Project 1: Genome Scale algorithms

Maria, Mateo and Jacob

2/20/2018

```
# Import Naive
df_Naive = read.csv("Test ExactSubstring_Naive.csv", header = FALSE)

colnames(df_Naive) = c("Function", "Time", "n", "m", "z", "RunName")

df_Naive$RunName = factor(x = df_Naive$RunName)
df_Naive$Function = factor(x = df_Naive$Function)

# Import KMP
df_KMP = read.csv("Test ExactSubstring_KMP.csv", header = FALSE)

colnames(df_KMP) = c("Function", "Time", "n", "m", "z", "RunName")

df_KMP$RunName = factor(x = df_KMP$RunName)
df_KMP$Function = factor(x = df_KMP$Function)
```

Program description

The programs are implemented in Python, we had no unsolved issues implementing the algorithms. Execution of the program on commandline takes the form: python search-naive.py [text-filename] [str-pattern] python search-kmp.py [text-filename] [str-pattern] and the output is directly to the console.

Test of search-naive

We used the data provided for testing which outputs the following results: python search-naive.py banana.txt ana 2 4

python search-naive.py mississippi.txt ss 3 6

python search-naive.py walrus-and-carpenter.txt Walrus 5 660 915 1059 1773 1951 2326 2617 2855 3050

python search-naive.py ancient-mariner.txt Albatross 2876 2991 3272 3443 5586 6899 14180 19139 23729

Test of search-kmp

python search-kmp.py banana.txt ana 2 4

python search-kmp.py mississippi.txt ss 3 6

python search-kmp.py walrus-and-carpenter.txt Walrus 5 660 915 1059 1773 1951 2326 2617 2855 3050

python search-kmp.py ancient-mariner.txt Albatross 2876 2991 3272 3443 5586 6899 14180 19139 23729

Runtime - Naive

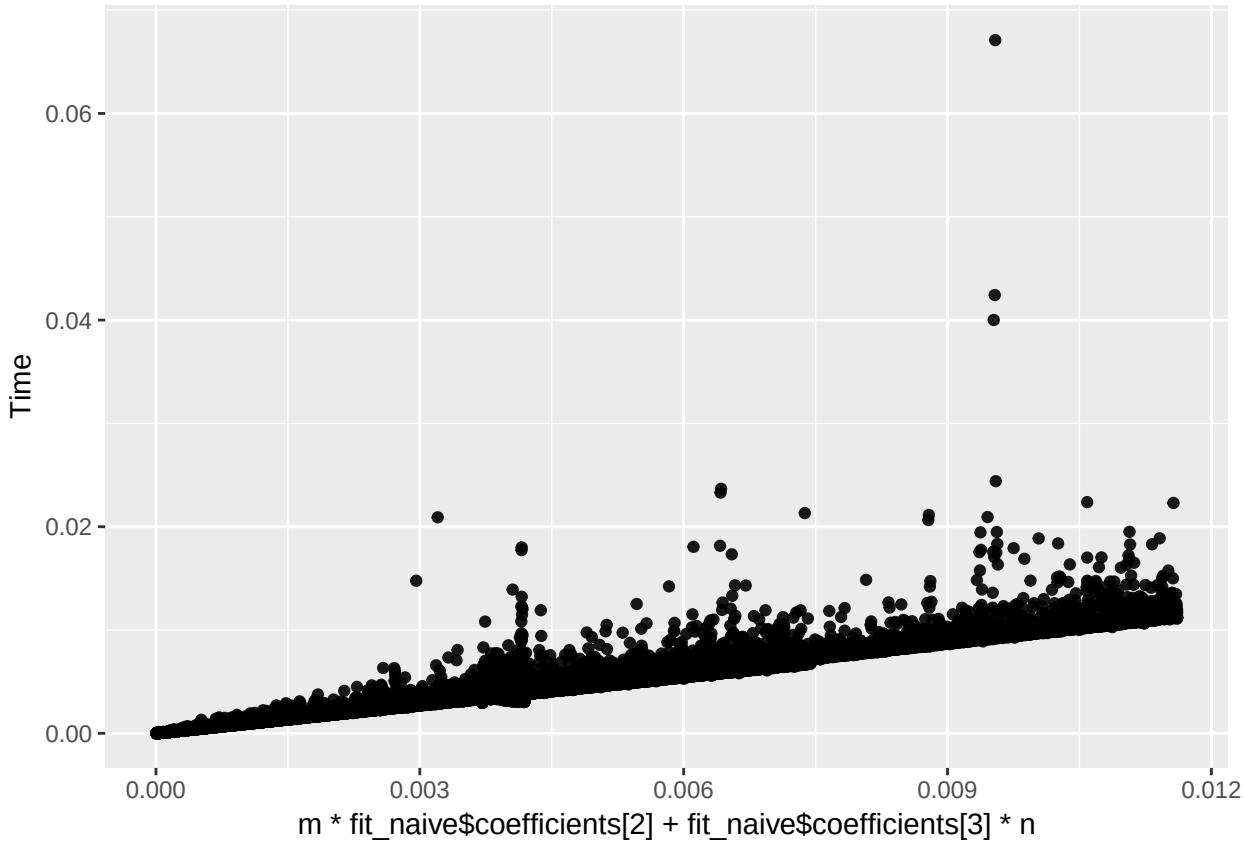
The implementation of the naive algorithm runs in O(n * m) time. With best case input at 0 occurrences of pattern in the text, and worst case is constant matching to keep the inner loop active (m = n^0.5).

