Performance of the bit package

Dr. Jens Oehlschlägel

2024-09-19

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	41.6	42.2

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 `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.

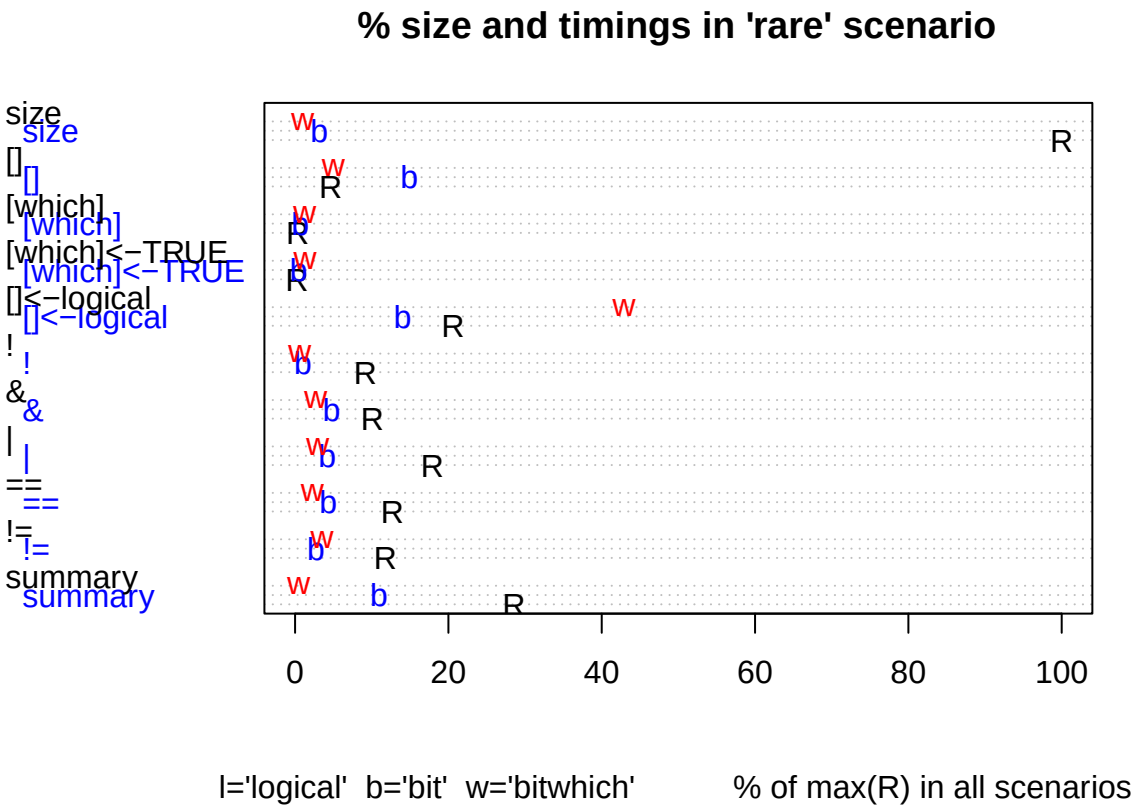

Figure 1: % size and execution time for bit (b) and bitwhich (w) relative to logical (R) in the 'rare' scenario

