

# Lazy vs. non lazy

*O. Denas*

*12/28/2016*

## Contents

<b>1</b>	<b>Double vs. single rank</b>	<b>1</b>
<b>2</b>	<b>Lazy vs non-lazy</b>	<b>2</b>
2.1	Input properties . . . . .	2
2.2	Code . . . . .	4
2.3	Run time . . . . .	5
2.4	Sandbox timing . . . . .	6
2.5	Check . . . . .	7

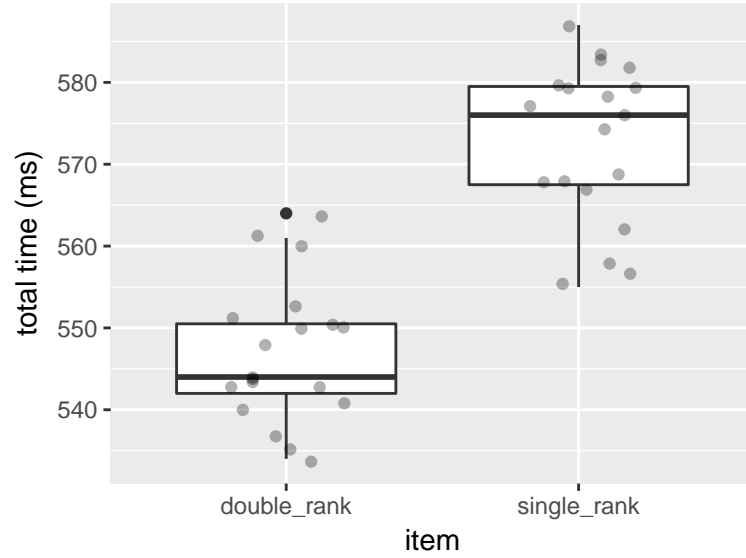
## 1 Double vs. single rank

Table 1: Time (in ms) of 500K calls to `w1()` based on `single_rank()` or `double_rank()` methods on 100MB random DNA input; Mean/sd over 20 repetitions.

item	avg_time	sd_time
double_rank	546.89	8.49
single_rank	572.74	9.65

Table 2: Absolute (`double_rank / single_rank`) and relative ( $100 * |double\_rank - single\_rank| / single\_rank$ ) ratios of average times.

