test

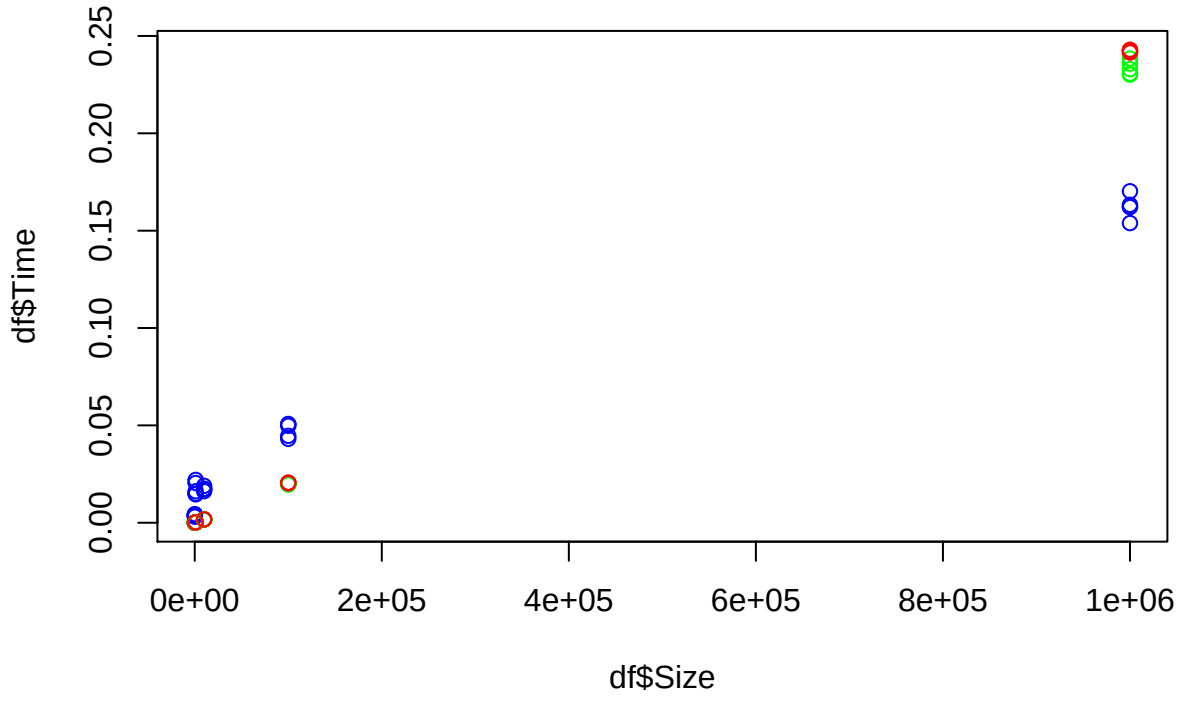
Experiments in class

We did these tests in class to learn some experimental and plotting methods

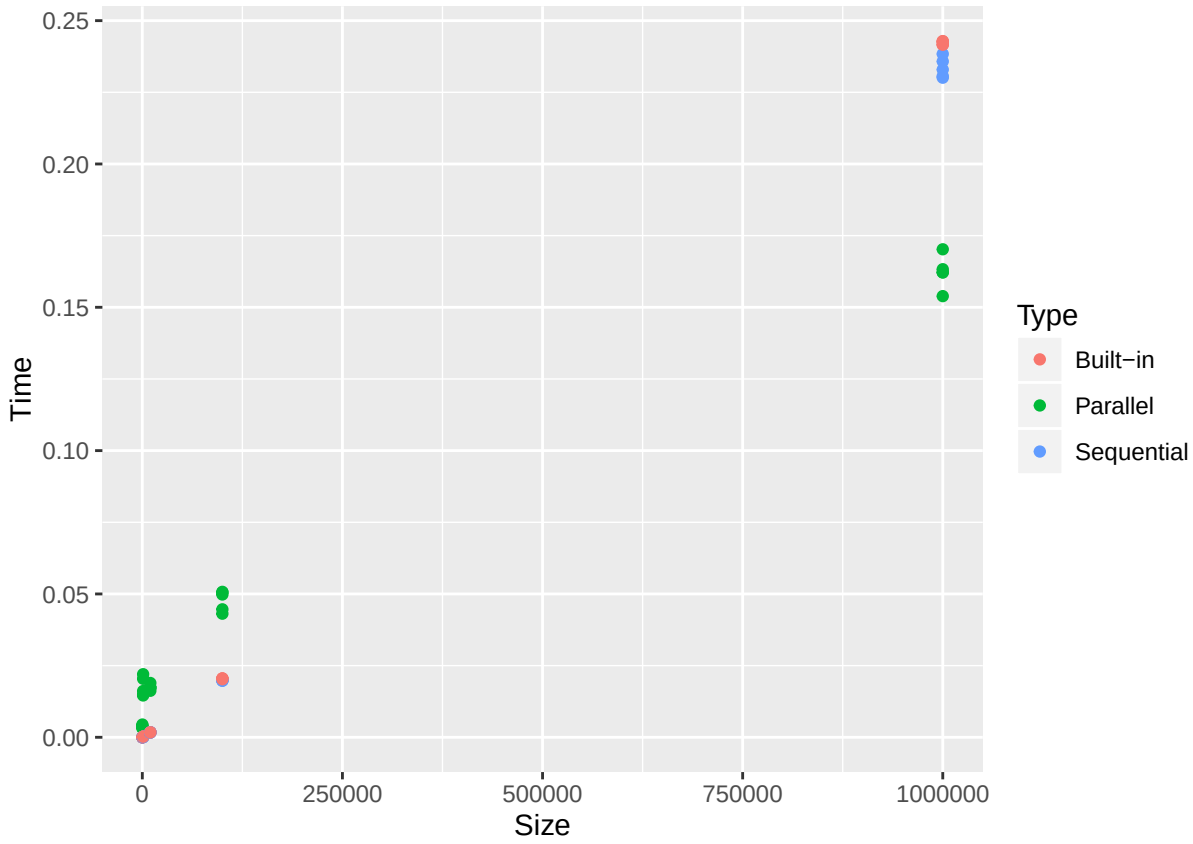
```
#install.packages("ggplot2")

library(ggplot2)
library(plyr)

  df <- read.csv("/home/yacine/Documents/performance/M2R-ParallelQuicksort/data/sama_2014-10-13/measure
  head(df)
```

```
##   Size        Type     Time
## 1  100  Sequential 0.000010
## 2  100    Parallel 0.004024
## 3  100    Built-in 0.000013
## 4  100  Sequential 0.000010
## 5  100    Parallel 0.004448
## 6  100    Built-in 0.000014
```

```
  plot(df$Size,df$Time,col=c("red","blue","green")[df$Type])
```