The following plot shows the runtime / regression of the Y axis and regression on X axis. Where the regression is: Time ~ n + m. It clearly shows that the runtime is more than linear.

```
fit_naive = lm( data = df_Naive, formula = Time ~ m + n)
fit_naive
```

```
##
## Call:
## lm(formula = Time ~ m + n, data = df_Naive)
##
## Coefficients:
## (Intercept)            m            n
##  -1.827e-04    2.399e-07    3.711e-07
```

```
ggplot( data = df_Naive ) +
  geom_point(
    aes(
      y = Time,
      x = m * fit_naive$coefficients[2] + fit_naive$coefficients[3] * n
    ),
    color = "#000000", alpha = 0.9
  )
```



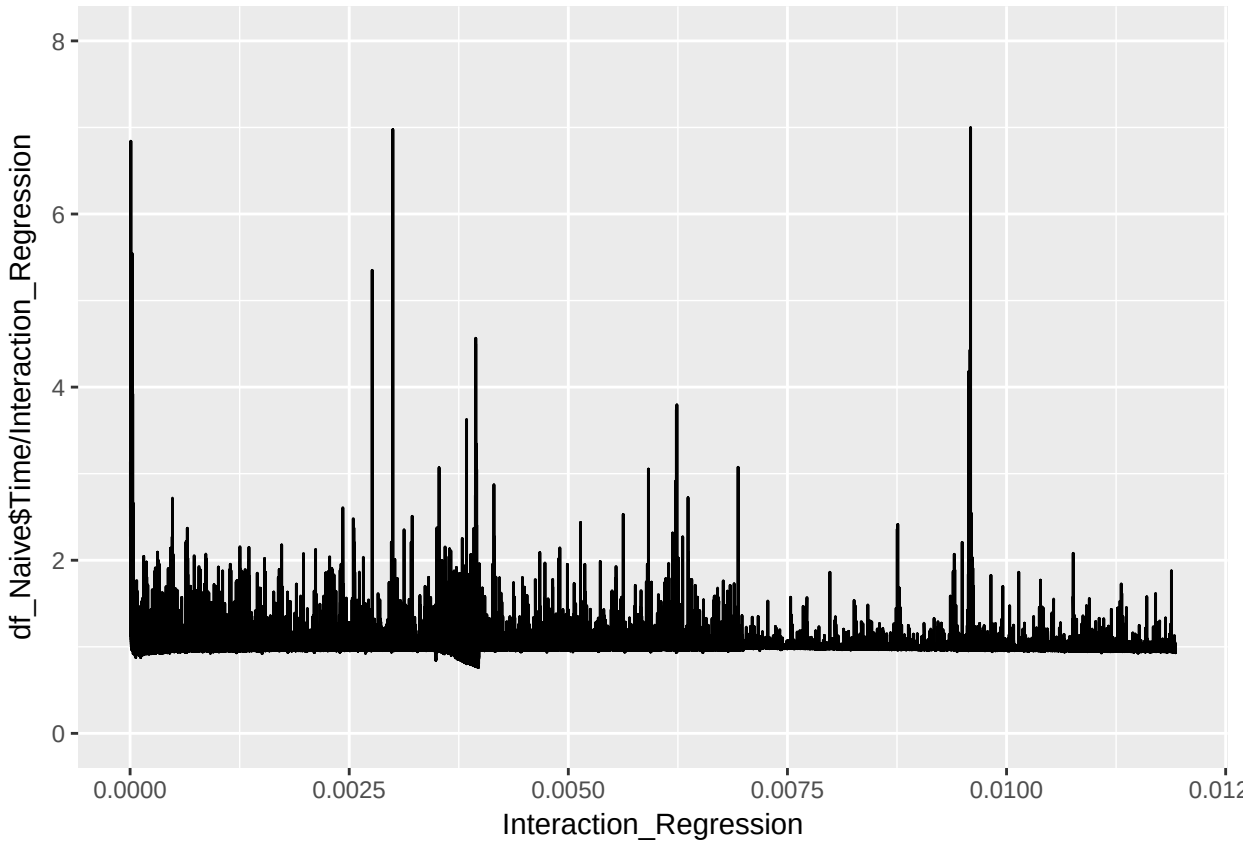
The following plot shows the runtime / regression of the Y axis and regression on X axis. Where the regression