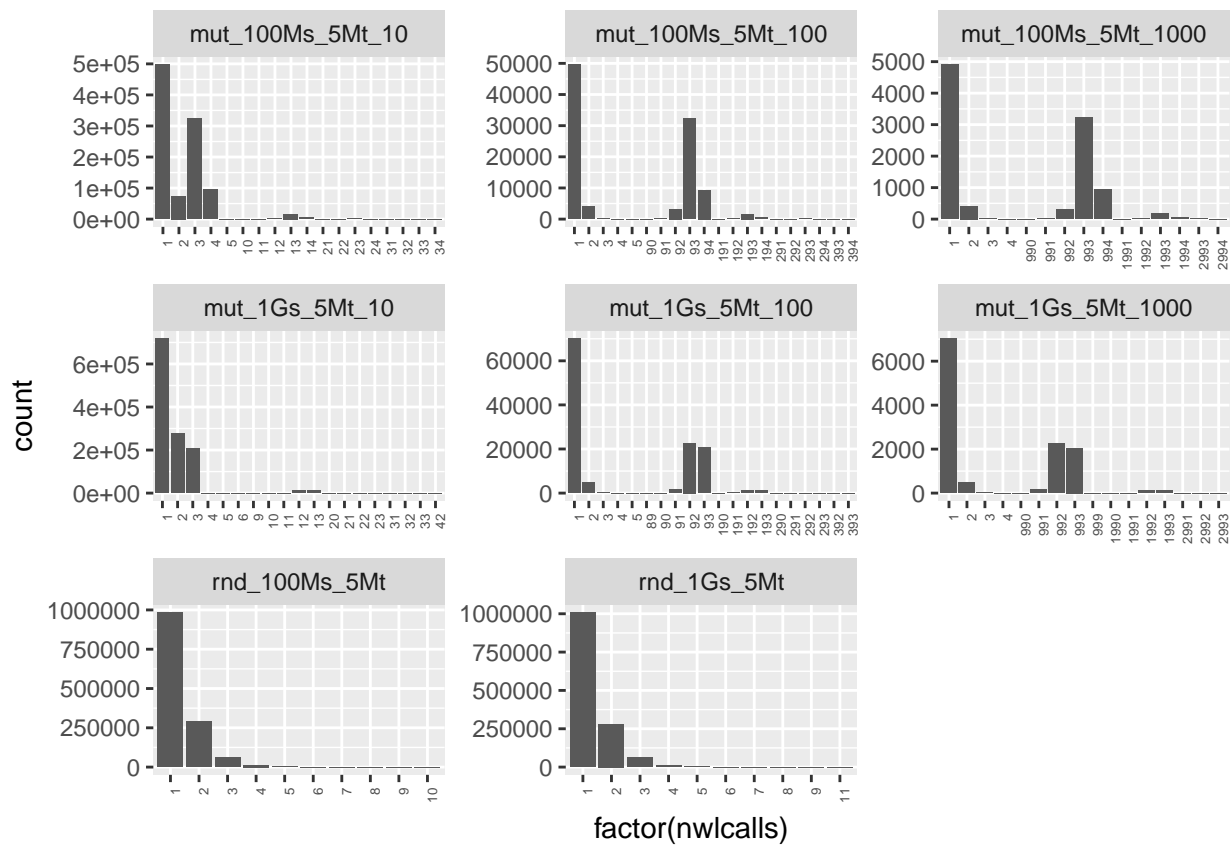
.	double_rank	single_rank	abs_ratio	rel_ratio
.	546.89	572.74	0.95	4.51



## 2 Lazy vs non-lazy

### 2.1 Input properties

For various types (“mut\_XMs\_YMt\_Z” means  $s$  and  $t$  are random identical strings of length  $X$ , and  $Y$  million respectively with mutations inserted every  $Z$  characters. “rnd\_XMs\_YMt” means  $s$  and  $t$  are random strings of length  $X$ , and  $Y$  million respectively) of inputs run the MS algorithm and count the number of consecutive `lazy_w1()` calls.