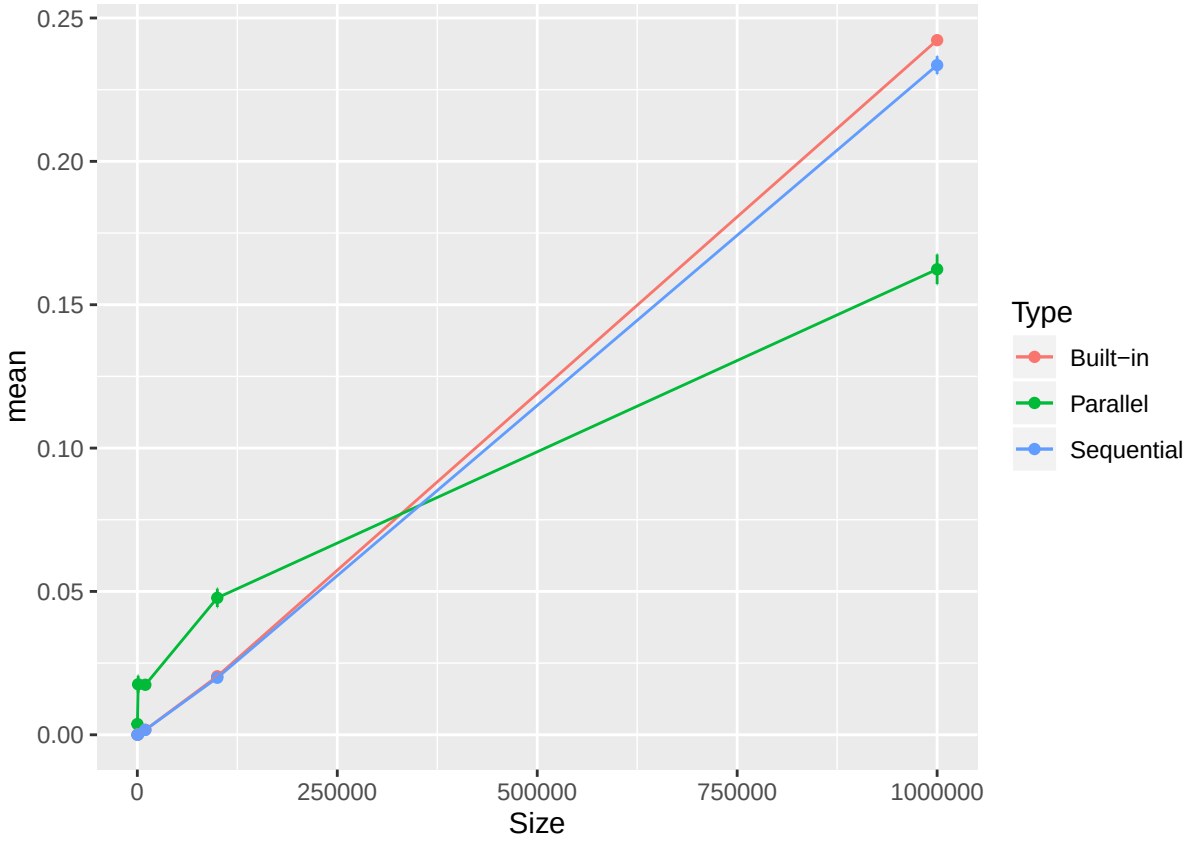

```
  ggplot(data=df, aes(x=Size, y=Time, color=Type))+geom_point()
```




```
df_sum = ddply(df, c("Size", "Type"), summarize, num=length(Time), mean=mean(Time), sd= sd(Time), se=
df_sum
```

```
##       Size        Type num      mean           sd           se
## 1      100    Built-in   5 0.0000126 1.140175e-06 1.019804e-06
## 2      100    Parallel   5 0.0037454 5.188842e-04 4.641041e-04
## 3      100  Sequential   5 0.0000098 4.472136e-07 4.000000e-07
## 4     1000    Built-in   5 0.0002078 3.834058e-06 3.429286e-06
## 5     1000    Parallel   5 0.0176116 3.378959e-03 3.022233e-03
## 6     1000  Sequential   5 0.0001278 1.095445e-06 9.797959e-07
## 7    10000    Built-in   5 0.0017194 1.165333e-05 1.042305e-05
## 8    10000    Parallel   5 0.0174410 9.699515e-04 8.675510e-04
## 9    10000  Sequential   5 0.0016958 4.669261e-05 4.176314e-05
## 10  100000    Built-in   5 0.0204072 1.263555e-04 1.130158e-04
## 11  100000    Parallel   5 0.0477688 3.609278e-03 3.228237e-03
## 12  100000  Sequential   5 0.0198892 1.405763e-04 1.257353e-04
## 13 1000000    Built-in   5 0.2422674 6.296517e-04 5.631776e-04
## 14 1000000    Parallel   5 0.1623540 5.800859e-03 5.188446e-03
## 15 1000000  Sequential   5 0.2335652 3.502431e-03 3.132669e-03
```

```
ggplot(data=df_sum,aes(x=Size, y=mean, ymin=mean-se, ymax= mean+se, color=Type))+geom_errorbar()+geom_
```




```
#ggplot(data=df, aes(x=Size, y=Time, color=factor(Type), shape=factor(option_compil)))+geom_point()
```