is: Time ~ n * m. This plot flatlines around y = 1, with random spikes of longer runtime which can be explained by background resource usage.

```
fit_naive_poly = glm( data = df_Naive, formula = Time ~ n * m )
fit_naive_poly
```

```
##
## Call:  glm(formula = Time ~ n * m, data = df_Naive)
##
## Coefficients:
## (Intercept)            n            m          n:m
##   4.503e-05    3.488e-07   -7.732e-08    2.443e-11
##
## Degrees of Freedom: 59999 Total (i.e. Null);  59996 Residual
## Null Deviance:       0.3888
## Residual Deviance: 0.0191    AIC: -727300
```

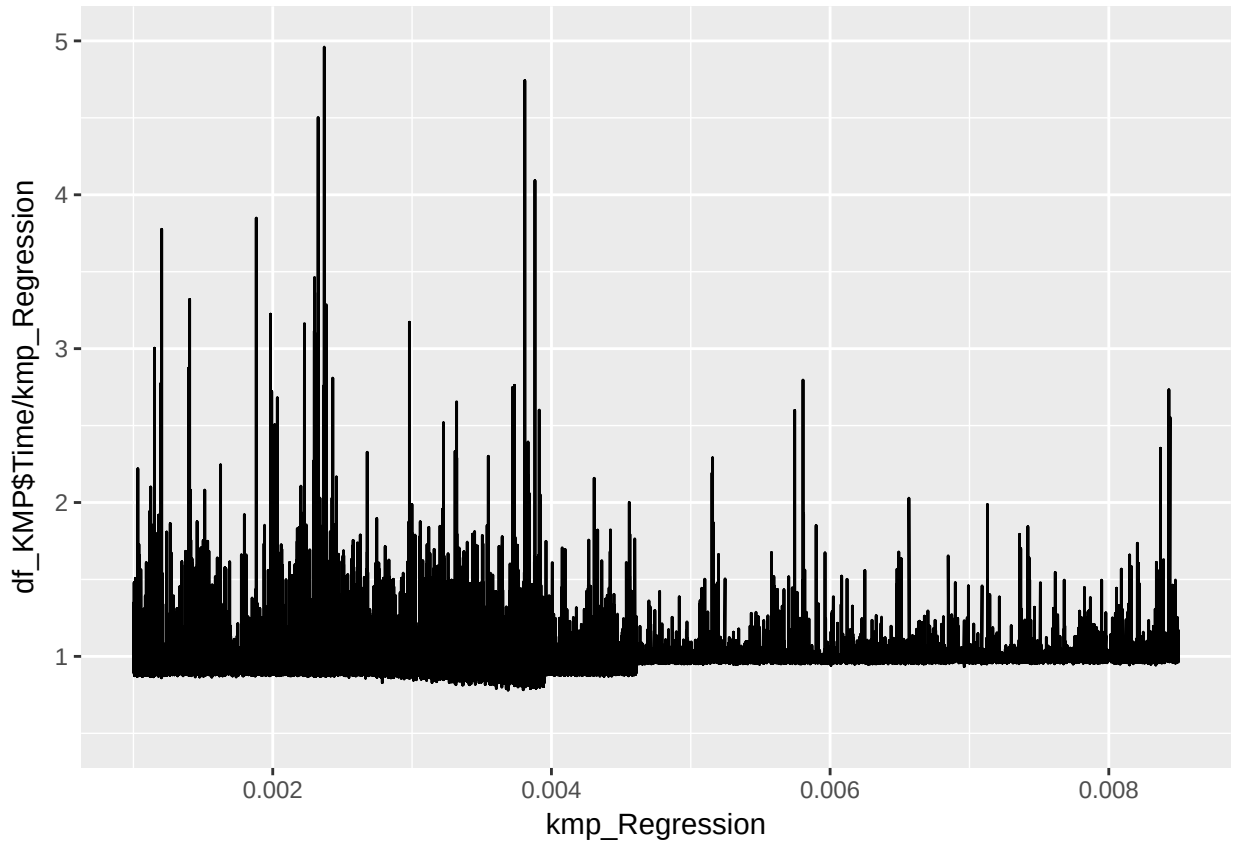
```
inc = fit_naive_poly$coefficients[1]
c_n = fit_naive_poly$coefficients[2]
c_m = fit_naive_poly$coefficients[3]
c_nm = fit_naive_poly$coefficients[4]

Interaction_Regression = inc + c_n * df_Naive$n + c_m * df_Naive$m + df_Naive$n * df_Naive$m * c_nm
Interaction_Regression = 0 + c_n * df_Naive$n + 0 * df_Naive$m + df_Naive$n * df_Naive$m * c_nm
Interaction_Regression = 0 + c_n * df_Naive$n + 0 * df_Naive$m + df_Naive$n * df_Naive$m * c_nm

ggplot(  ) +
  geom_line(
    aes(
      y = df_Naive$Time / Interaction_Regression,
      x = Interaction_Regression
    )
  ) + ylim(0, 8)
```



Runtime - KMP

The implementation of the KMP algorithm runs in O(n + m) time. The best case is a pattern with a simple failure function (all pointing to the base case), where the worst case keeps the inner loops busy by jumping with failures. The