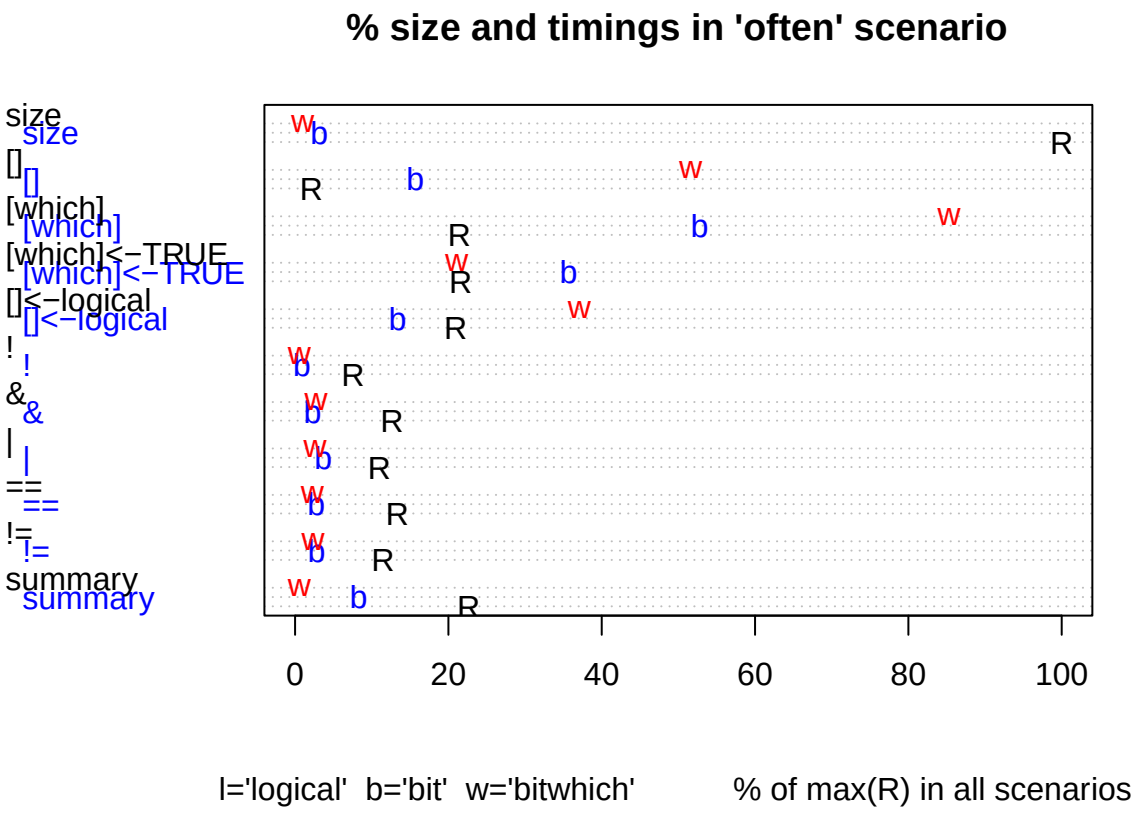

Figure 2: % size and execution time for bit (b) and bitwhich (w) relative to logical (R) in the 'often' scenario

