R Notebook

Shad, Edin

december 1, 2016

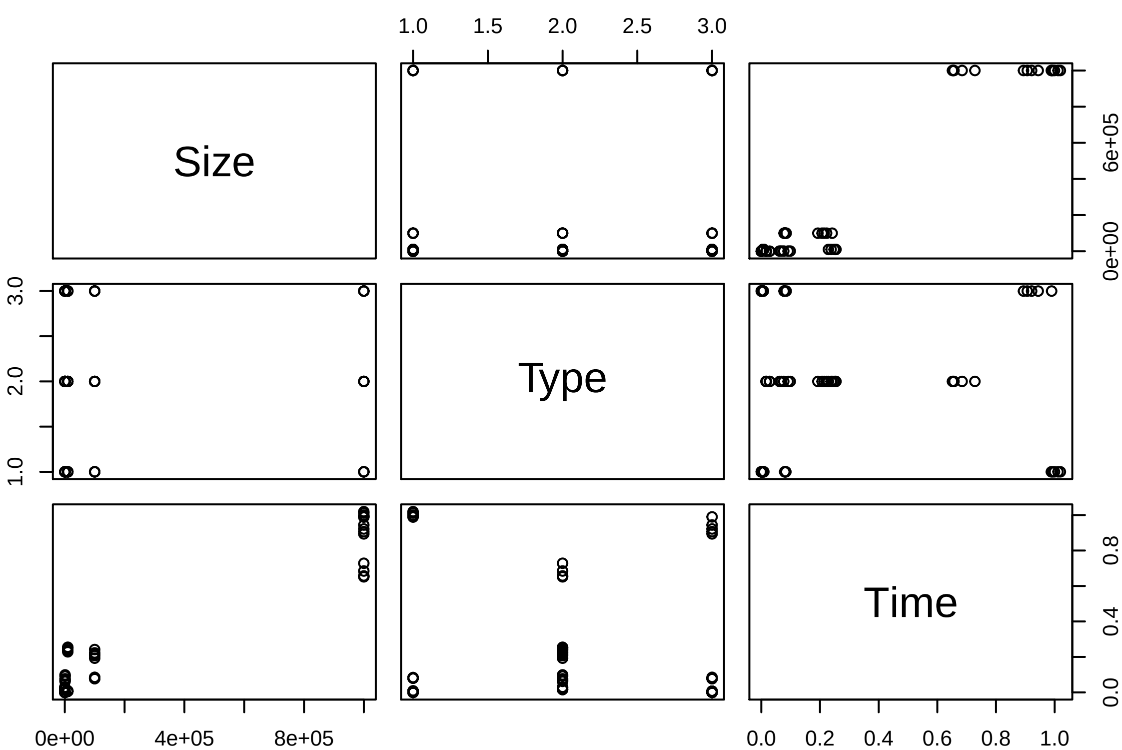
Scientific methodology and preformance evaluation

Winter School, ENS Lyon

We start with looking at the data provided on the web page: http://mescal.imag.fr/membres/arnaud.legrand/teaching/2013/M2R_EP_archive_quicksort.tgz

We won't focus on the particular implementation, although this is a very important part of the experiment, we leave it for the future work. We rather analyse the data, as we would do if somebody would give us implementation as black boxes which we query. One way to think about this, is that we do not compare in general which type of implementation works the best, but rather which out of three implementation is better for different sizes of arrays. So we turn towards determining when we should use one of these three algorithms to get faster running times.

```
data = read.csv("archive_quicksort/measurements.csv")
df = data.frame(data)
plot(df)
```



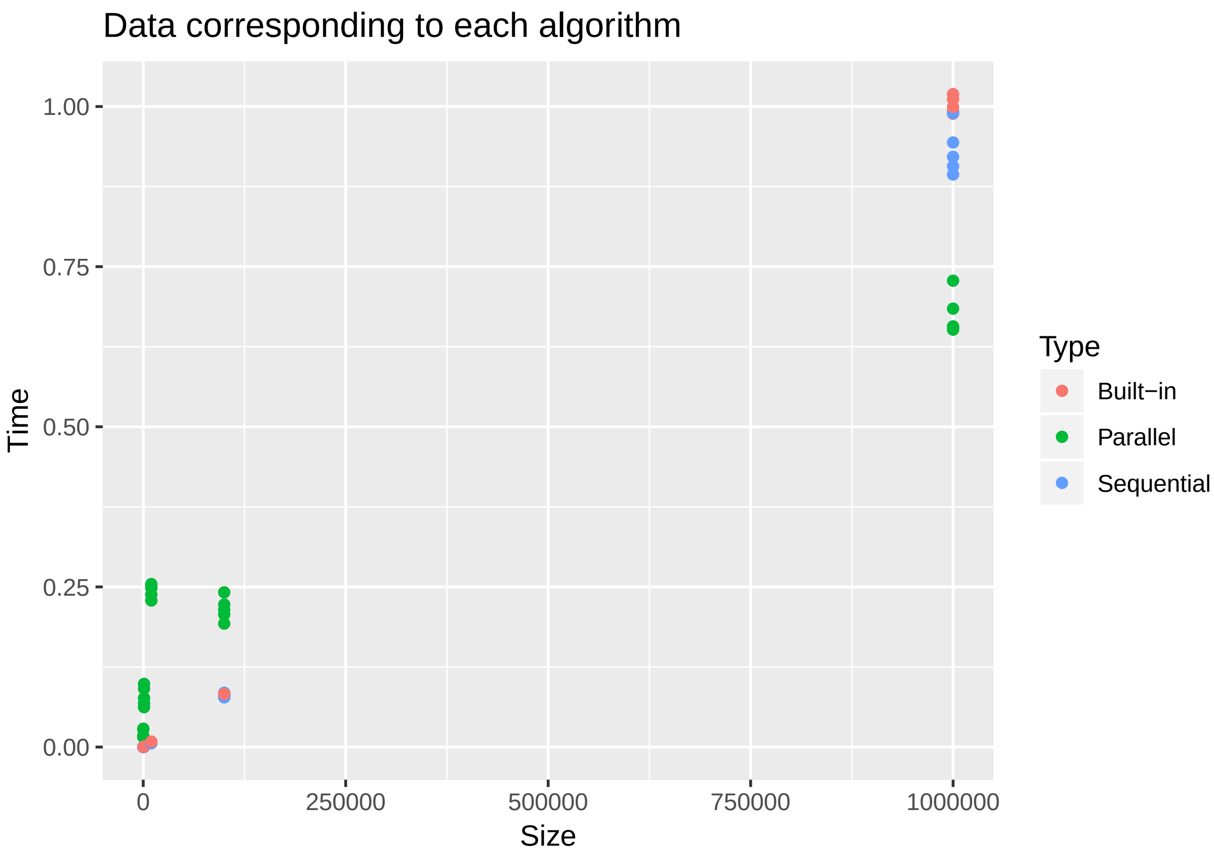

```
summary(df)
```

```
##       Size              Type          Time
##  Min.   :    100   Built-in  :25   Min.   :0.000039
##  1st Qu.:   1000   Parallel  :25   1st Qu.:0.000714
##  Median :  10000   Sequential:25   Median :0.076455
```

```
##  Mean   : 222220                     Mean   :0.223024
##  3rd Qu.: 100000                     3rd Qu.:0.239711
##  Max.   :1000000                     Max.   :1.019417
```

The above plot didn't say much. We observe that Size and Type are discrete variables and that Time should be assumed as continous. In order to see more, lets separate data according to the algorithm used.

```r
library(ggplot2)
ggplot(df, aes(x = Size, y = Time, colour = Type,  group = Type)) + geom_point() + ggtitle("Data corres
```

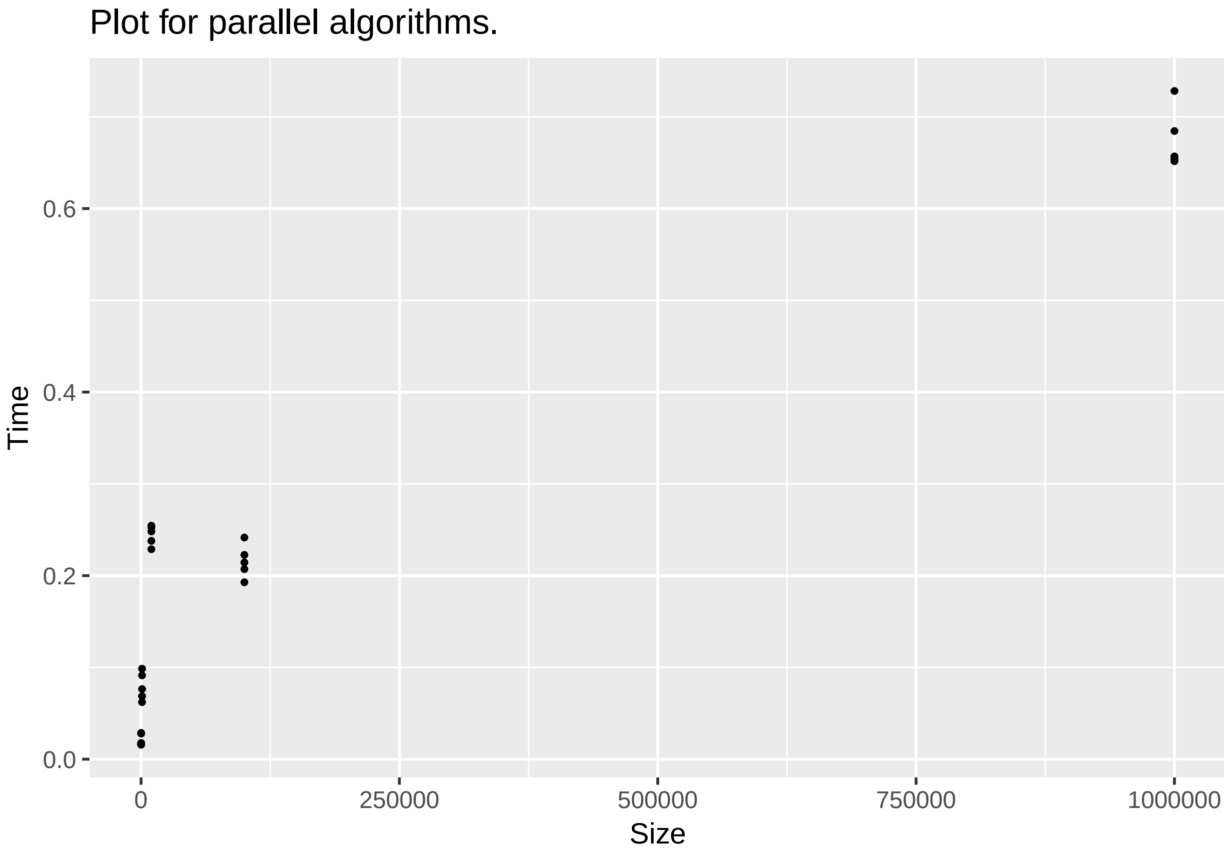


Obsearvtions: sizes for which the algorithms were tested are quite scarce and we can not conclude much. If we would have just this data, we could try to fit linear regression for each of the algorithms. Even if the results of linear regresion return good values in terms of R-value and p-values, we could be missing some strange behaviours since our data is so scarce. In any way we do the linear regresion for data of each algorithm. Before that, we plot again data of each algorithm separately just to see if they look linear.