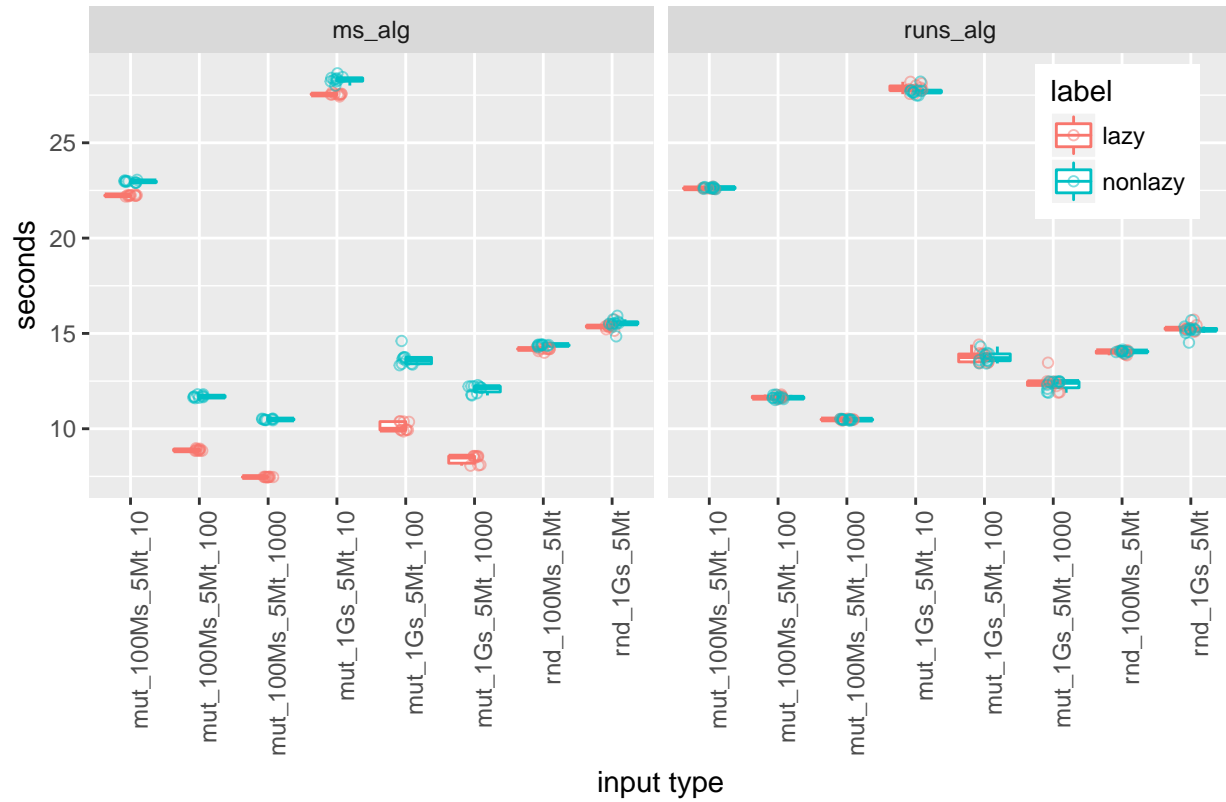
## 2.2 Code

The lazy and non-lazy versions differ in a couple of lines of code as follows

```
if(flags.lazy){
    for(; I.first <= I.second && h_star < ms_size; ){
        c = t[h_star];
        I = bstep_interval(st, I, c); //I.bstep(c);
        if(I.first <= I.second){
            v = st.lazy_wl(v, c);
            h_star++;
        }
    }
    if(h_star > h_star_prev) // // we must have called lazy_wl(). complete the node
        st.lazy_wl_followup(v);
} else { // non-lazy weiner links
    for(; I.first <= I.second && h_star < ms_size; ){
        c = t[h_star];
        I = bstep_interval(st, I, c); //I.bstep(c);
        if(I.first <= I.second){
            v = st.lazy_wl(v, c);
            h_star++;
        }
    }
}
```

## 2.3 Run time

### MS vector construction time



The right panel shows the time to construct the `runs` vector. This stage is the same for both versions and is shown as a control. On the left panel it can be seen that speedup correlates positively with both the size of the indexed string and the mutation period.