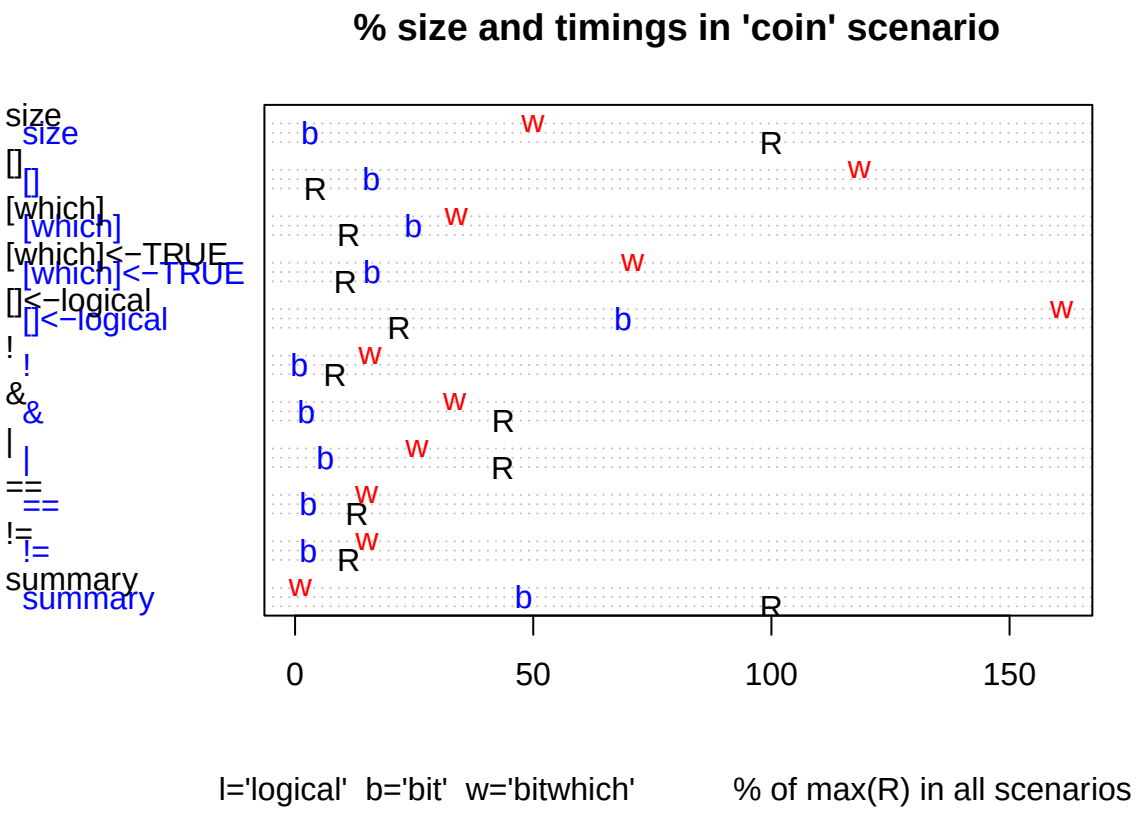

Figure 3: % size and execution time for bit (b) and bitwhich (w) relative to logical (R) in the 'coin' scenario

% 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	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	4.3	2.1	4.6	NA
bit	16.0	15.7	14.9	NA
bitwhich	118.5	51.6	5.0	NA
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time assigning

Table 5: % time of logical

	coin	often	rare	chunk
logical	21.8	20.9	20.6	NA
bit	68.9	13.4	14.0	NA
bitwhich	160.9	37.1	42.9	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	11.3	21.5	0.3	NA
bit	24.8	52.8	0.7	NA
bitwhich	33.9	85.3	1.2	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	10.6	21.6	0.3	NA
bit	16.1	35.7	0.5	NA
bitwhich	70.9	21.1	1.3	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	8.4	7.6	9.2	8.0
bit	0.9	0.9	1.0	0.9
bitwhich	15.8	0.6	0.6	1.8
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean AND

Table 9: % time for Boolean &

	coin	often	rare	chunk
logical	43.7	12.6	10.1	9.2
bit	2.4	2.3	4.8	2.4
bitwhich	33.5	2.7	2.7	4.4
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean OR

Table 10: % time for Boolean |

	coin	often	rare	chunk
logical	43.6	11.0	17.9	11.5
bit	6.4	3.7	4.2	2.3
bitwhich	25.6	2.6	3.0	4.9
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean EQUALITY

Table 11: % time for Boolean ==

	coin	often	rare	chunk
logical	13.0	13.4	12.7	12.4

	coin	often	rare	chunk
bit	2.8	2.8	4.4	2.9
bitwhich	15.0	2.2	2.3	3.2
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean XOR

Table 12: % time for Boolean !=

	coin	often	rare	chunk
logical	11.3	11.5	11.8	13.3
bit	2.8	2.8	2.7	2.4
bitwhich	15.1	2.3	3.5	3.3
which	NA	NA	NA	NA
ri	NA	NA	NA	NA

% time Boolean SUMMARY

Table 13: % time for Boolean summary

	coin	often
logical	100	22.6
bit	48	8.3

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)