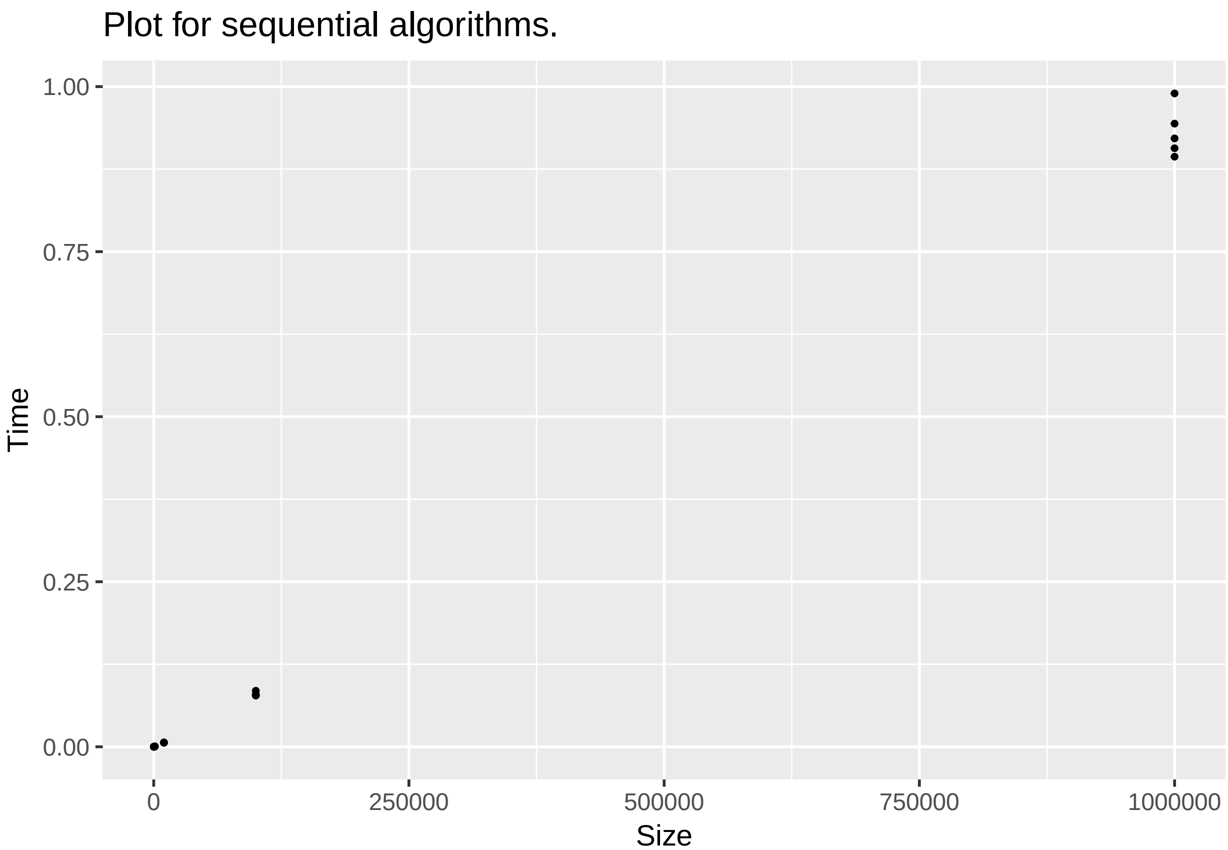
```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
para = df[df$Type == " Parallel",]
sequ = df[df$Type == " Sequential",]
builtIn = df[df$Type == " Built-in",]
```

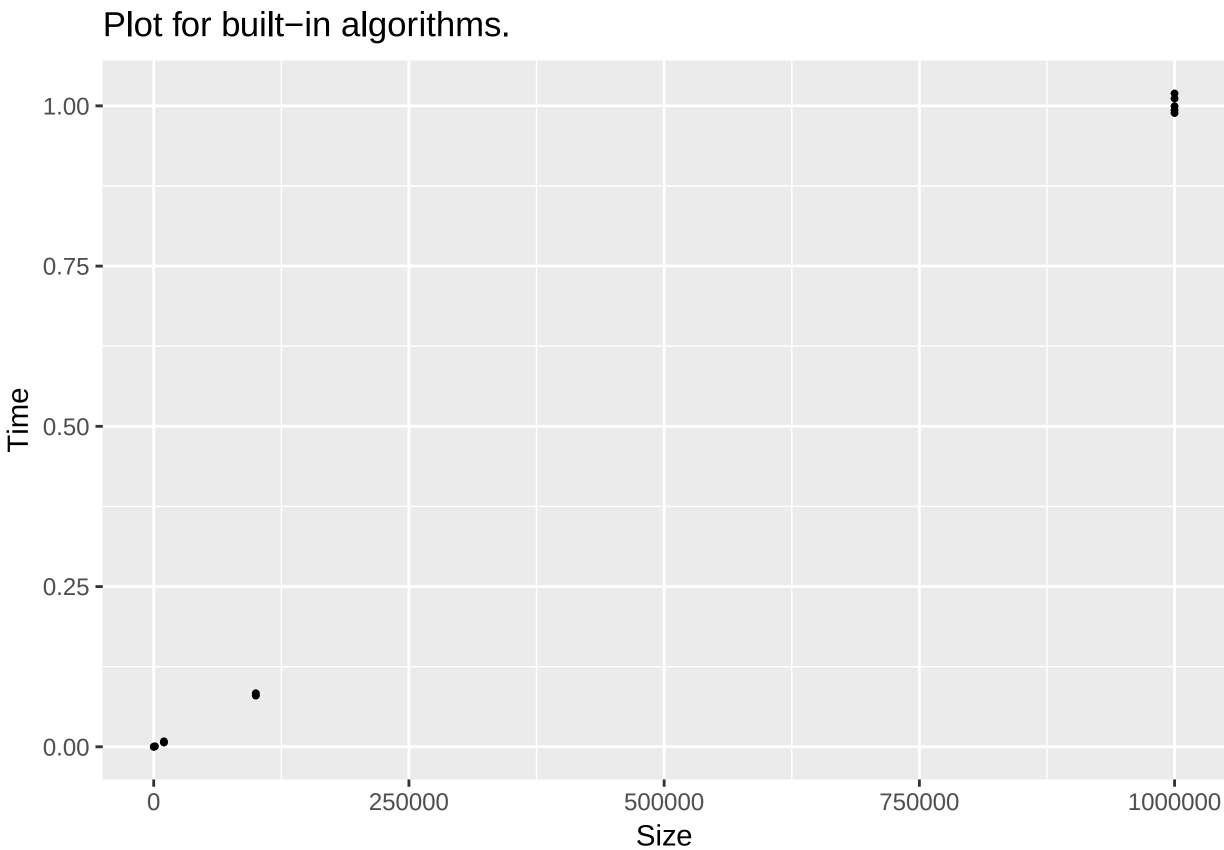
```
ggplot(para, aes(x = Size, y= Time)) + geom_point(size = 0.7) + ggtitle("Plot for parallel algorithms.")
```




```
ggplot(sequ, aes(x = Size, y= Time)) + geom_point(size = 0.7) + ggtitle("Plot for sequential algorithms
```




```
ggplot(builtIn, aes(x = Size, y= Time)) + geom_point(size = 0.7) + ggtitle("Plot for built-in algorithms
```

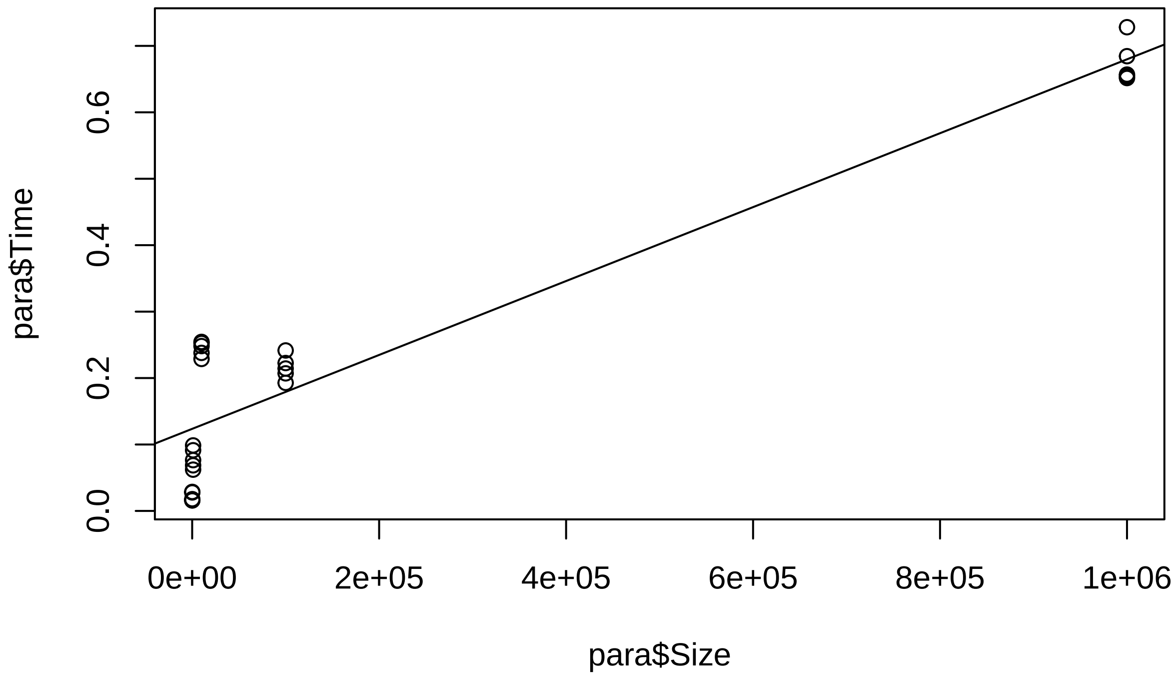


With exception of one outliner for parallel algorithm, we see that data resemble a linear function

```
regp = lm(data = para, Time ~ Size)
summary(regp)
```

```
##
## Call:
## lm(formula = Time ~ Size, data = para)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.10783 -0.05543 -0.02297  0.04831  0.12536
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.235e-01  1.812e-02   6.816 5.96e-07 ***
## Size        5.563e-07  4.032e-08  13.795 1.30e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.07877 on 23 degrees of freedom
## Multiple R-squared:  0.8922, Adjusted R-squared:  0.8875
## F-statistic: 190.3 on 1 and 23 DF,  p-value: 1.304e-12
```

```
plot(x = para$Size,y = para$Time)
abline(regp)
```



Results of linear fit for parallel algorithm seems quite strong for this data.

