Performance of the bit package

Dr. Jens Oehlschlägel

2020-06-14

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	17.7	17.7

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 `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.

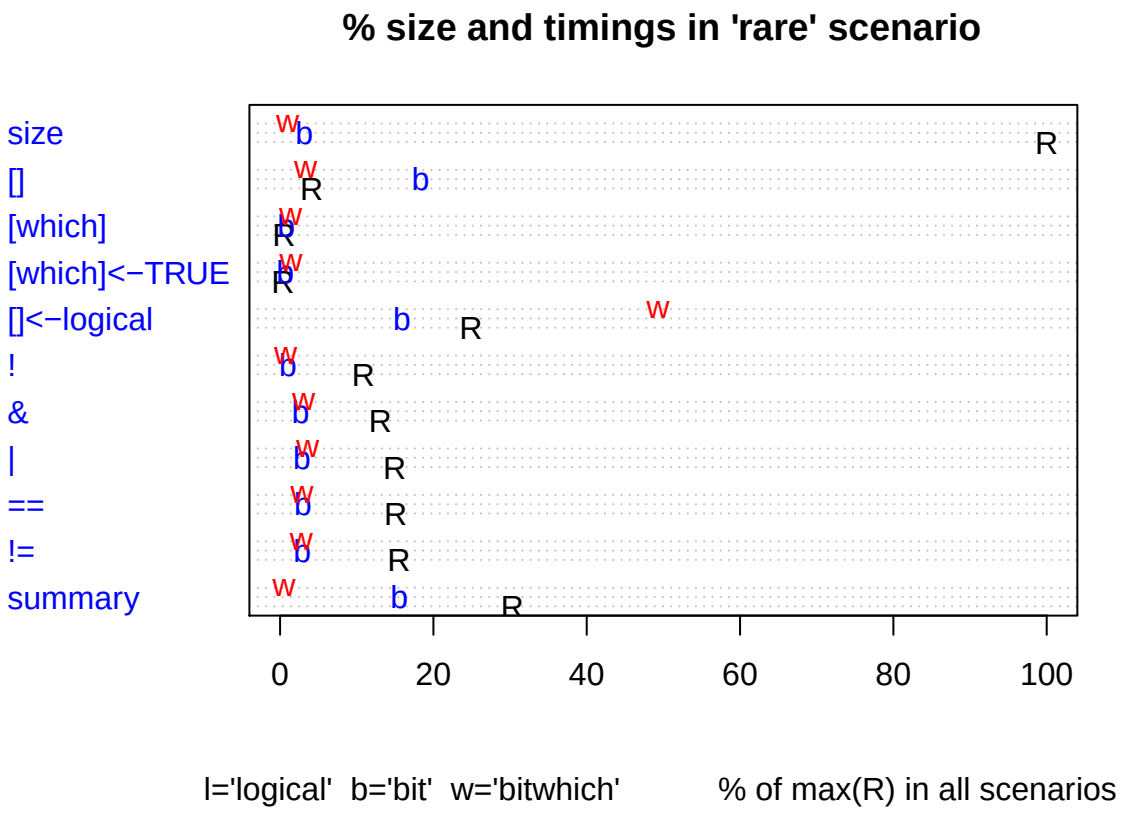

Figure 1: % size and execution time for bit (b) and bitwhich (w) relative to logical (R) in the 'rare' scenario