following plot shows the runtime / regression of the Y axis and regression on X axis. Where the regression is: Time ~ n + m. This plot flatlines around y = 1, with random spikes of longer runtime which can be explained by background resource usage.

```
fit_kmp = glm( data = df_KMP, formula = Time ~ n + m )
inc = fit_kmp$coefficients[1]
c_m = fit_kmp$coefficients[3]
c_n = fit_kmp$coefficients[2]
kmp_Regression = inc + c_n * df_KMP$n + c_m * df_KMP$m
kmp_Regression = 0 + c_n * df_KMP$n + c_m * df_KMP$m

ggplot(  ) +
  geom_line(
    aes(
      y = df_KMP$Time / kmp_Regression,
      x = kmp_Regression
    )
  ) + xlim(0.001, 0.0085) + ylim(0.5, 5)
```



Testdata

Tests of runtime was done the same way for both algorithms. - varying m and keeping n the same. - keepoing m the same and varying n. - varying both n and m together.

This was done with random generated strings (text and pattern), and another run with best case (text = "aaa…", pattern = "bbb…") and worst case (text = "aaa…", pattern = "aaa…").

Number of occurrences (z)

The number of occurrences of the pattern in the string has no significant impact on the naive implementation. Here is the p-value of an ANOVA of the addition to the regression.

```
fit_naive = lm( data = df_Naive, formula = Time ~ n + m )
fit_naive_z = lm( data = df_Naive, formula = Time ~ n + m + z )

an_naive = anova(fit_naive, fit_naive_z)
an_naive$`Pr(>F)`
```

```
## [1]        NA 0.1065594
```