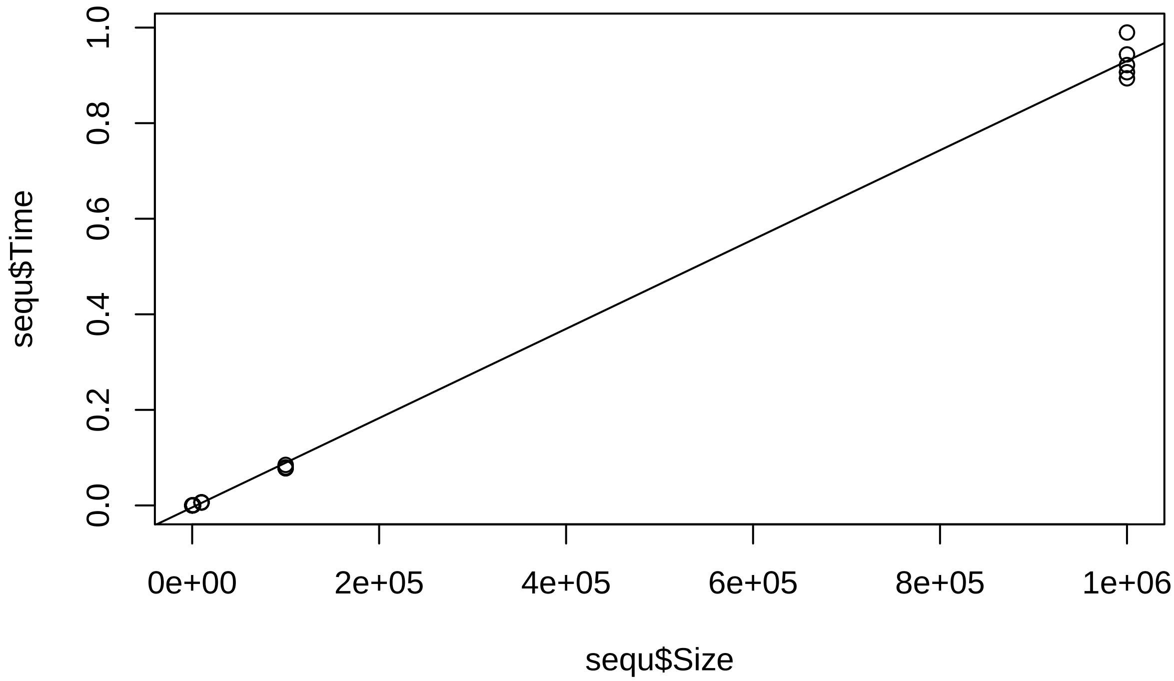
```r
regq = lm(data = sequ, Time ~ Size)
summary(regq)
```

```
##
## Call:
## lm(formula = Time ~ Size, data = sequ)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.036347 -0.008607  0.001317  0.004038  0.059532
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.092e-03  3.822e-03  -1.071    0.295
## Size         9.343e-07  8.504e-09 109.864   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01661 on 23 degrees of freedom
## Multiple R-squared:  0.9981, Adjusted R-squared:  0.998
## F-statistic: 1.207e+04 on 1 and 23 DF,  p-value: < 2.2e-16
```

```r
par(mfrow~c(1,1))
```

```
## NULL
```

```r
plot(x = sequ$Size,y = sequ$Time)
abline(regq)
```



Linear fit for the sequential model

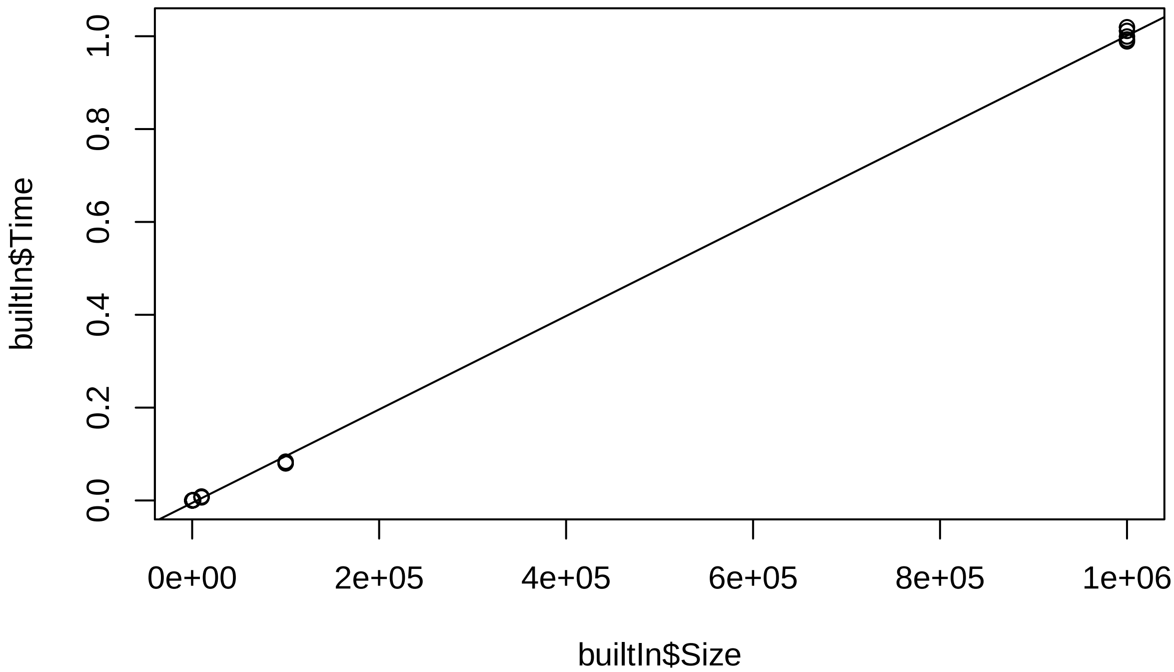
```r
regb = lm(data = builtIn, Time ~ Size)
summary(regb)
```

```
##
## Call:
## lm(formula = Time ~ Size, data = builtIn)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.015615 -0.007997  0.003712  0.005146  0.018362
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.206e-03  2.107e-03   -2.47   0.0213 *
## Size         1.006e-06  4.689e-09  214.62   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.009159 on 23 degrees of freedom
## Multiple R-squared:  0.9995, Adjusted R-squared:  0.9995
## F-statistic: 4.606e+04 on 1 and 23 DF,  p-value: < 2.2e-16
```

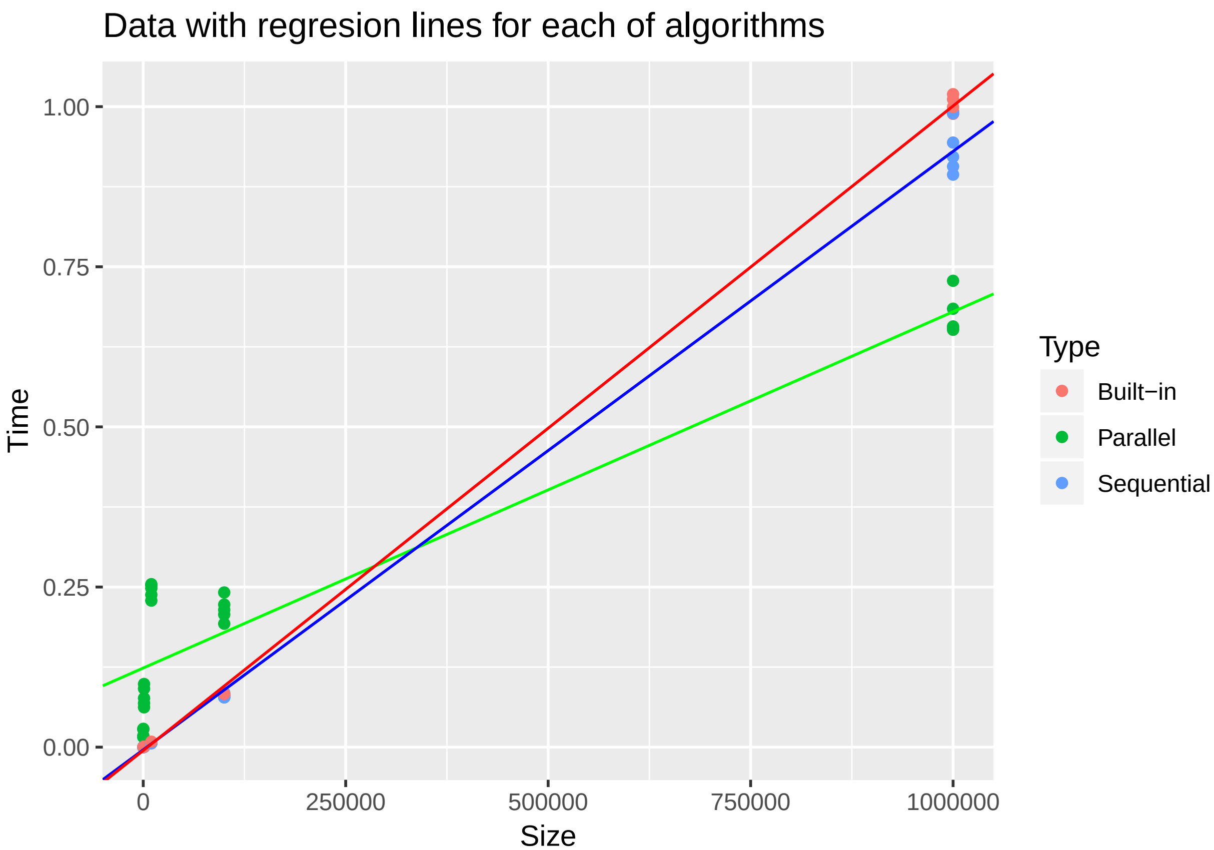
```r
par(mfrow~c(1,1))
```

```
## NULL
```

```r
plot(x = builtIn$Size,y = builtIn$Time)
abline(regb)
```




```
ggplot(df, aes(x = Size, y = Time, colour = Type,  group = Type)) + geom_point() + ggtitle("Data with re
```