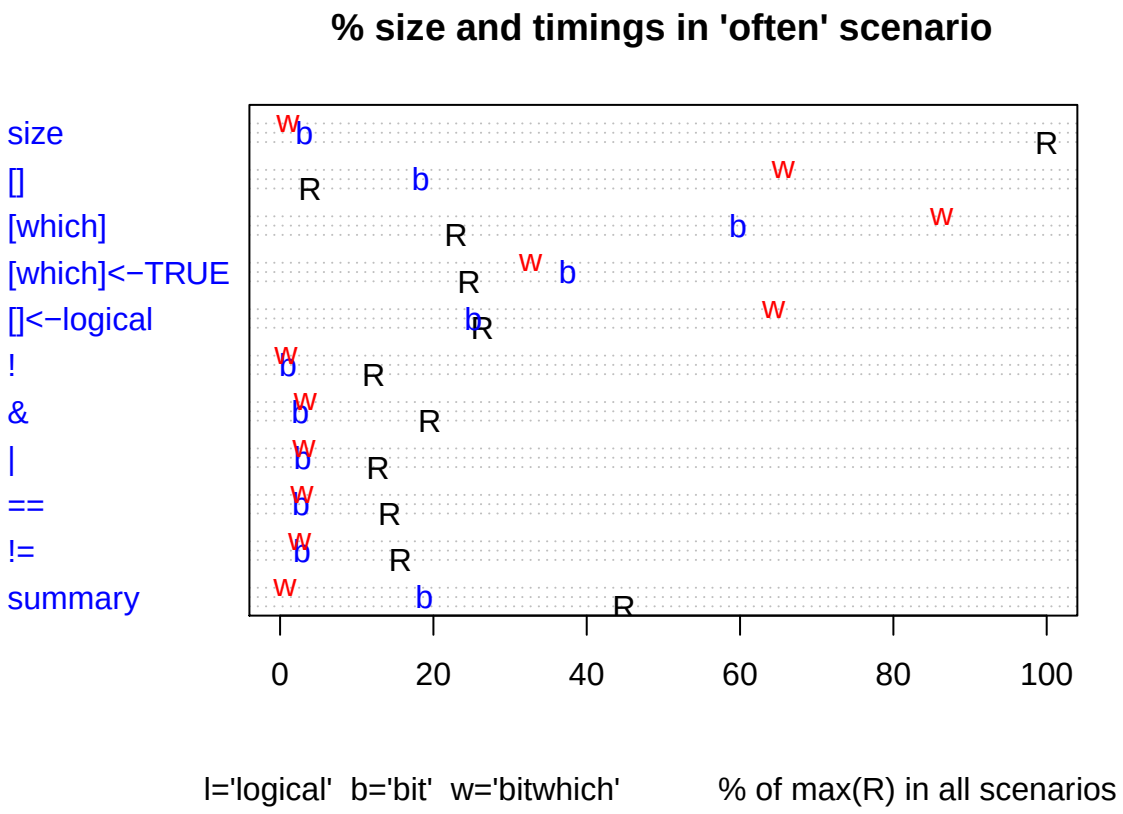

Figure 2: % size and execution time for bit (b) and bitwhich (w) relative to logical (R) in the 'often' scenario

