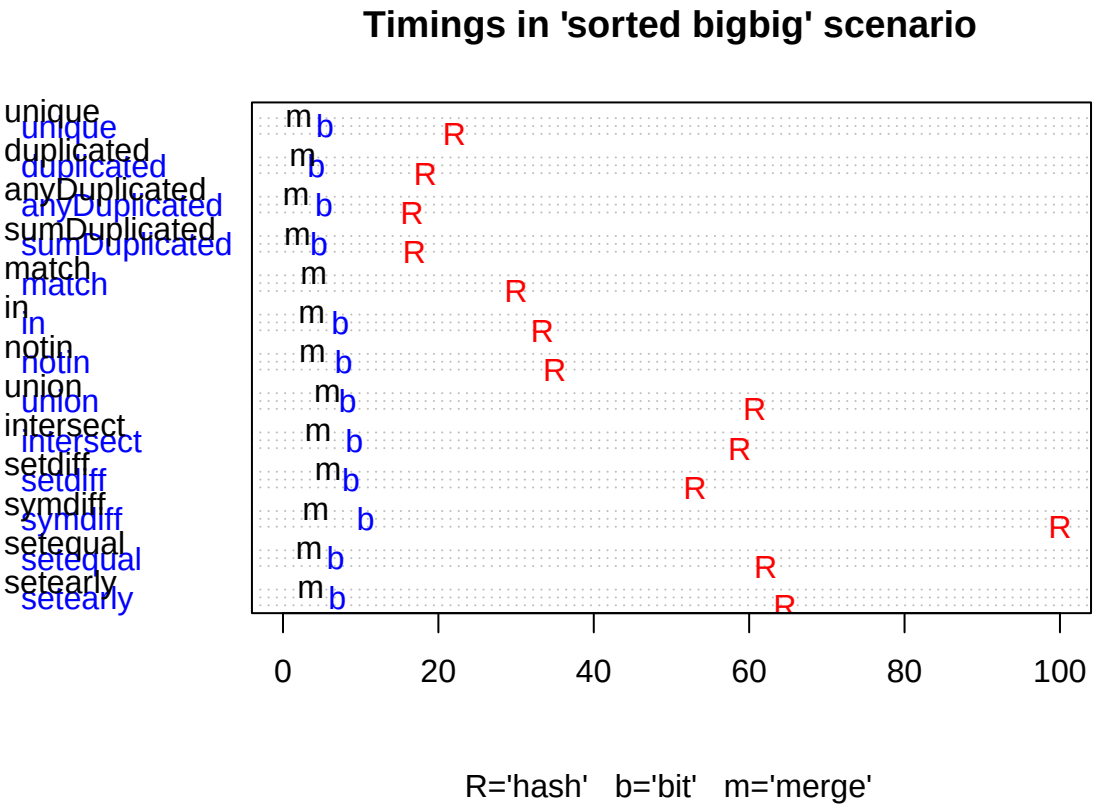

Figure 6: Execution time for R, bit and merge in 'sorted bigbig' scenario

% time for sorting

Table 14: sorted data relative to R's sort

	small	big
sort	171.3	637.1
sortunique	100.9	52.9

Table 15: unsorted data relative to R's sort

	small	big
sort	25.1	72.4
sortunique	20.5	13.7

% time for unique

Table 16: sorted data relative to R

	small	big
bit	160.5	24.5
merge	34.6	9.2
sort	0.0	0.0

Table 17: unsorted data relative to R

	small	big
bit	170.7	17.2

% time for anyDuplicated

Table 20: sorted data relative to R

	small	big
bit	170.1	31.7
merge	29.9	10.2
sort	0.0	0.0

Table 21: unsorted data relative to R

	small	big
bit	247.3	21.0
merge	274.1	69.4
sort	241.6	63.7

% time for sumDuplicated

Table 22: sorted data relative to R

	small	big
bit	153.5	27.5
merge	35.0	11.0
sort	0.0	0.0

Table 23: unsorted data relative to R

	small	big
bit	140.0	13.8
merge	218.4	59.5
sort	187.2	54.2

% time for match

Table 24: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	NA	NA	NA	NA
merge	25.6	0	13.9	13.2
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	434.8	58.2	121.0	60.7

	smallsmall	smallbig	bigsmall	bigbig
sort	379.4	58.2	113.5	55.0

% time for in

Table 26: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	167.9	7	13.4	22.1
merge	21.5	0	12.4	11.2
sort	0.0	0	0.0	0.0

Table 27: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	107.2	3.5	7.5	8.8
merge	182.5	48.2	104.8	52.4
sort	161.2	48.2	97.7	47.7

% time for notin

Table 28: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	282.5	8.1	12.2	22.4
merge	39.0	0.0	10.9	10.8
sort	0.0	0.0	0.0	0.0

Table 29: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	413.8	5.1	7.3	10.7
merge	277.6	53.6	93.5	60.5
sort	245.1	53.6	86.9	55.0

% time for union

Table 30: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	100.3	22.0	25.4	13.7
merge	61.0	8.8	9.9	9.4
sort	0.0	0.0	0.0	0.0

Table 31: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	35.9	10.3	15.6	8.7
merge	86.1	39.8	53.0	40.4
sort	64.6	35.3	47.6	35.1

% time for intersect

Table 32: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	61.1	13.4	7	15.6
merge	11.7	0.0	0	7.7
sort	0.0	0.0	0	0.0

Table 33: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	73.6	6.0	4.2	7.9
merge	143.2	50.9	34.6	31.5
sort	121.0	50.9	34.6	28.1

% time for setdiff

Table 34: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	72.1	6.5	14.7	16.5
merge	30.0	0.1	5.8	11.0
sort	0.0	0.0	0.0	0.0

Table 35: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	66.3	3.0	8.9	5.8
merge	181.8	57.7	31.2	38.9
sort	152.5	57.7	28.0	33.7

% time for symdiff

Table 36: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	51.8	10.8	12.5	10.6
merge	12.4	3.7	3.8	4.2
sort	0.0	0.0	0.0	0.0

Table 37: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	41.7	6.5	5.7	5.7
merge	66.1	16.6	16.2	19.0
sort	56.1	14.7	14.4	17.1

% time for setequal

Table 38: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	101.1	100.6	10.3	10.9
merge	26.4	23.6	5.2	5.4
sort	0.0	0.0	0.0	0.0

Table 39: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	200.4	106.6	5.8	6.5
merge	182.9	91388.9	18.7	35.0
sort	157.6	91365.3	15.9	32.1

% time for setearly

Table 40: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	93.9	5.8	16.2	10.8
merge	13.4	0.0	0.1	5.5
sort	0.0	0.0	0.0	0.0

Table 41: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	133.9	2.9	8.1	4.9
merge	123.2	40.9	125.1	29.1
sort	106.6	40.9	125.1	26.6