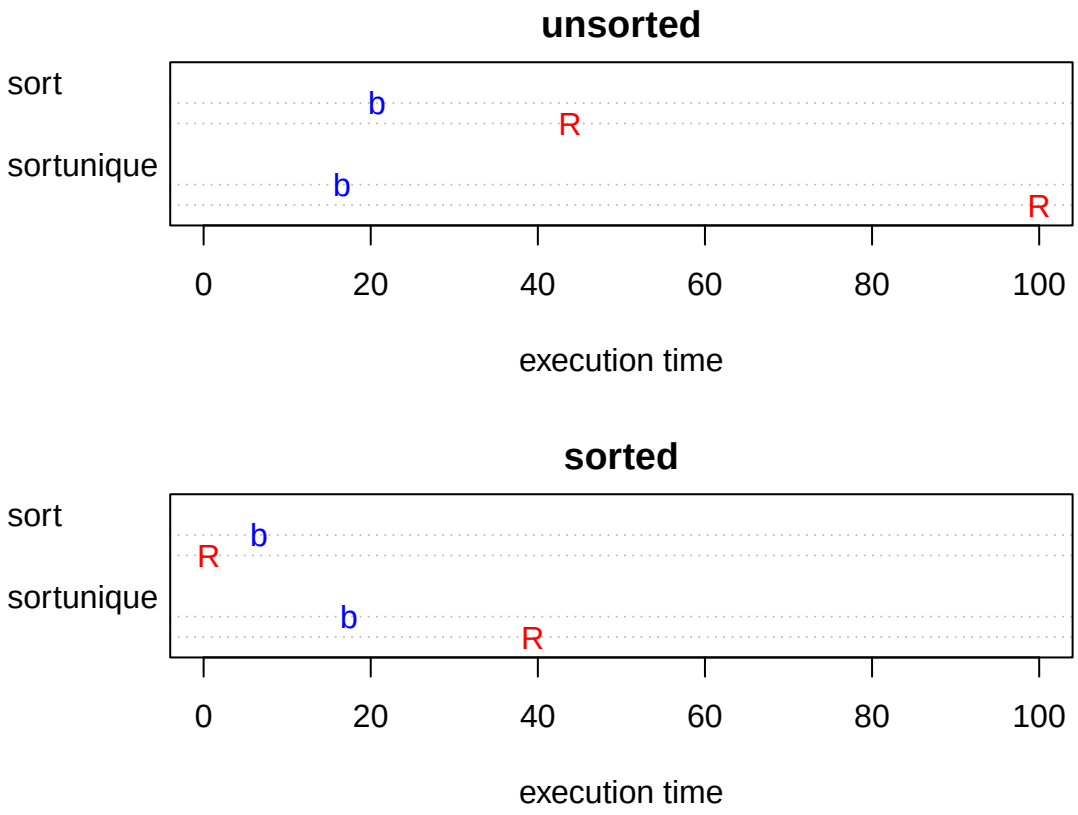

Figure 4: Execution time for R (R) and bit (b)