The sizes

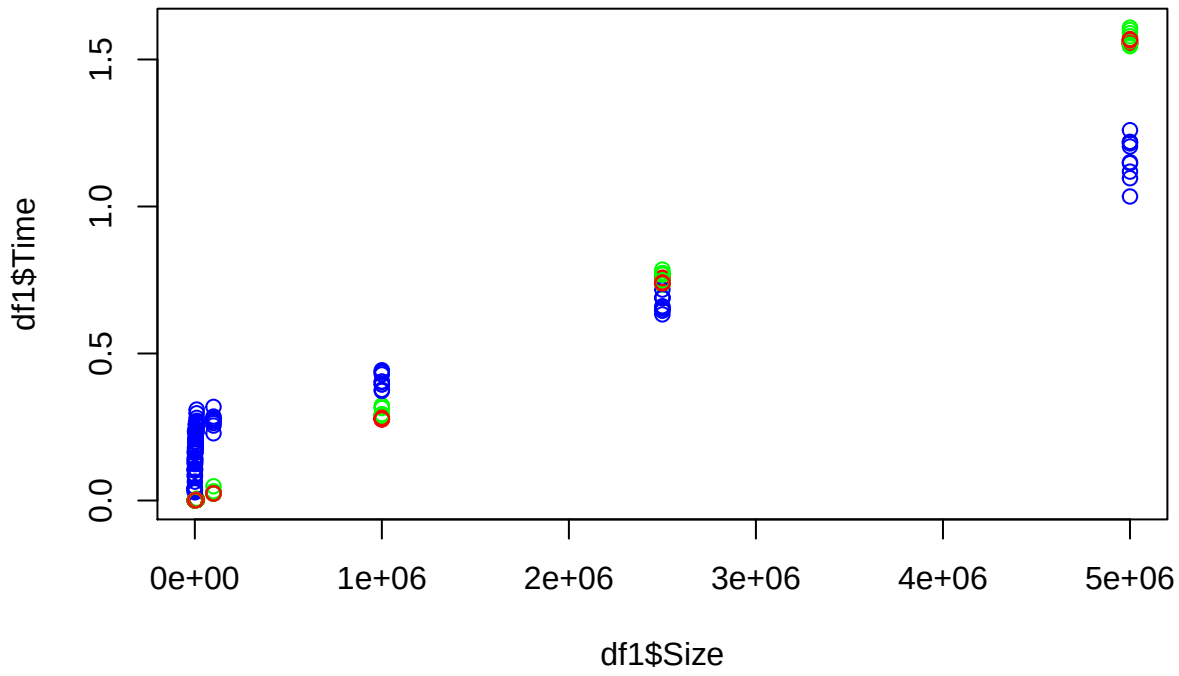
Instead of increasing the size of the array gradually, let's try to choose different sizes in a pretty mixed way. In the following line, we have the array sizes we use in the script:

1000 2500000 10000 100 5000 1000000 800 100000 430 5000000 4000

```
 library(ggplot2)
 library(plyr)

df1 <- read.csv("/home/yacine/Documents/performance/M2R-ParallelQuicksort/data/yacine-S550CA_2016-01-3
 head(df1)
```