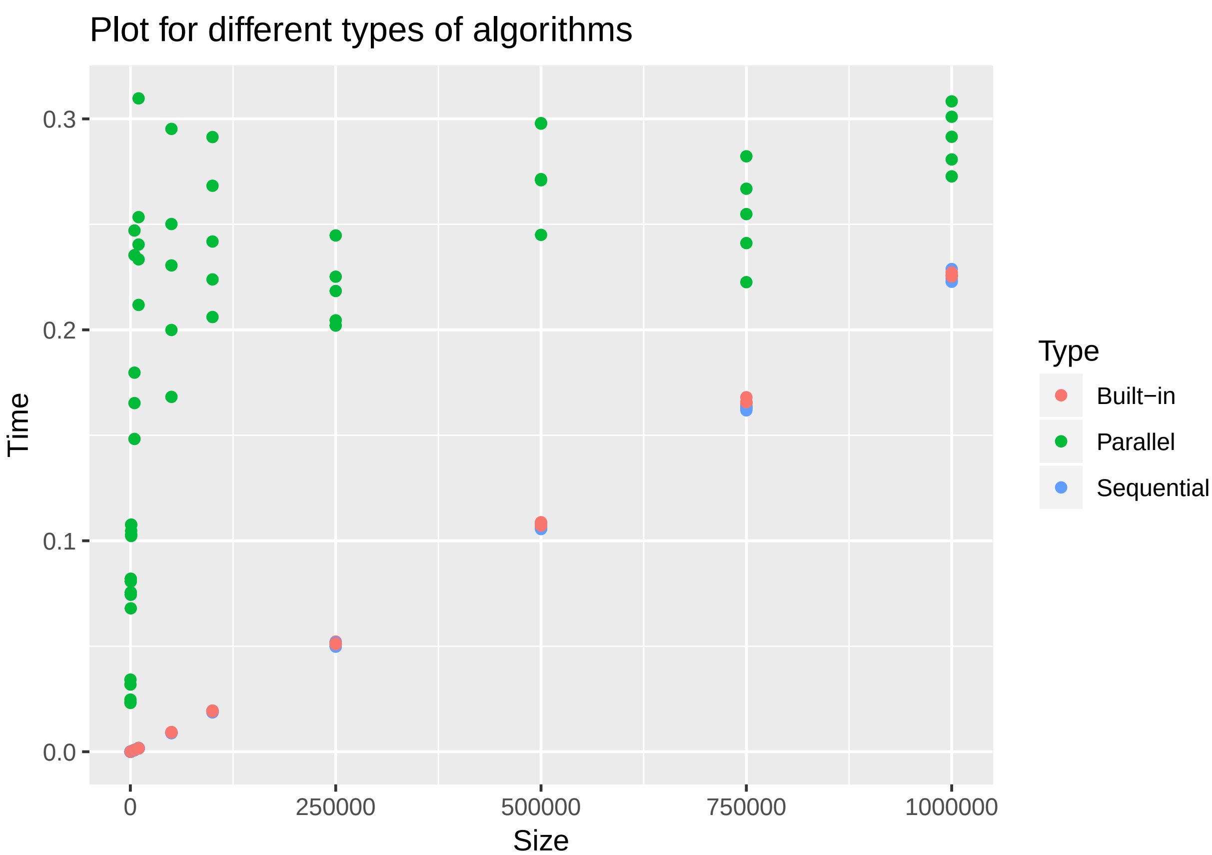


'Under above assumptions, the parallel algorithm seems to outrun the other two algorithms for the arrays of size bigger than 250000 and that sequential and built-in algorithm are alsmost the same. From this data it is not reliable to say that sequential is overall better than built-in since the data set is small, and not well calibrated. One more problem of the current data is that we don't know how were they measured, on how many machines and what kind of machines. In order to improve this, we generate new data sets using providede “black-boxes” and extend the data for sizes up to milion. It is worth noting that the R-squared values are very high. Especially for the sequential and the built-in regression we obtain values of around 99% but for the parallel a value around 89. This could suggest that the parallel implementation is not linear.

```r
dataEdin = read.csv("MyData/measurements.csv")
dE = data.frame(dataEdin)
summary(dE)
```

```
##       Size                Type         Time
##  Min.   :    100   Built-in  :55   Min.   :0.000009
##  1st Qu.:   1000   Parallel  :55   1st Qu.:0.001565
##  Median :  50000   Sequential:55   Median :0.051495
##  Mean   : 242418                   Mean   :0.100817
##  3rd Qu.: 500000                   3rd Qu.:0.211804
##  Max.   :1000000                   Max.   :0.309712
```

```r
ggplot(dE, aes(x = Size, y = Time, colour = Type,  group = Type)) + geom_point() + ggtitle("Plot for di
```



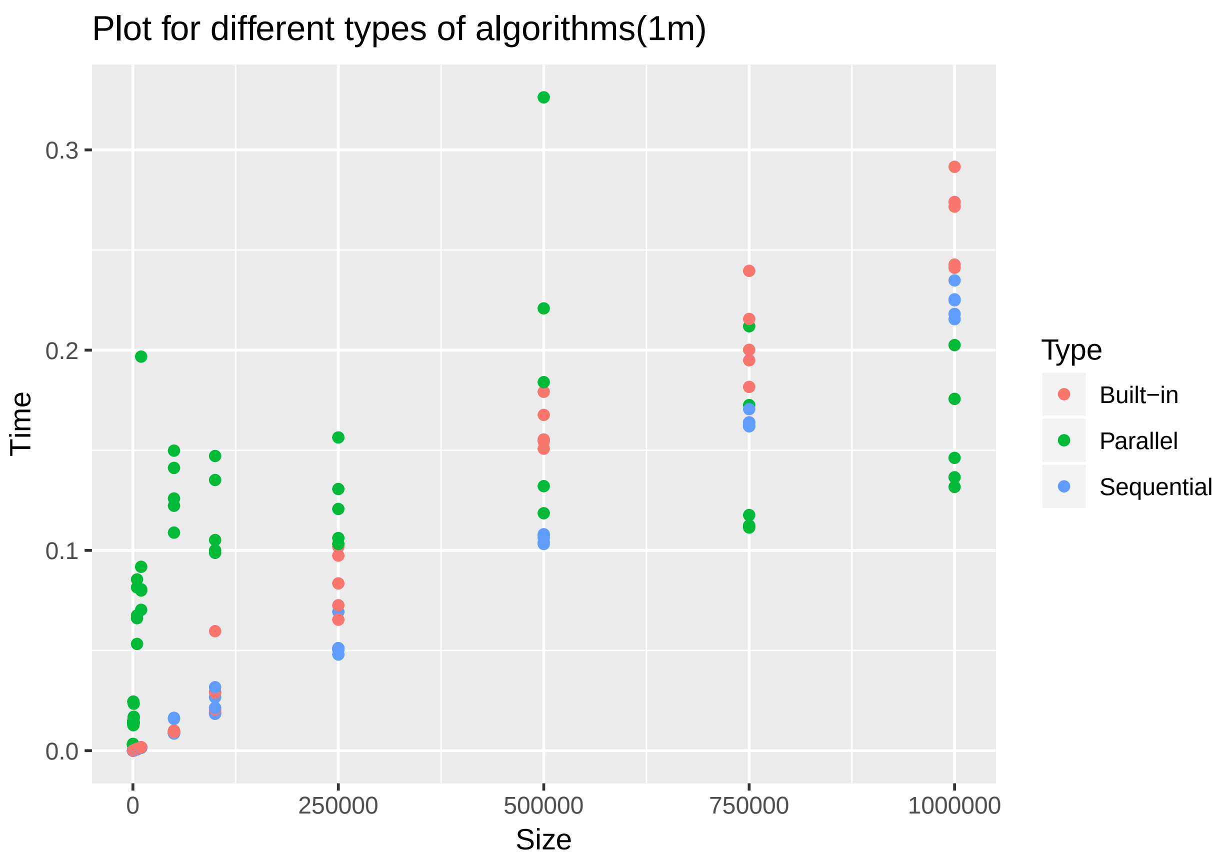
We observe that the paralel algorithm shows very slow running time for smaller arrays. But it seems that it will be better after some big enough size. That is way we for the moment stop here, and try to get more data for the biggest sizes of arrays. But first, let us see what happens with the same procedure on different arhitecture.

```r
dataShad = read.csv("ShadMeasurements.csv")
dS = data.frame(dataShad)
summary(dS)
```

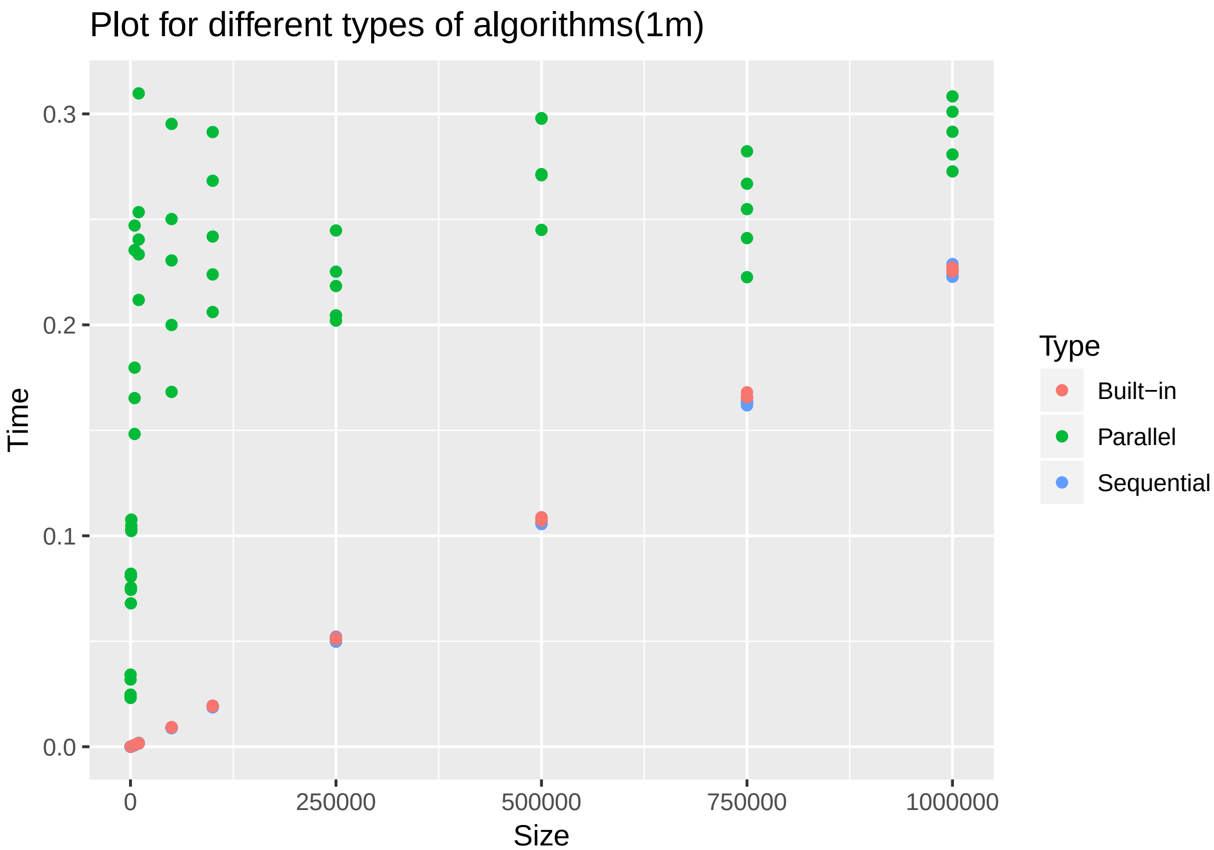
```
##       Size                Type         Time
##  Min.   :    100   Built-in  :55   Min.   :0.000007
```

```
##  1st Qu.:   1000    Parallel  :55   1st Qu.:0.001457
##  Median :  50000    Sequential:55   Median :0.029517
##  Mean   : 242418                    Mean   :0.073540
##  3rd Qu.: 500000                    3rd Qu.:0.131701
##  Max.   :1000000                    Max.   :0.326254
```