The number of occurrences of the pattern in the string has no significant impact on the KMP implementation. Here is the p-value of an ANOVA of the addition to the regression.

```
fit_kmp = lm( data = df_KMP, formula = Time ~ n + m )
fit_kmp_z = lm( data = df_KMP, formula = Time ~ n + m + z )
```

```r
an_kmp = anova(fit_kmp, fit_kmp_z)
an_kmp$`Pr(>F)`
```

```
## [1]         NA 0.09159061
```

Measuring time in a different way

We have tested two different scanarios; the best case and the worst case (bc,wc). For Naive algorithm we have implemented the worst case as sequence with same repeated characters compared with patterns that that are roughly equal size as n and also that sequence **S** has n number of A's and the pattern **P** has (m-1) number of A's followed by a single B. The best case scenario would be matching pattern **P** with sequence **S** which has no occurances in **S**.

```r
library(dplyr)
library(ggpubr)
```

```
## Loading required package: magrittr
```

```
##
## Attaching package: 'magrittr'
```

```
## The following object is masked from 'package:purrr':
##
##     set_names
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

```r
df_Naive = read.csv("Test ExactSubstring_Naive_bc-wc.csv", header = FALSE)
df_KMP = read.csv("Test ExactSubstring_KMP_bc-wc.csv", header = FALSE)
colnames(df_Naive) = c("Function", "Time", "n", "m", "z", "RunName")
colnames(df_KMP) = c("Function", "Time", "n", "m", "z", "RunName")

dat_kmp = df_KMP
dat_naive = df_Naive
ploting <- function(dat_naive, dat_kmp){
      dat_naive <- dat_naive %>%
          group_by(n, m, RunName) %>%
            mutate(median_time = median(Time)) %>%
            mutate(type = "Naive")

      dat_kmp <- dat_kmp %>%
            group_by(n, m,RunName) %>%
            mutate(median_time = median(Time)) %>%
            mutate(type = "KMP")

      all <- rbind(dat_kmp, dat_naive)
      colnames(all) = c("Function", "Time", "n", "m", "z", "RunName","median_time","Type")

      constant_n <- all %>%
              filter(n == 5000)

      constant_m <- all %>%
              filter(m == 10)
```

```
      return(list(constant_n, constant_m, all))
}

fig <- ploting(df_Naive, df_KMP)
```

```
## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character
```

```
## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector
```

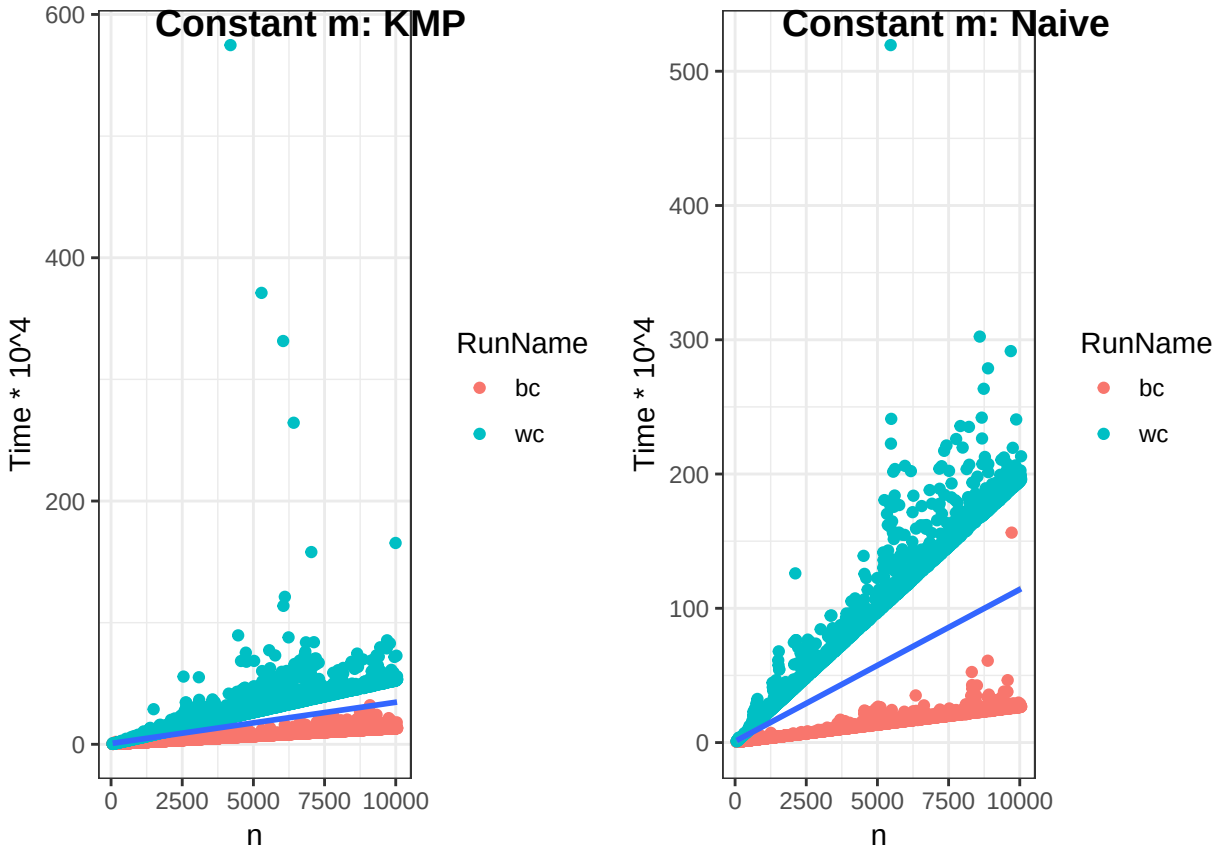
```
## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector
```