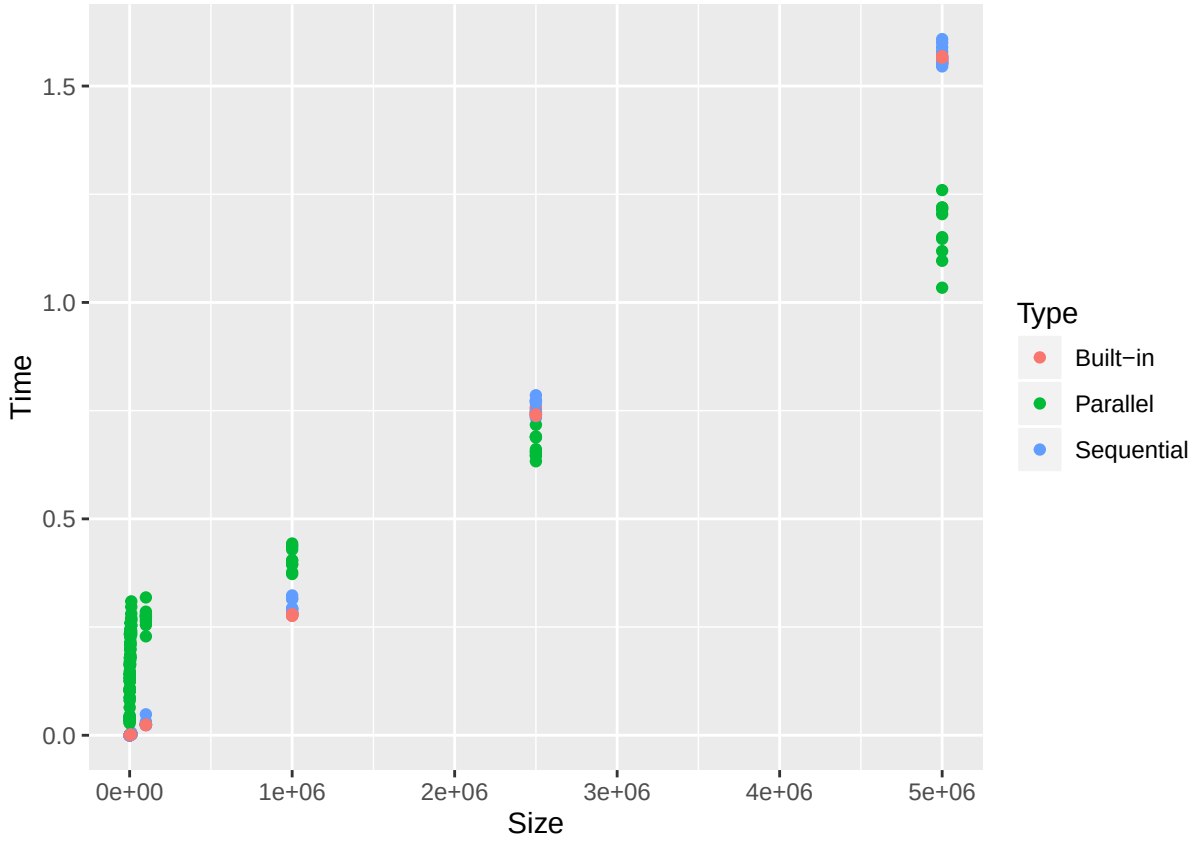
```
##   Size        Type     Time
## 1 1000  Sequential 0.000383
## 2 1000    Parallel 0.140358
## 3 1000    Built-in 0.000257
## 4 1000  Sequential 0.000164
## 5 1000    Parallel 0.108913
## 6 1000    Built-in 0.000257
```

```
plot(df1$Size,df1$Time,col=c("red","blue","green")[df1$Type])
```



Let's see the different execution times with ggplot

```
ggplot(data=df1, aes(x=Size, y=Time, color=Type))+geom_point()
```



We can clearly see that the parallel quick sort is not very efficient for little array sizes. The built in quick sort is the best, then the sequential and the parallel one is the worst when we have little arrays. This tend to change after the 2.5M size. When we have a 5M size, it is considerably better to use the parallel quick sort.