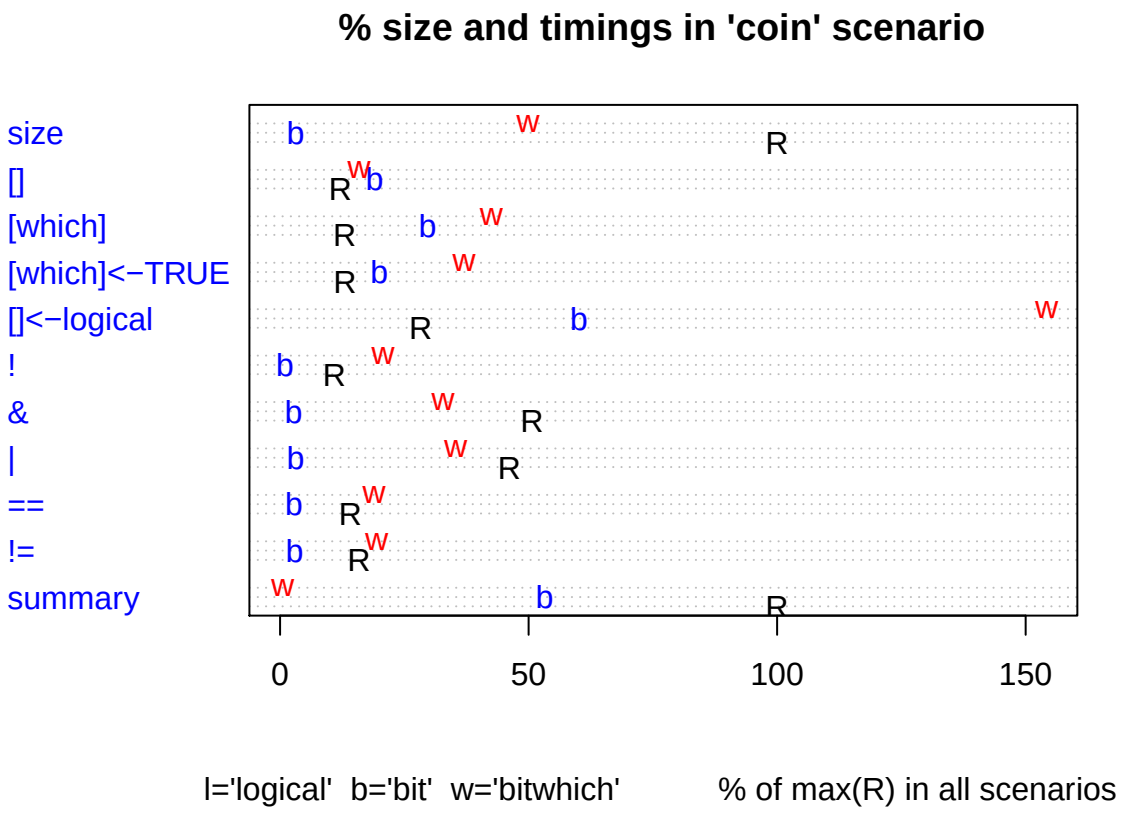

Figure 3: % size and execution time for bit (b) and bitwhich (w) relative to logical (R) in the 'coin' scenario

% 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	49.9	1.0	1.0	5.0
which	49.9	99.0	1.0	5.0
ri	NA	NA	NA	0.0

% time extracting

Table 4: % time of logical

	coin	often	rare	chunk
logical	12.1	3.9	4.1	NA
bit	19.0	18.3	18.3	NA
bitwhich	15.9	65.7	3.3	NA
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time assigning

Table 5: % time of logical

	coin	often	rare	chunk
logical	28.3	26.4	24.9	NA
bit	60.1	25.2	15.9	NA
bitwhich	154.2	64.4	49.3	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	13.0	23.0	0.5	NA
bit	29.7	59.7	0.8	NA
bitwhich	42.5	86.3	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	13.1	24.6	0.4	NA
bit	20.0	37.5	0.7	NA
bitwhich	37.0	32.7	1.4	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	10.9	12.2	10.8	13.5
bit	1.0	1.0	1.0	1.0
bitwhich	20.7	0.8	0.8	2.5
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean AND

Table 9: % time for Boolean &

	coin	often	rare	chunk
logical	50.7	19.5	13.1	12.0
bit	2.7	2.7	2.7	2.6
bitwhich	32.8	3.3	3.1	5.9
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	12.8	15.0	14.6
bit	3.1	3.0	2.9	3.1
bitwhich	35.3	3.1	3.6	6.0
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean EQUALITY

Table 11: % time for Boolean ==

	coin	often	rare	chunk
logical	14.1	14.3	15.1	14.5
bit	2.8	2.7	3.0	2.6

	coin	often	rare	chunk
bitwhich	18.9	2.8	2.8	4.4
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean XOR

Table 12: % time for Boolean !=

	coin	often	rare	chunk
logical	15.9	15.7	15.5	15.5
bit	2.9	2.8	2.9	3.0
bitwhich	19.4	2.6	2.8	4.2
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean SUMMARY

Table 13: % time for Boolean summary

	coin	often
logical	100.0	44.9
bit	53.2	18.8

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)