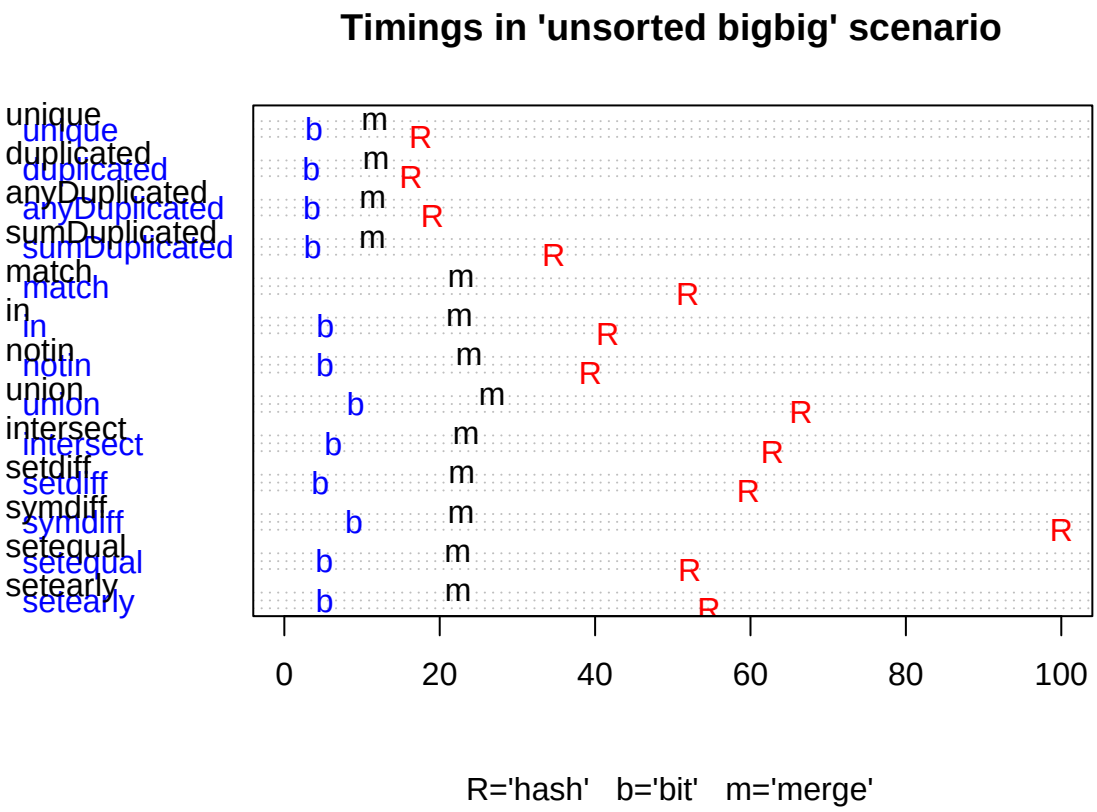

Figure 5: Execution time for R, bit and merge relative to most expensive R in 'unsorted bigbig' scenario