```
      #a <- qplot(y = Constant_n$Time/Constant_n$m, x = Constant_n$m) + ylim(0,0.001)
      #b <- qplot(y = Constant_m$Time/Constant_m$n, x = Constant_m$n) + ylim(0,2.5*10^-6)

      constant_n_1 <- ggplot(data = fig[[1]]) +
        geom_point(mapping = aes(y = Time/m, x = m, color = Type, shape = RunName)) + theme_bw() +  yli

      constant_m_n_linear <- ggplot(data = fig[[2]]) + theme_bw() +
        geom_point(mapping = aes(y = Time/n, x = n, color = Type,shape = RunName))+  ylim(0, 1e-6)


    kmp<- fig[[2]] %>%
      filter(Type == "KMP")

    naive<- fig[[2]] %>%
      filter(Type == "Naive")

    constant_m_kmp <- ggplot(data = kmp) +
        geom_point(mapping = aes(y = Time*10^4, x = n, color = RunName)) +
        geom_smooth(method = "lm", mapping = aes(y = Time*10^4, x = n)) + theme_bw()

    constant_m_naive <- ggplot(data = naive) +
        geom_point(mapping = aes(y = Time*10^4, x = n, color= RunName)) +
        geom_smooth(method = "lm", mapping = aes(y = Time*10^4, x = n)) + theme_bw()

    all_naive <- fig[[3]] %>%
      filter(Type == "Naive")
    all_KMP <- fig[[3]] %>%
      filter(Type == "KMP")

    all_mn_kmp <- ggplot(data = all_KMP) +
        geom_point(mapping = aes(y = median_time/(m*n), x = m, color = RunName))  + theme_bw()  + ylim(0
    all_mn_naive <- ggplot(data = all_naive) +
        geom_point(mapping = aes(y = median_time/(m*n), x = m, color = RunName))  + theme_bw() + ylim(0

    all_m_plus_n_kmp <- ggplot(data = all_KMP) +
        geom_point(mapping = aes(y = median_time/(m+n), x = m, color = RunName))  + theme_bw()
    all_m_plus_n_naive <- ggplot(data = all_naive) +
        geom_point(mapping = aes(y = median_time/(m+n), x = m, color = RunName))  + theme_bw()


# ggarrange(constant_m_kmp, constant_m_naive, all_mn_kmp, all_mn_naive,all_m_plus_n_kmp,all_m_plus_n_na
```

```
#       labels = c('Constant m: KMP ', 'Constant m: Naive', 'KMP: Time/n*m:' ,'Naive: Time/m*n', 'KMP:
#       ncol = 3, nrow = 3)
```

First we keep **m** constant and vary the **n** showing that slope is much more steeper when using Naive algorithm indicating that time is not linear. In KMP the slope does not change taking into account both, worst case and best case, which indicates linear time.