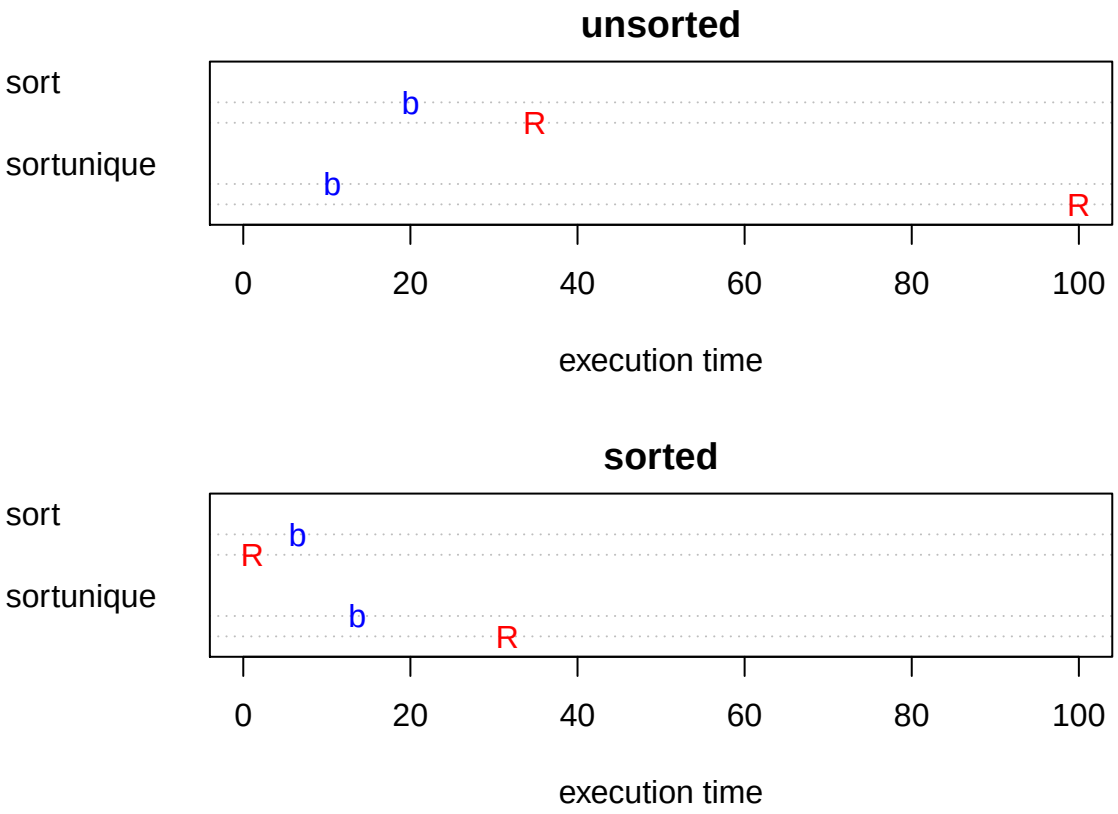

Figure 4: Execution time for R (R) and bit (b)