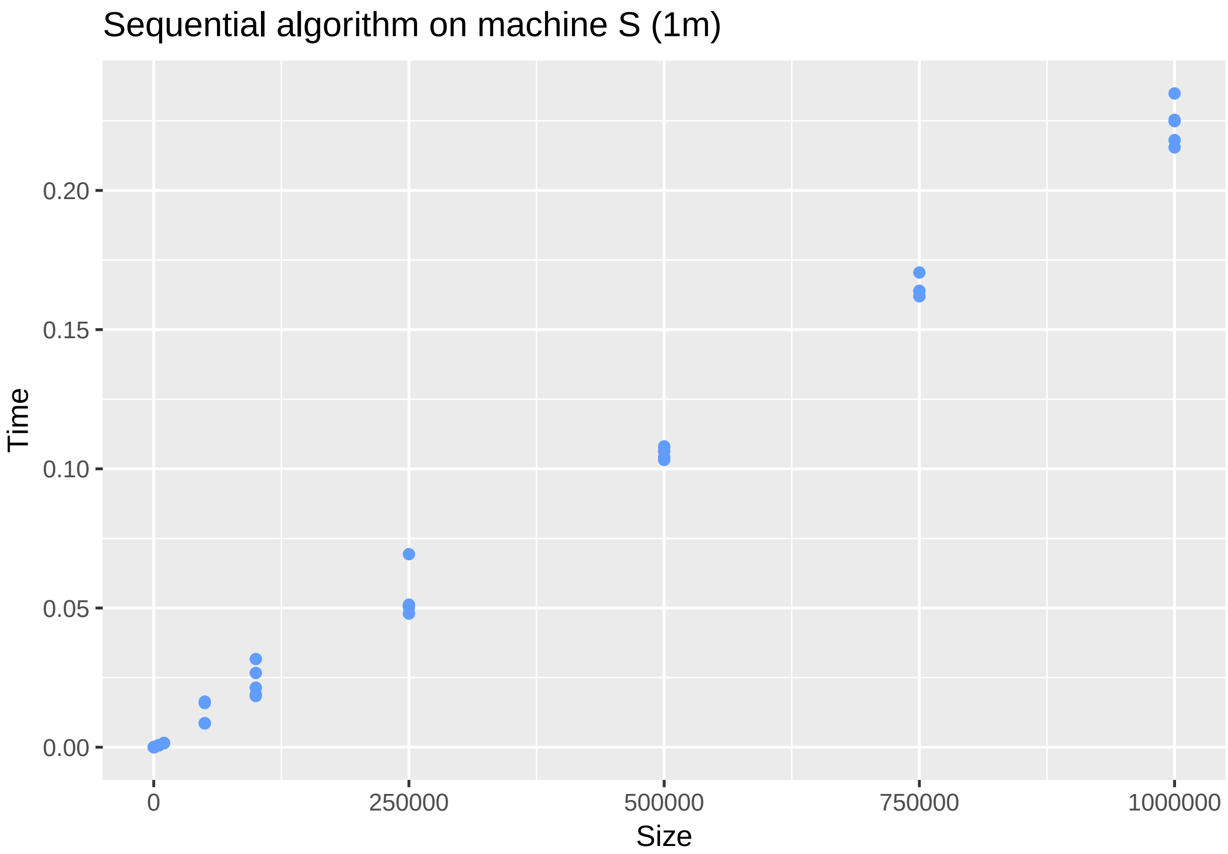
```
ggplot(dS, aes(x = Size, y = Time, colour = Type,  group = Type)) + geom_point() + ggtitle("Plot for di
```



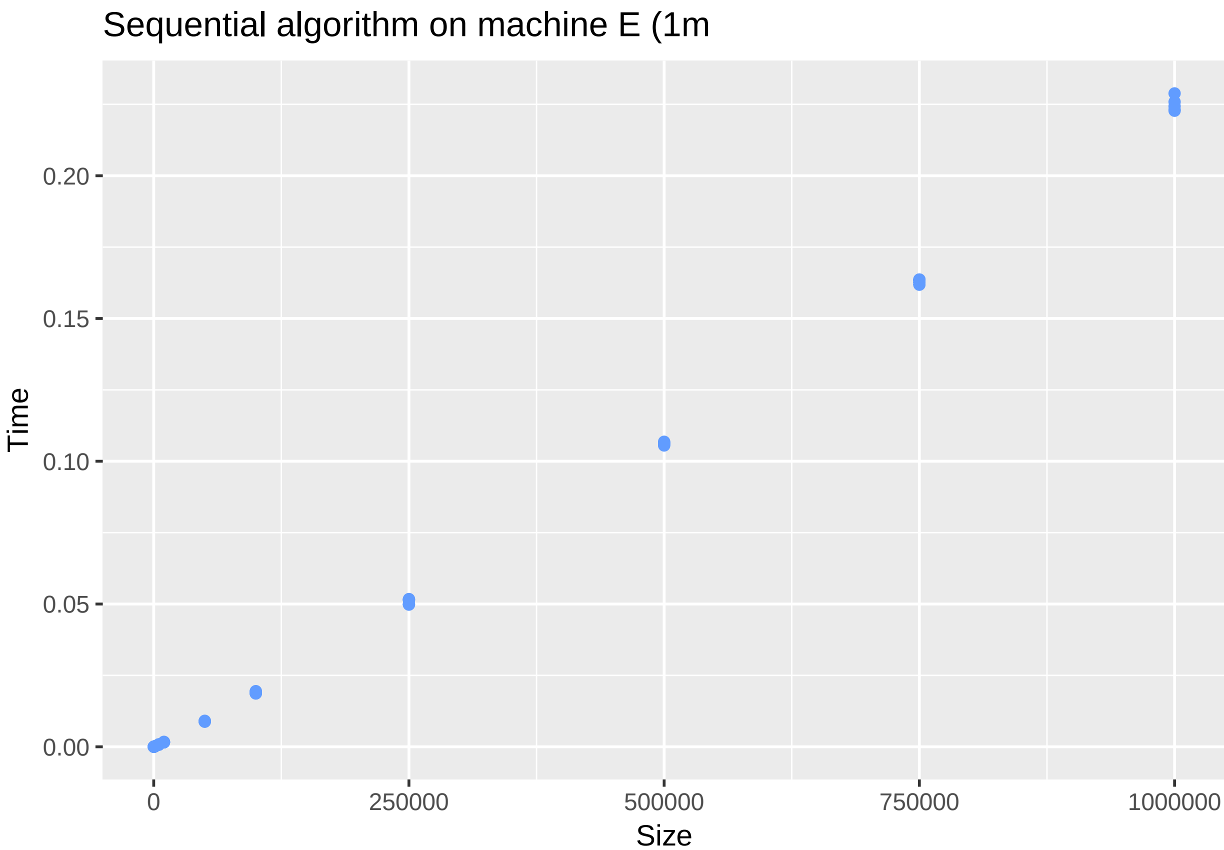

```
ggplot(dE, aes(x = Size, y = Time, colour = Type,  group = Type)) + geom_point() + ggtitle("Plot for di
```



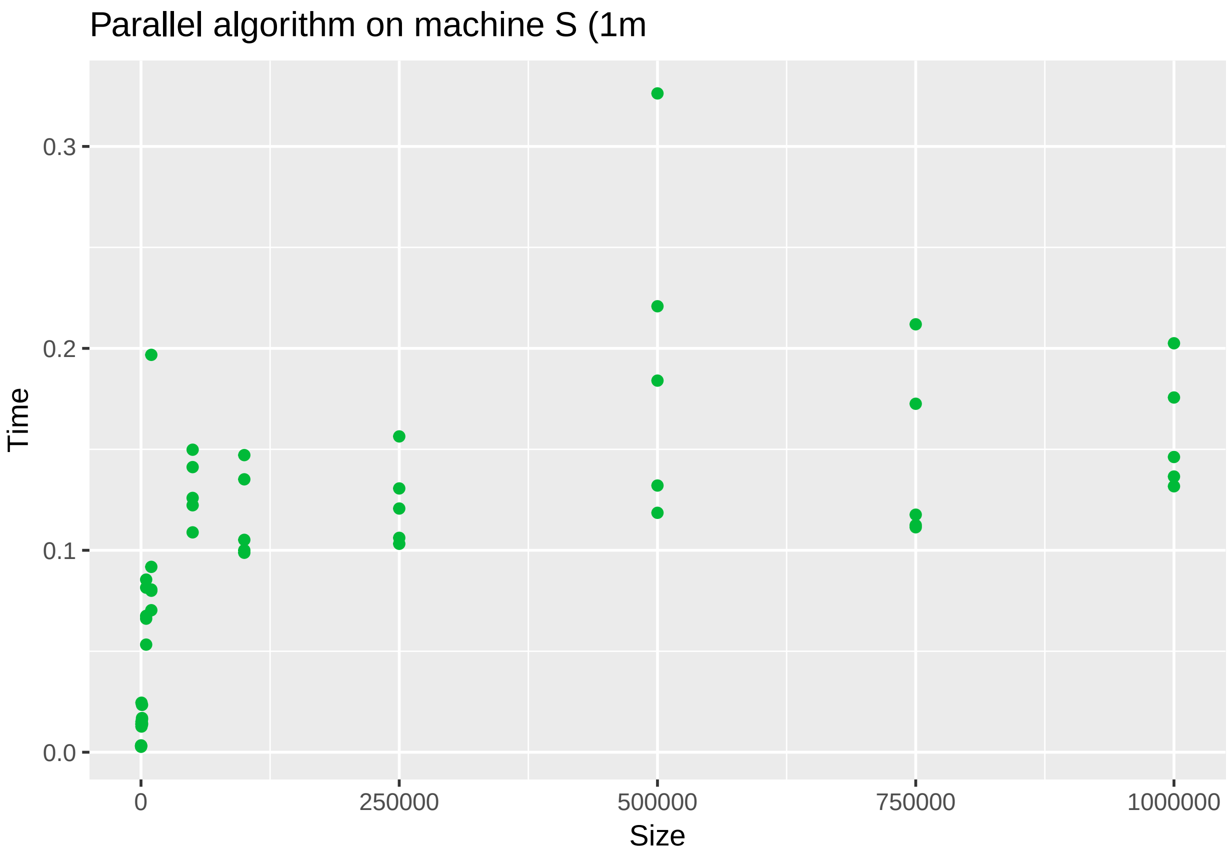

```
ggplot(dS[dS$Type == " Sequential",],  aes(x = Size, y = Time, colour = Type,  group = Type)) + geom_poi
```



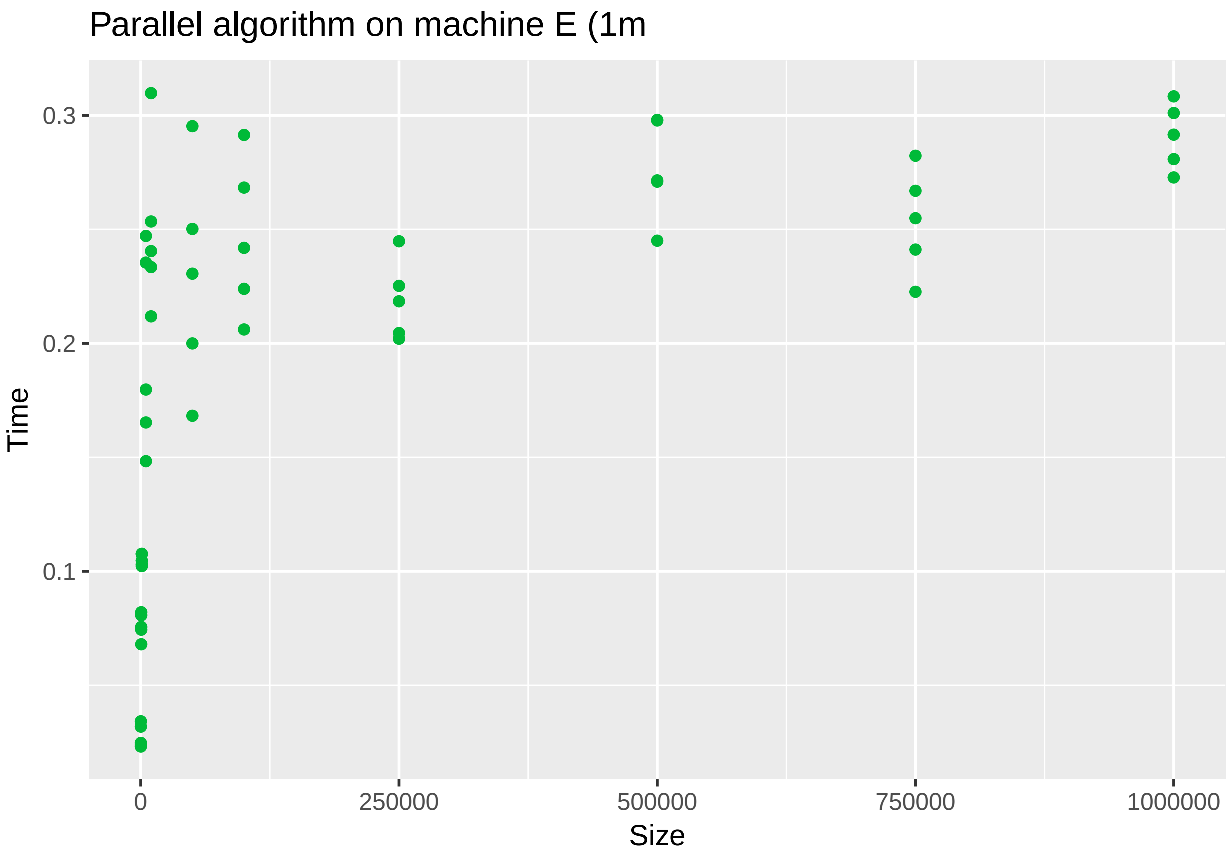

```
ggplot(dE[dE$Type == " Sequential",] , aes(x = Size, y = Time, colour = Type,  group = Type)) + geom_poi
```