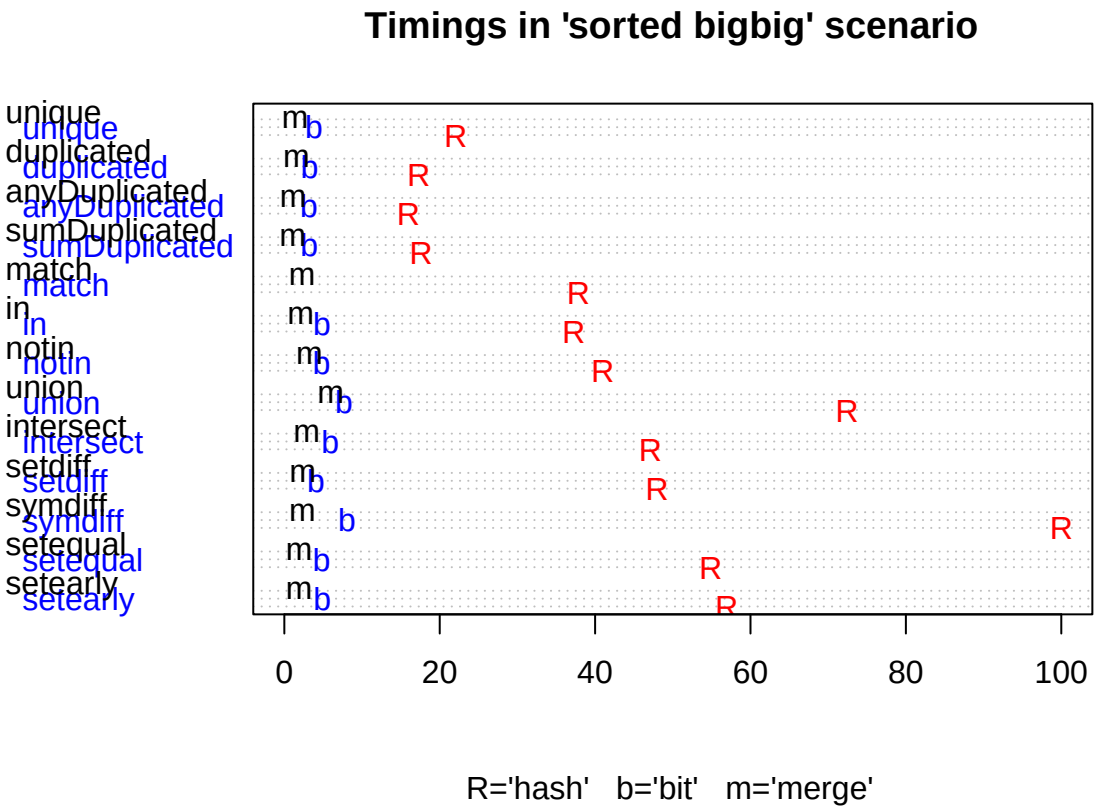

Figure 6: Execution time for R, bit and merge in 'sorted bigbig' scenario

% time for sorting

Table 14: sorted data relative to R's sort

	small	big
sort	282	1055.2
sortunique	164	44.2

Table 15: unsorted data relative to R's sort

	small	big
sort	35.6	47.3
sortunique	21.0	16.6

% time for unique

Table 16: sorted data relative to R

	small	big
bit	204.2	17.3
merge	70.0	6.4
sort	0.0	0.0

Table 17: unsorted data relative to R

	small	big
bit	159.5	21.8
merge	215.1	66.4
sort	141.9	57.7

% time for duplicated

Table 18: sorted data relative to R

	small	big
bit	89.7	18.8
merge	20.5	9.0
sort	0.0	0.0

Table 19: unsorted data relative to R

	small	big
bit	247.5	21.2
merge	252.5	72.5
sort	164.9	62.2

% time for anyDuplicated

Table 20: sorted data relative to R

	small	big
bit	314.2	19.9
merge	37.7	7.2
sort	0.0	0.0

Table 21: unsorted data relative to R

	small	big
bit	379.5	18.8
merge	222.9	59.7
sort	183.5	53.1

% time for sumDuplicated

Table 22: sorted data relative to R

	small	big
bit	153	18.4
merge	28	6.2
sort	0	0.0

Table 23: unsorted data relative to R

	small	big
bit	132.2	10.5
merge	151.2	32.7
sort	125.3	29.2

% time for match

Table 24: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	NA	NA	NA	NA
merge	54	0	38.6	6.1
sort	0	0	0.0	0.0

Table 25: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	NA	NA	NA	NA
merge	380.2	34	117.1	43.8

	smallsmall	smallbig	bigsmall	bigbig
sort	255.0	34	98.5	39.0

% time for in

Table 26: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	238.5	3	8.3	13.1
merge	46.3	0	24.4	5.6
sort	0.0	0	0.0	0.0

Table 27: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	364.0	5.4	5.4	12.7
merge	274.8	69.6	95.6	54.2
sort	227.6	69.5	81.2	48.7

% time for notin

Table 28: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	258.3	3.7	8.9	11.7
merge	42.3	0.0	8.0	7.9
sort	0.0	0.0	0.0	0.0

Table 29: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	314.0	5.8	4.9	13.4
merge	152.9	76.3	72.2	60.4
sort	125.7	76.3	68.0	51.4

% time for union

Table 30: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	108.6	28.3	26.6	10.6
merge	357.2	19.3	17.9	8.2
sort	0.0	0.0	0.0	0.0

Table 31: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	50.5	37.5	35.9	13.8
merge	126.9	80.6	75.8	40.2
sort	34.3	54.8	52.2	30.5

% time for intersect

Table 32: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	81.4	6.9	3.4	12.6
merge	23.3	0.0	0.0	6.1
sort	0.0	0.0	0.0	0.0

Table 33: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	66.9	7.9	5.2	10.1
merge	110.4	52.1	33.9	37.3
sort	86.8	52.1	33.9	32.3

% time for setdiff

Table 34: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	106.5	2.9	23.4	8.5
merge	24.5	0.0	16.0	4.9
sort	0.0	0.0	0.0	0.0

Table 35: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	82.4	4.0	17.4	7.8
merge	103.5	68.8	57.9	38.2
sort	83.9	68.7	39.1	33.9

% time for symdiff

Table 36: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	51.6	12.6	8.7	8.1
merge	21.7	6.4	6.3	2.3
sort	0.0	0.0	0.0	0.0

	smallsmall	smallbig	bigsmall	bigbig

Table 37: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	25.7	15.4	17.3	9.0
merge	32.6	25.0	28.7	22.8
sort	22.3	17.3	19.6	20.3

% time for setequal

Table 38: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	86.7	73.1	6.8	8.8
merge	23.5	32.9	2.8	3.5
sort	0.0	0.0	0.0	0.0

Table 39: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	104.8	150.0	4.6	9.9
merge	144.2	55483.5	10.9	42.8
sort	117.8	55443.2	9.0	38.8

% time for setearly

Table 40: sorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	77.5	2.9	22.0	8.6
merge	23.3	0.0	0.1	3.4
sort	0.0	0.0	0.0	0.0

Table 41: unsorted data relative to R

	smallsmall	smallbig	bigsmall	bigbig
bit	108.7	2.5	10.7	9.5
merge	130.4	27.2	114.3	40.9
sort	104.6	27.2	114.3	37.0