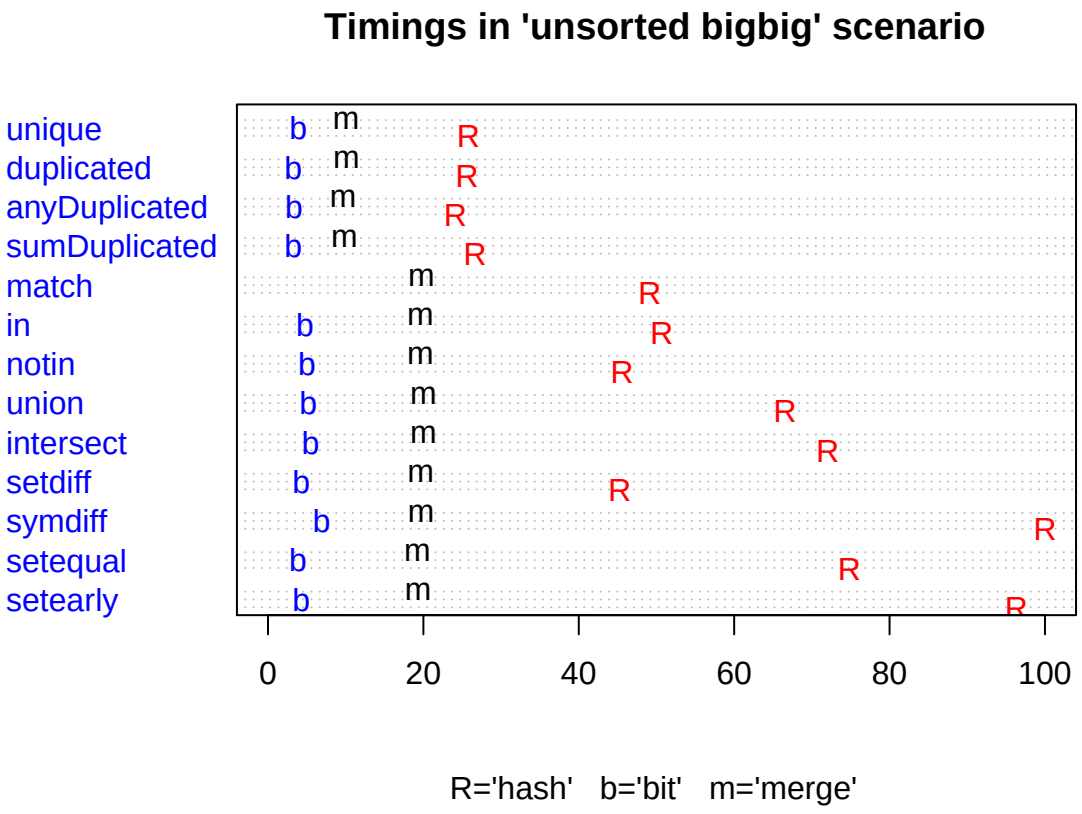

Figure 5: Execution time for R, bit and merge relative to most expensive R in 'unsorted bigbig' scenario