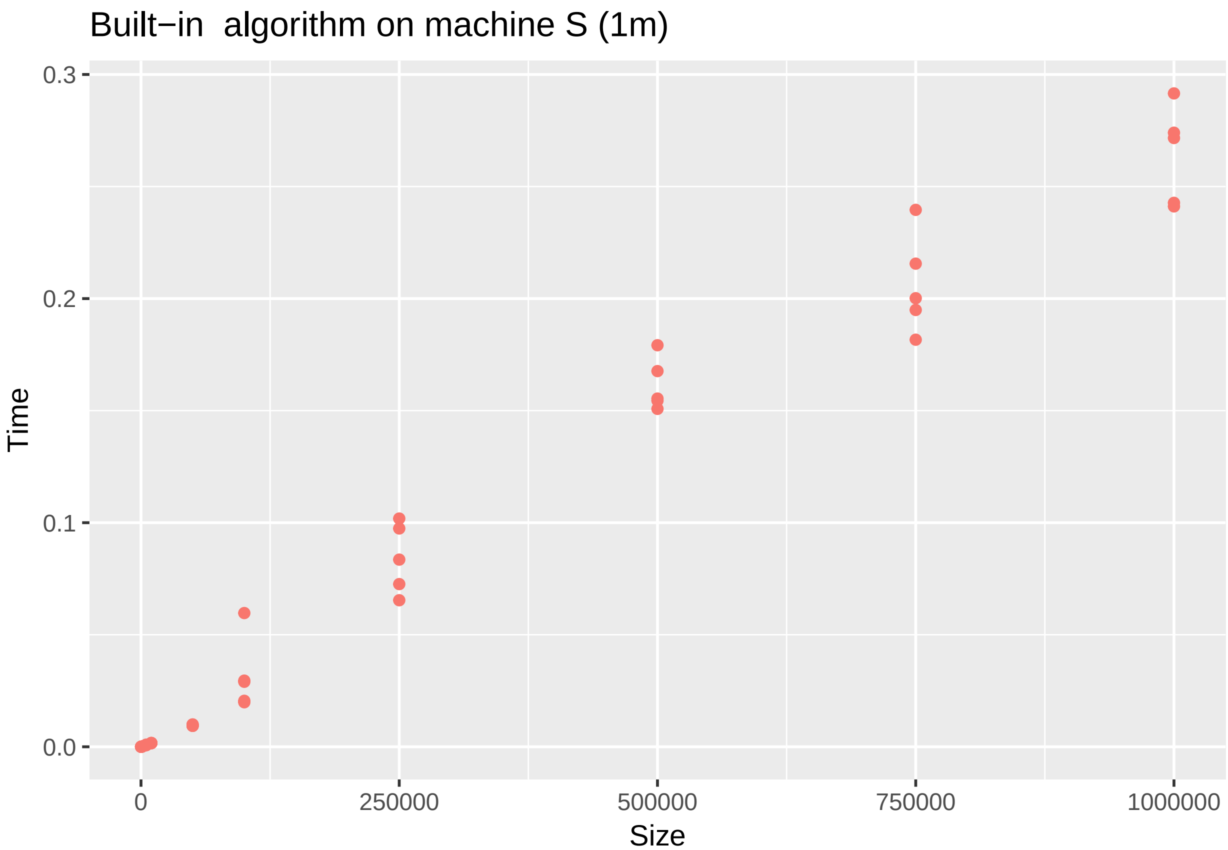

```
ggplot(dS[dS$Type == " Parallel",],  aes(x = Size, y = Time, colour = Type,  group = Type)) + geom_poin
```



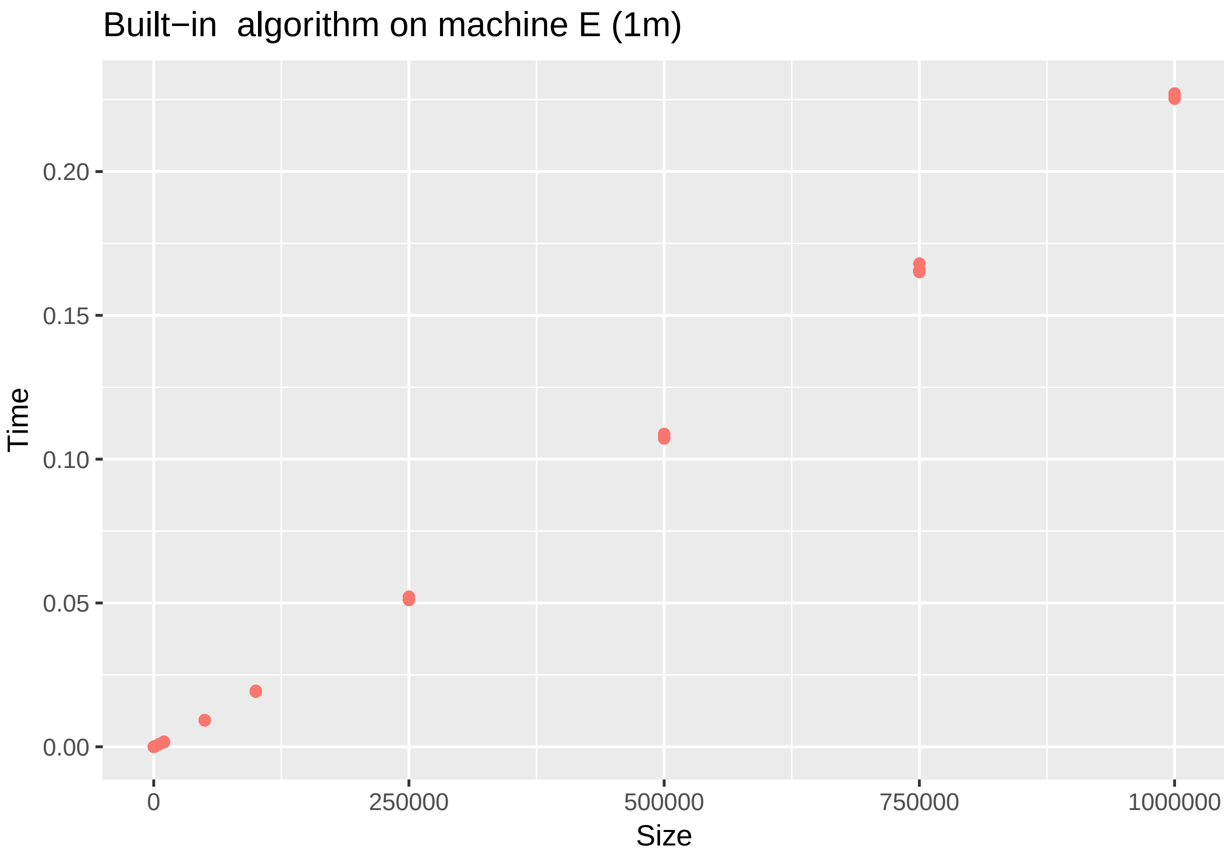

```
ggplot(dE[dE$Type == " Parallel",] , aes(x = Size, y = Time, colour = Type,  group = Type)) + geom_poin
```




```
ggplot(dS[dS$Type == " Built-in",],  aes(x = Size, y = Time, colour = Type,  group = Type)) + geom_poin
```




```
ggplot(dE[dE$Type == " Built-in",] , aes(x = Size, y = Time, colour = Type,  group = Type)) + geom_poin
```



We observe that in both cases the parallel algorithm doesn't follow linear scaling, while both sequential and built-in algorithm show linear increase of execution time with the size. We also decide that for a better estimation we need to increase the maximum size. We move the maximum up to two milion.

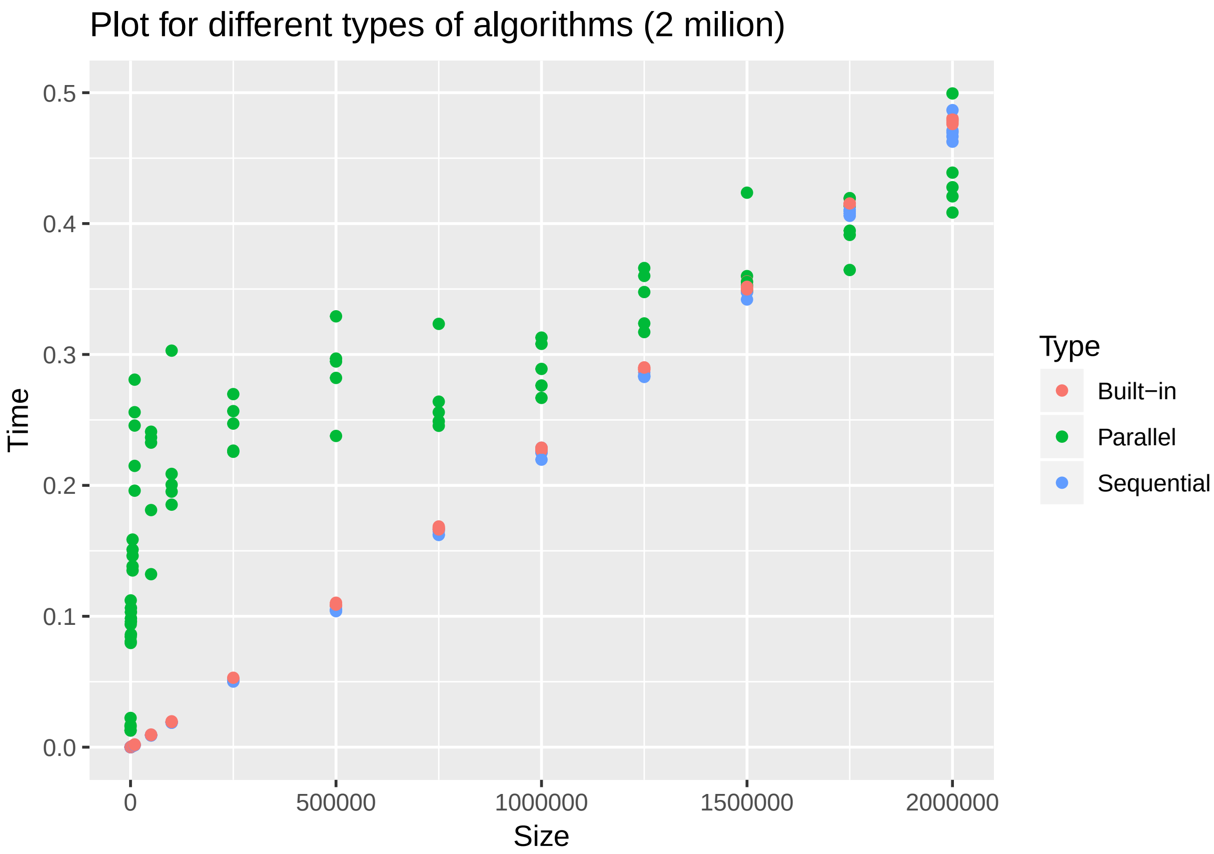
???? Guided by known expected running time of quicksort we prose the logarithmic-linear model. To get some more information and intuition about the performances of each implementation of quicksort we extend the measurements for sizes up to two milion.

```
dataEdin2 = read.csv("MyData2milion/measurements.csv")
dE2 = data.frame(dataEdin2)
summary(dE2)
```

```
##       Size                   Type         Time
##  Min.   :    100   Built-in  :75   Min.   :0.000009
##  1st Qu.:   5000   Parallel  :75   1st Qu.:0.009231
##  Median : 250000   Sequential:75   Median :0.162244
##  Mean   : 611107                   Mean   :0.174722
##  3rd Qu.:1250000                   3rd Qu.:0.294626
##  Max.   :2000000                   Max.   :0.499404
```