```
ggarrange(constant_m_kmp, constant_m_naive,
      labels = c('Constant m: KMP ', 'Constant m: Naive'),
      ncol = 2, nrow =1)
```

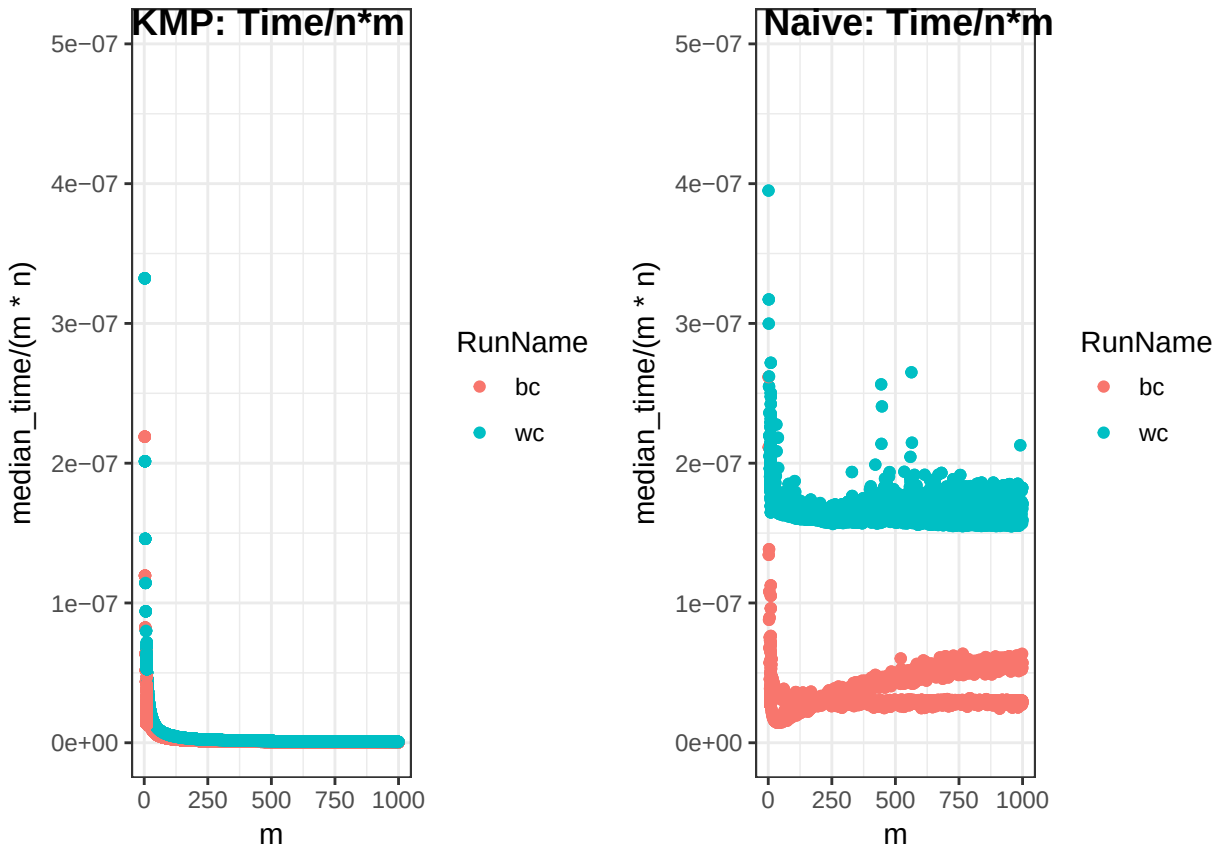


When we plot the time measurments of KMP algorithm by dividing ***(nm) we can see that points will tend to get closer to the 0, which indicates wrong scale. Since, KMP runs in m+n time we can not prove much using this plot. On the other hand, Naive algorithm confirms n*** **m** time consumption since it tends to become constant when **m** is above 200.