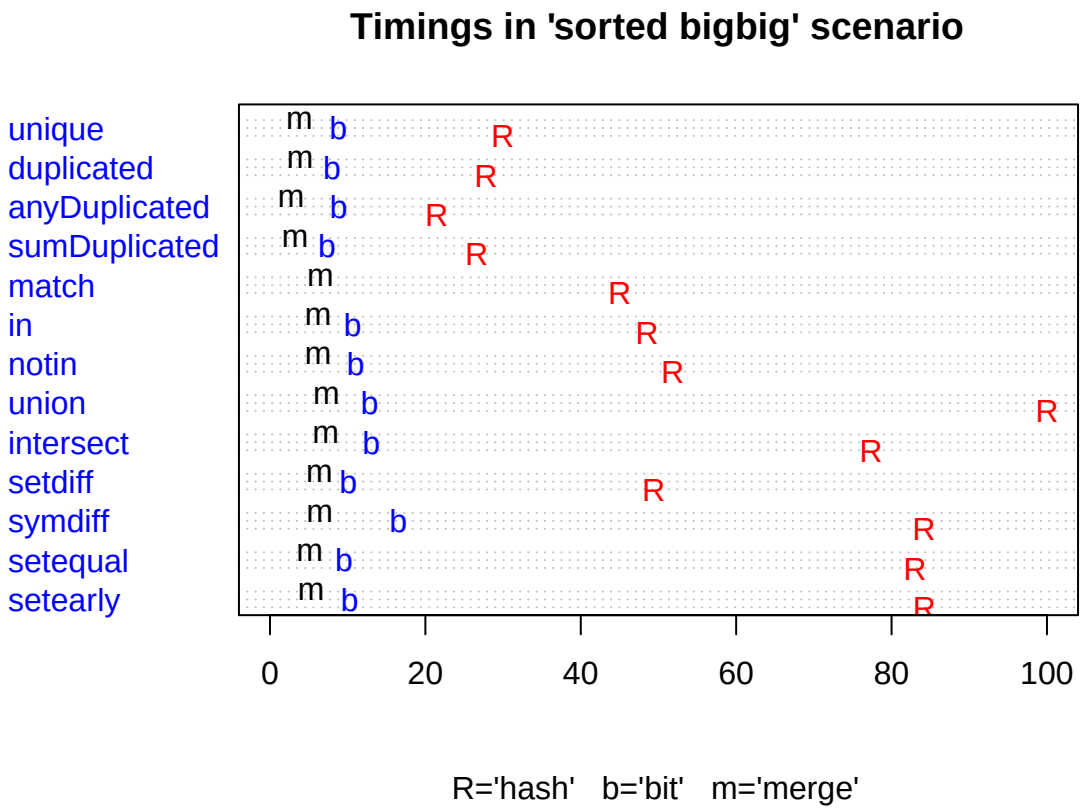

Figure 6: Execution time for R, bit and merge in 'sorted bigbig' scenario

% time for sorting

Table 14: sorted data relative to R's sort

	small	big
sort	174.7	608.7
sortunique	91.9	43.2

Table 15: unsorted data relative to R's sort

	small	big
sort	80.7	57.4
sortunique	19.4	10.7

% time for unique

Table 16: sorted data relative to R

	small	big
bit	140.2	29.4
merge	30.0	12.7
sort	0.0	0.0

Table 17: unsorted data relative to R

	small	big
bit	136.6	15.2

% time for anyDuplicated

Table 20: sorted data relative to R

	small	big
bit	185.8	41.2
merge	33.7	12.7
sort	0.0	0.0

Table 21: unsorted data relative to R

	small	big
bit	213.4	13.8
merge	533.4	40.1
sort	498.5	36.0

% time for sumDuplicated

Table 22: sorted data relative to R

	small	big
bit	145.1	27.6
merge	26.8	12.2
sort	0.0	0.0

Table 23: unsorted data relative to R

	small	big
bit	165.3	12.2
merge	365.1	37.0
sort	340.9	32.6

% time for match

Table 24: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	NA	NA	NA	NA
merge	54	0	25.2	14.5
sort	0	0	0.0	0.0

Table 25: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	NA	NA	NA	NA
merge	802.6	41.8	117.0	40.2

	smallsmall	smallbig	bigsmall	bigbig
sort	743.2	41.8	102.6	35.3

% time for in

Table 26: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	229.3	5.5	16.3	21.9
merge	45.5	0.0	22.1	12.8
sort	0.0	0.0	0.0	0.0

Table 27: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	234.0	2.8	9.6	9.4
merge	700.4	41.5	105.4	38.7
sort	655.6	41.5	92.2	34.3

% time for notin

Table 28: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	191.1	6.7	14.3	21.3
merge	26.1	0.0	19.4	12.0
sort	0.0	0.0	0.0	0.0

Table 29: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	471.4	2.8	9.8	11.1
merge	607.4	39.9	96.7	43.1
sort	566.6	39.9	84.5	38.1

% time for union

Table 30: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	81.6	22.7	30.7	12.8
merge	53.9	10.4	10.9	7.3
sort	0.0	0.0	0.0	0.0

Table 31: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	84.3	13.1	17.4	7.8
merge	352.8	38.3	47.2	30.1
sort	297.2	33.3	41.0	26.1

% time for intersect

Table 32: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	79.5	15.4	26.0	16.9
merge	26.7	0.0	0.1	9.3
sort	0.0	0.0	0.0	0.0

Table 33: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	102.8	7.5	16.0	7.6
merge	321.5	44.2	99.5	27.8
sort	292.9	44.2	99.4	24.1

% time for setdiff

Table 34: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	101.1	8.6	18.2	20.4
merge	36.1	0.0	6.7	12.9
sort	0.0	0.0	0.0	0.0

Table 35: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	100.4	5.1	11.4	9.5
merge	453.6	45.8	28.2	43.5
sort	418.4	45.8	24.5	38.4

% time for symdiff

Table 36: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	75.6	12.0	9.9	19.7
merge	16.8	3.8	4.1	7.6
sort	0.0	0.0	0.0	0.0

Table 37: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	71.9	6.8	6.1	6.9
merge	193.9	13.9	12.3	19.7
sort	177.9	11.9	10.4	17.4

% time for setequal

Table 38: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	102.0	105.1	13.0	11.5
merge	28.9	27.3	5.2	6.2
sort	0.0	0.0	0.0	0.0

Table 39: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	111.0	104.5	5.1	5.2
merge	349.6	67376.0	14.7	25.7
sort	321.7	67350.0	11.9	23.2

% time for setearly

Table 40: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	66.5	5.3	14.7	12.2
merge	18.1	0.0	0.1	6.3
sort	0.0	0.0	0.0	0.0

Table 41: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	105.9	3.0	10.2	4.5
merge	344.8	30.5	104.7	20.0
sort	316.0	30.5	104.6	18.0