We again plot it, to get more intution.

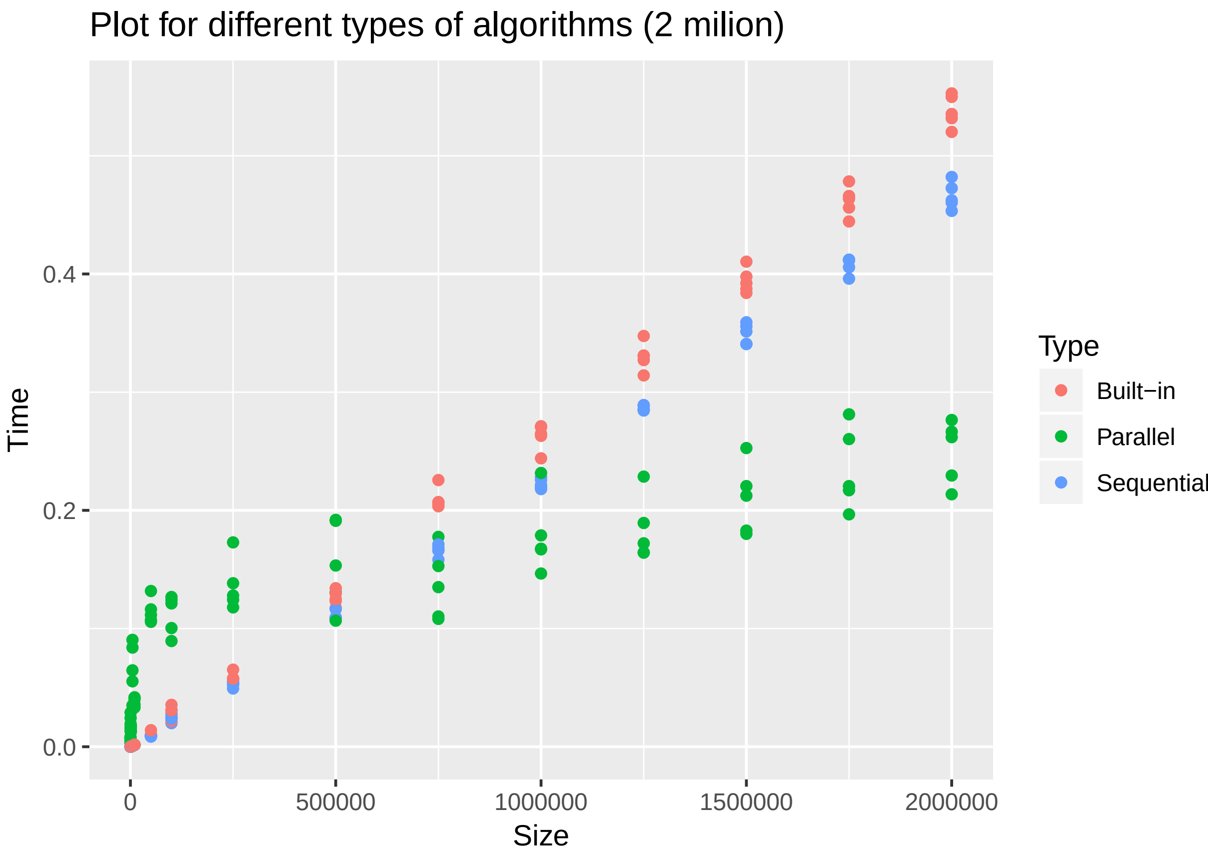
```
ggplot(dE2, aes(x = Size, y = Time, colour = Type,  group = Type)) + geom_point() + ggtitle("Plot for di
```




```r
dataShad2 = read.csv("ShadMeasurements2.csv")
dS2 = data.frame(dataShad2)
summary(dS2)
```

```
##       Size                  Type         Time
##  Min.   :    100   Built-in  :75   Min.   :0.000008
##  1st Qu.:   5000   Parallel  :75   1st Qu.:0.008685
##  Median : 250000   Sequential:75   Median :0.108788
##  Mean   : 611107                   Mean   :0.141720
##  3rd Qu.:1250000                   3rd Qu.:0.225626
##  Max.   :2000000                   Max.   :0.552768
```

```r
ggplot(dS2, aes(x = Size, y = Time, colour = Type,  group = Type)) + geom_point() + ggtitle("Plot for di
```



We see again that the Parallel algorithm does not really follow a linear model but we try it noonetheles.

```
paraS2 = dS2[dS2$Type == " Parallel",]
regpS2 <- lm(data=paraS2, Time~Size)
summary(regpS2)
```

```
##
## Call:
## lm(formula = Time ~ Size, data = paraS2)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.058312 -0.032083 -0.002933  0.037808  0.085910
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 6.149e-02  6.094e-03   10.09 1.74e-15 ***
## Size        1.020e-07  6.607e-09   15.43  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.03953 on 73 degrees of freedom
## Multiple R-squared:  0.7654, Adjusted R-squared:  0.7622
## F-statistic: 238.2 on 1 and 73 DF,  p-value: < 2.2e-16
```

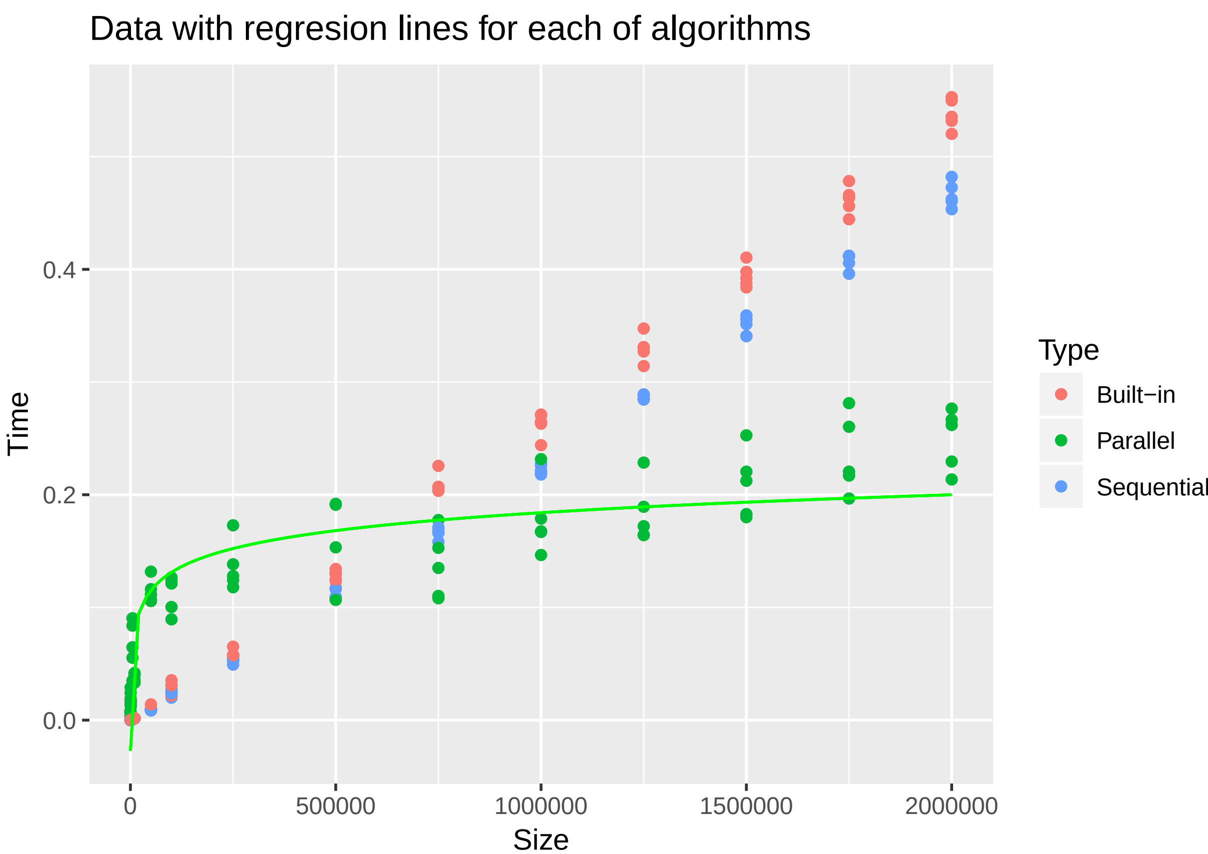
```
regpS2log <- lm(data=paraS2, Time~log(Size))
summary(regpS2log)
```

```
##
```

```
## Call:
## lm(formula = Time ~ log(Size), data = paraS2)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.069313 -0.024475 -0.005256  0.022168  0.084333
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.132906   0.014009  -9.487 2.28e-14 ***
## log(Size)    0.022944   0.001204  19.060  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.03339 on 73 degrees of freedom
## Multiple R-squared:  0.8327, Adjusted R-squared:  0.8304
## F-statistic: 363.3 on 1 and 73 DF,  p-value: < 2.2e-16
```

By doing the a linear regression anyway we find that the adjusted R-squared value is around 0.76. We do a logarithmic regression and retrieve an adjusted R-squared value of 0.83 which could suggest that the model is better. However we also note that the std. error is larger for the logarithmic model compared to the linear model.