## 2.4 Sandbox timing

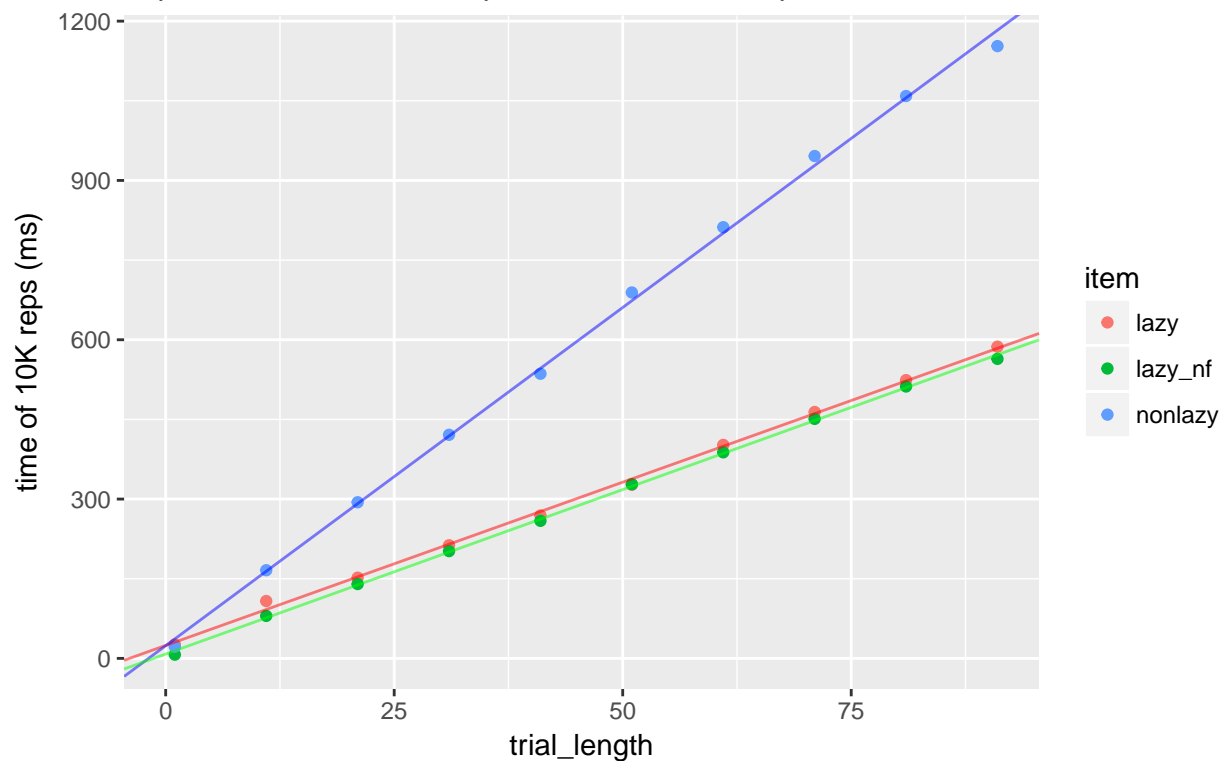
Measure the time of 10k repetitions of

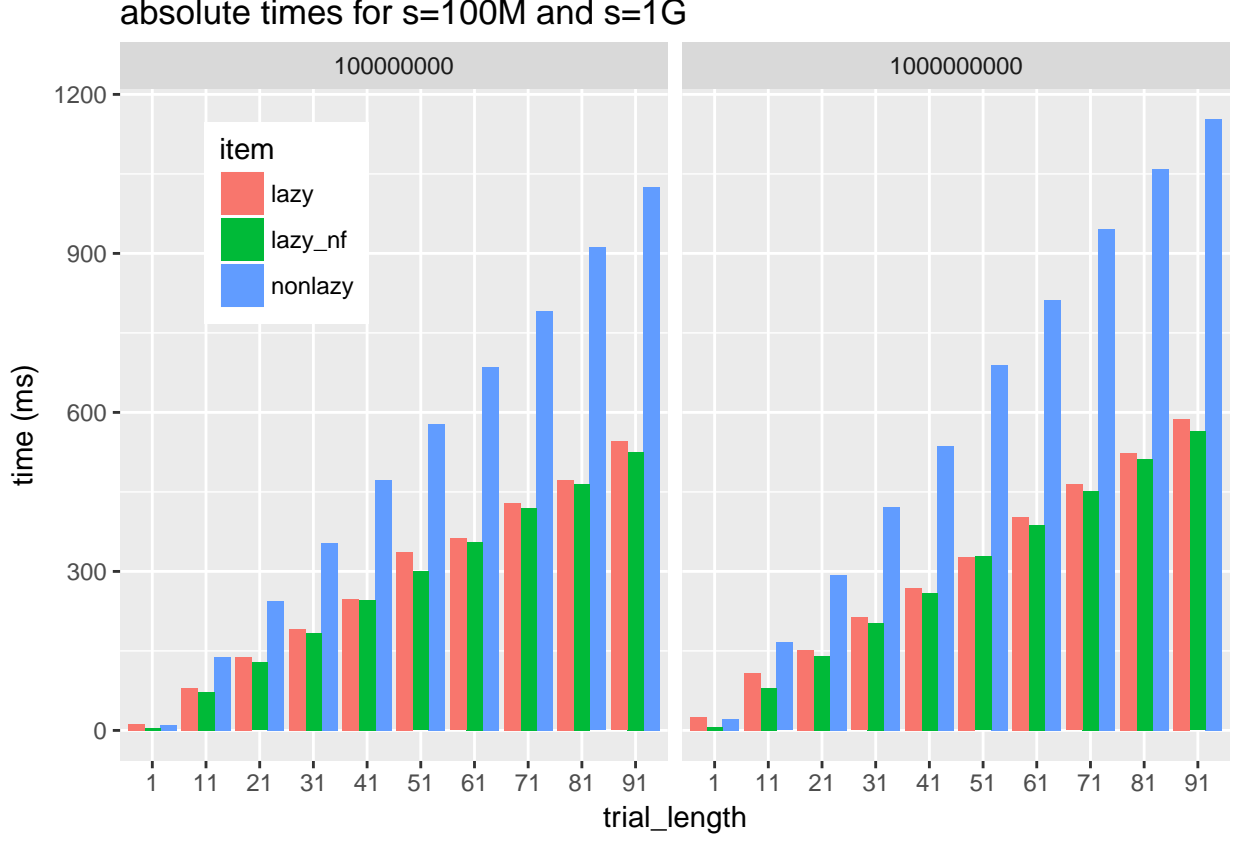
- (lazy)  $n$  consecutive `lazy_wl()` calls followed by a `lazy_wl_followup()`
- (nonlazy)  $n$  consecutive `wl()` calls
- (lazy\_nf)  $n$  consecutive `lazy_wl()` calls

```
// lazy
for(size_type i = 0; i < trial_length; i++)
    v = st.lazy_wl(v, s_rev[k--]);
if(h_star > h_star_prev) // // we must have called lazy_wl(). complete the node
    st.lazy_wl_followup(v);
...
// non-lazy
for(size_type i = 0; i < trial_length; i++)
    v = st.wl(v, s_rev[k--]);
...
// lazy_nf
for(size_type i = 0; i < trial_length; i++)
    v = st.lazy_wl(v, s_rev[k--]);
```

indexed input size 1G

lazy:  $24.34 + 6.1491*n$ ; nonlazy:  $23.90 + 12.7370*n$ ; lazy\_nf:  $8.21 + 6.1933*n$