```
ggarrange(all_mn_kmp, all_mn_naive,
      labels = c('KMP: Time/n*m', 'Naive: Time/n*m'),
      ncol = 2, nrow =1)
```

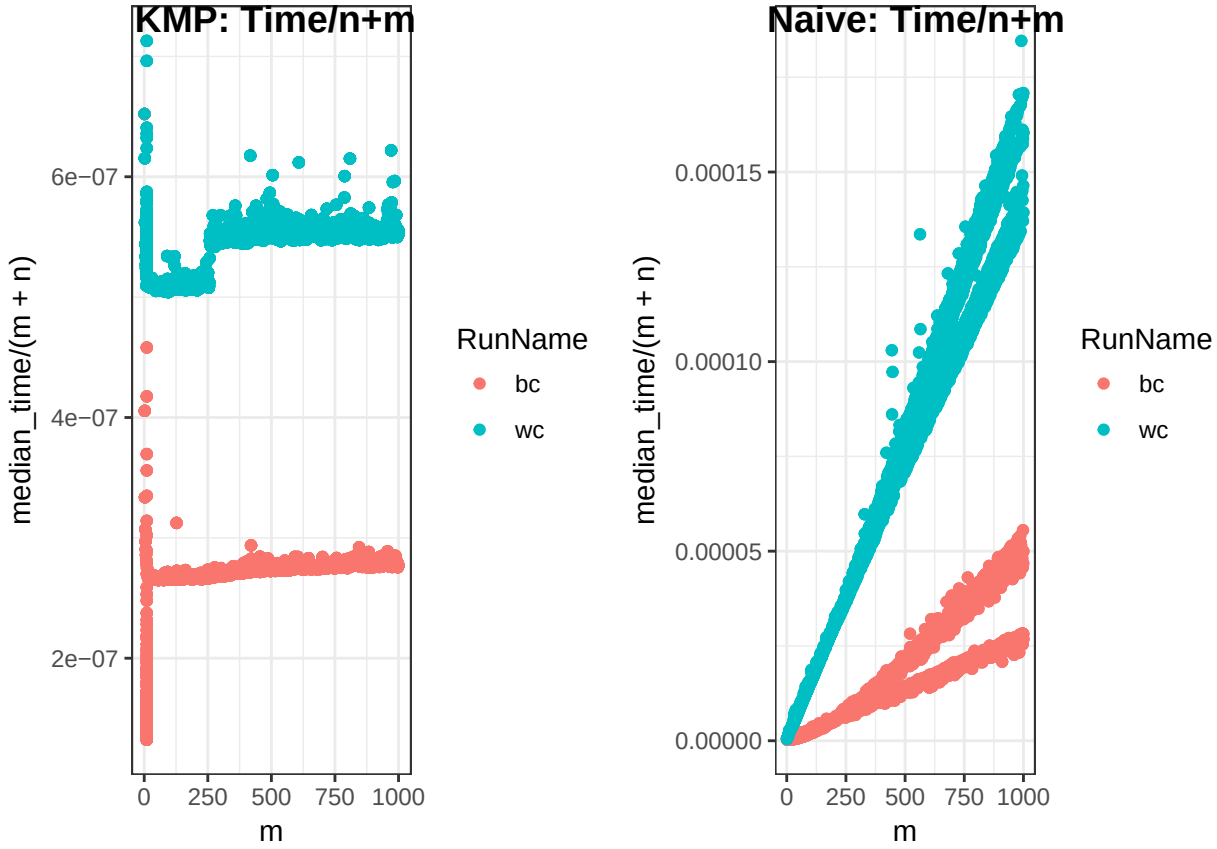
```
## Warning: Removed 20 rows containing missing values (geom_point).
```

```
## Warning: Removed 6 rows containing missing values (geom_point).
```



In this case, we want to prove that KMP runs in linear time and when you try to plot Naive algorithm on the scale n+m the running time does not reach a constant time. On KMP plot, we can also see a shift around m = 250 which could be explained background processing or allocation of resources.

```
ggarrange(all_m_plus_n_kmp, all_m_plus_n_naive,
      labels = c('KMP: Time/n+m', 'Naive: Time/n+m'),
      ncol = 2, nrow =1)
```


KMP: Time/n+m
median_time/(m + n)
6e−07
4e−07
2e−07
0
250
500
750
1000
m
RunName
bc
wc
Naive: Time/n+m
median_time/(m + n)
0.00015
0.00010
0.00005
0.00000
0
250
500
750
1000
m
RunName
bc
wc

“‘