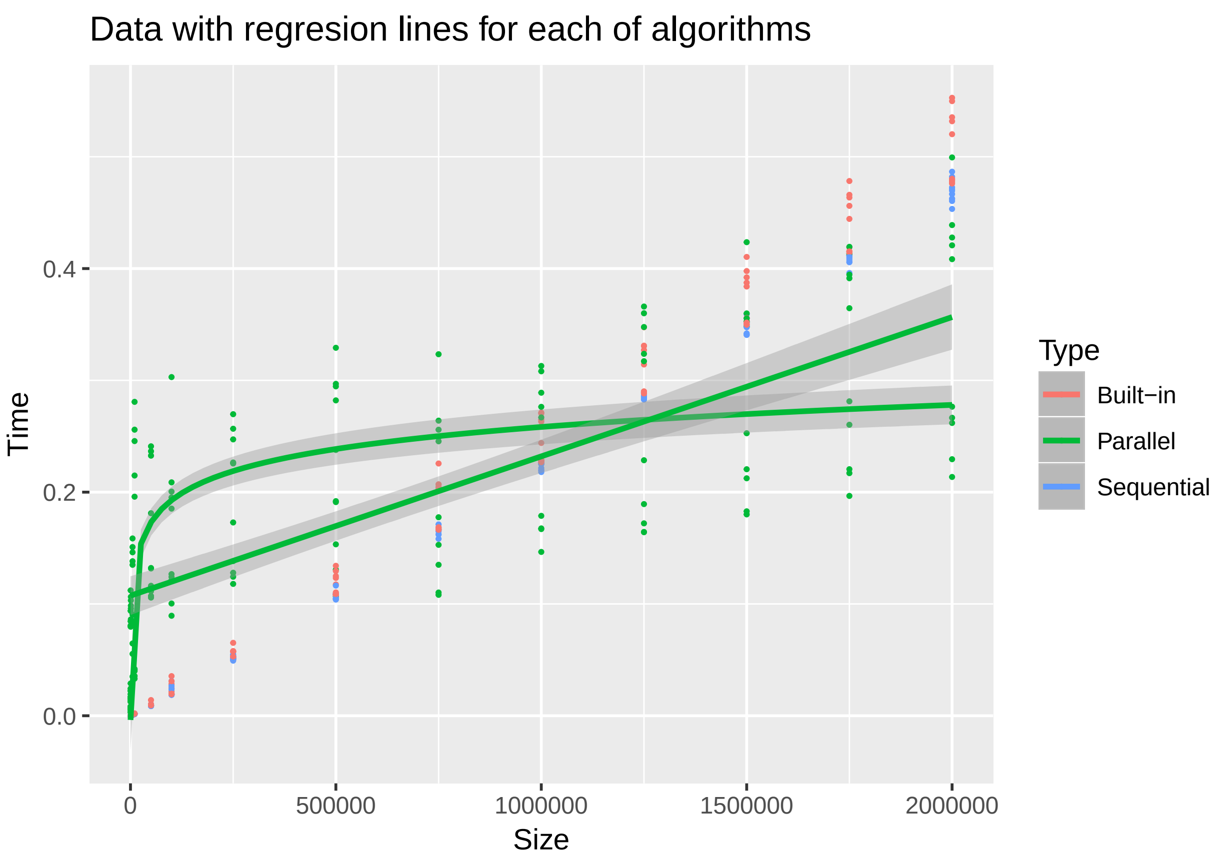
```
para2 = dS2[dS2$Type == "Parallel",]
test <- function(x) {coef(regpS2log)["log(Size)"]*log(x) + coef(regpS2log)["(Intercept)"]}
ggplot(dS2, aes(x = Size, y = Time, colour = Type,  group = Type)) + geom_point() + ggtitle("Data with :
```



```r
test1 <- rbind(dS2, dE2)
paraComb = test1[test1$Type == " Parallel",]
regComb <- lm(data=paraComb, Time~log(Size))
regLin <- lm(data=paraComb, Time~Size)
summary(regLin)
```

```
##
## Call:
## lm(formula = Time ~ Size, data = paraComb)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.14302 -0.07508 -0.00723  0.06553  0.18305
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.074e-01  8.772e-03   12.25   <2e-16 ***
## Size        1.246e-07  9.510e-09   13.10   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.08048 on 148 degrees of freedom
## Multiple R-squared:  0.537,  Adjusted R-squared:  0.5338
## F-statistic: 171.6 on 1 and 148 DF,  p-value: < 2.2e-16
```

```r
test <- function(x) {coef(regComb)["log(Size)"]*log(x) + coef(regComb)["(Intercept)"]}
ggplot(test1, aes(x = Size, y = Time, colour = Type,  group = Type)) + geom_point(size=0.4) + ggtitle("
```



```r
summary(regComb)
```

```
##
## Call:
## lm(formula = Time ~ log(Size), data = paraComb)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.141973 -0.060549  0.004071  0.050589  0.221361
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.134809   0.022144  -6.088 9.38e-09 ***
## log(Size)    0.028456   0.001903  14.955  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.07463 on 148 degrees of freedom
## Multiple R-squared:  0.6018, Adjusted R-squared:  0.5991
## F-statistic: 223.6 on 1 and 148 DF,  p-value: < 2.2e-16
```

We combined the data from the two computers but it does not really make sense, as expected we obtain low R-squared values and p-values.

```r
regpS2test <- lm(data=paraS2, Time~Size)
summary(regpS2test)
```

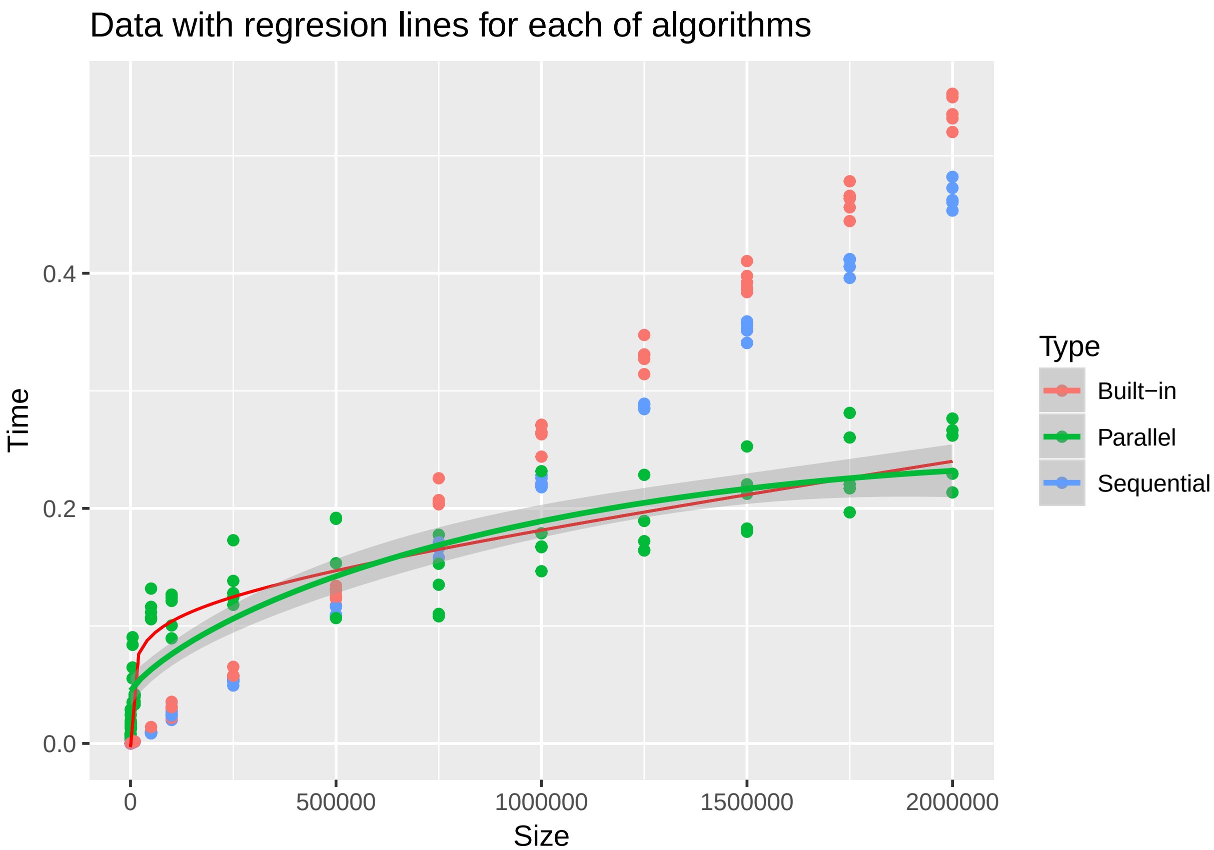
```
##
## Call:
## lm(formula = Time ~ Size, data = paraS2)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.058312 -0.032083 -0.002933  0.037808  0.085910
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 6.149e-02  6.094e-03   10.09 1.74e-15 ***
## Size        1.020e-07  6.607e-09   15.43  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.03953 on 73 degrees of freedom
## Multiple R-squared:  0.7654, Adjusted R-squared:  0.7622
## F-statistic: 238.2 on 1 and 73 DF,  p-value: < 2.2e-16
```

```r
regpS2t <- lm(data=paraS2, Time~log(Size)+Size)
summary(regpS2t)
```

```
##
## Call:
## lm(formula = Time ~ log(Size) + Size, data = paraS2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.05694 -0.01822 -0.00141  0.01667  0.05525
```

```
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -7.114e-02  1.411e-02  -5.043 3.31e-06 ***
## log(Size)    1.478e-02  1.507e-03   9.807 6.69e-15 ***
## Size         4.837e-08  6.986e-09   6.924 1.53e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02605 on 72 degrees of freedom
## Multiple R-squared:  0.8996, Adjusted R-squared:  0.8968
## F-statistic: 322.4 on 2 and 72 DF,  p-value: < 2.2e-16
```

```
para2 = dS2[dS2$Type == "Parallel",]
test <- function(x) {coef(regpS2t)["log(Size)"]*log(x) + coef(regpS2t)["(Intercept)"]+coef(regpS2t)["Si
ggplot(dS2, aes(x = Size, y = Time, colour = Type,  group = Type)) + geom_point() + ggtitle("Data with r
```




```
regpS23 <- lm(data=paraS2, Time~I(log(Size)*Size)+Size)
summary(regpS23)
```

```
##
## Call:
## lm(formula = Time ~ I(log(Size) * Size) + Size, data = paraS2)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.060640 -0.029721 -0.008666  0.033341  0.068815
##
```

```
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)    
## (Intercept)          4.477e-02  6.330e-03   7.073 8.10e-10 ***
## I(log(Size) * Size) -7.317e-08  1.501e-08  -4.876 6.26e-06 ***
## Size                 1.155e-06  2.161e-07   5.346 1.01e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.03451 on 72 degrees of freedom
## Multiple R-squared:  0.8236, Adjusted R-squared:  0.8187 
## F-statistic: 168.1 on 2 and 72 DF,  p-value: < 2.2e-16
```

The model xlog(x)+x provides a good fit for our data. Furthermore it also makes sense because we expect the model to behave like a O(xlog(x)) function where xlog(x) is the leading term. However we might have some other term who also dictates the running time of the function and changes the curve. For our model we chose this other term to be x which provides us with the green curve on the above plot. The model logx + x obtains better values but the model doesn't make sense and thus could just be a good model for this data (overfitting).