## 2.5 Check

In the experiments above we ran the program with the “lazy” or “non-lazy” flag and measured. The total time of each experiment can be written as  $t_l = l_l + a$  and  $t_n = l_n + a$  for the two versions respectively; only the  $t$ s being known. Furthermore, we have  $\hat{l}_l$  and  $\hat{l}_n$  estimations. Hence we should expect

$$\delta t = t_l - t_n = l_l + a - l_n - a = l_l - l_n \approx \delta \hat{l} = \hat{l}_l - \hat{l}_n$$

b_path	t_l	t_n	l_l	l_n	delta_t	delta_l_hat
mut_100Ms_5Mt_10	22.23	22.97	4.05	3.53	-0.74	0.52
mut_100Ms_5Mt_100	8.89	11.70	2.90	4.07	-2.81	-1.16
mut_100Ms_5Mt_1000	7.47	10.49	2.79	4.12	-3.02	-1.33
mut_1Gs_5Mt_10	27.54	28.32	4.08	4.08	-0.78	-0.01
mut_1Gs_5Mt_100	10.11	13.66	3.15	4.65	-3.55	-1.50
mut_1Gs_5Mt_1000	8.40	12.09	3.05	4.70	-3.68	-1.65
rnd_100Ms_5Mt	14.17	14.38	4.68	3.65	-0.20	1.03
rnd_1Gs_5Mt	15.38	15.51	4.15	3.94	-0.14	0.20

The numbers are not identical (process dependent factors might influence the running time of function calls), but they are correlated ( $\text{corr}(\delta t, \delta \hat{l}) = 0.96$ ).