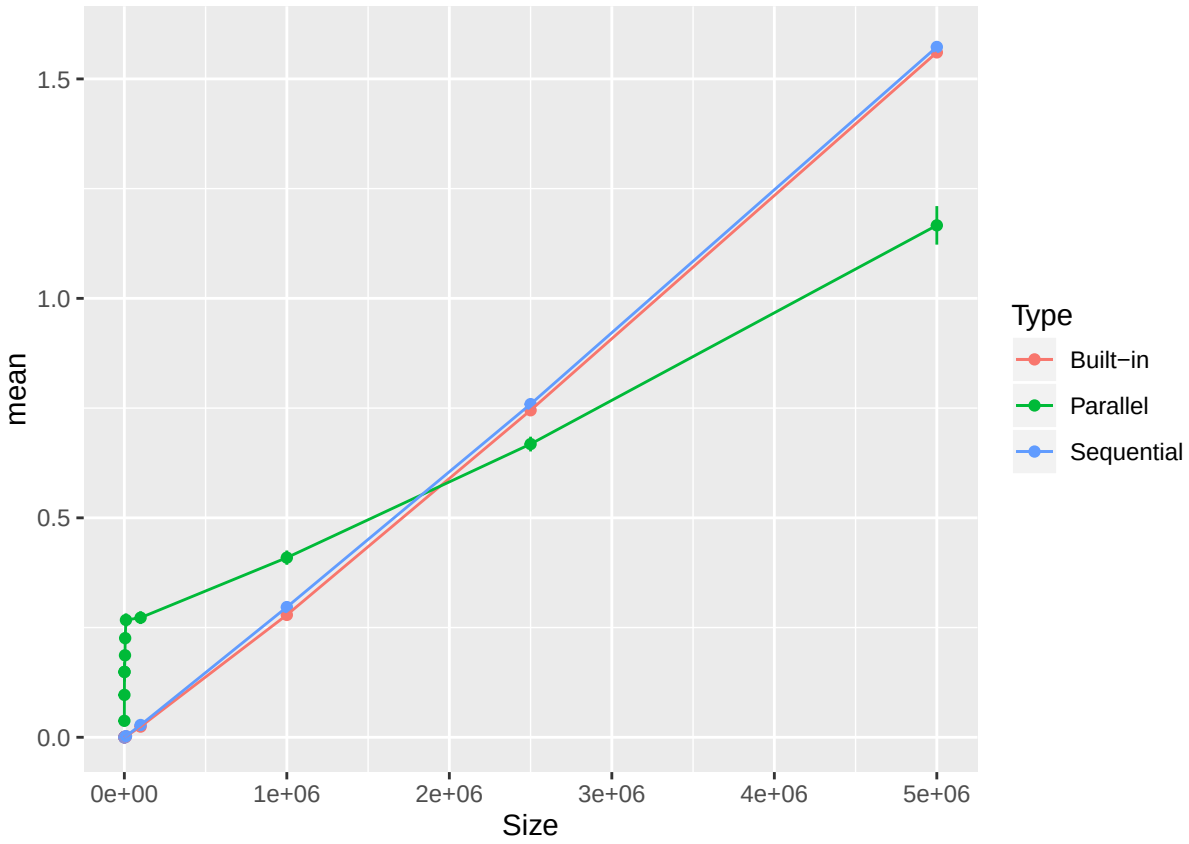
Confidence interval

Now let's see the confidence interval

```
    df1_sum = ddply(df1, c("Size", "Type"), summarize, num=length(Time), mean=mean(Time), sd= sd(Time),
  df1_sum
```

```
##       Size        Type num      mean           sd           se
## 1      100    Built-in  10 0.0000158 4.216370e-07 2.666667e-07
## 2      100    Parallel  10 0.0375457 5.667356e-03 3.584351e-03
## 3      100  Sequential  10 0.0000132 1.135292e-06 7.180220e-07
## 4      430    Built-in  10 0.0001748 1.619328e-06 1.024153e-06
## 5      430    Parallel  10 0.0965844 1.728921e-02 1.093466e-02
## 6      430  Sequential  10 0.0000677 5.478240e-06 3.464743e-06
## 7      800    Built-in  10 0.0002386 5.521674e-06 3.492214e-06
## 8      800    Parallel  10 0.1490409 1.709129e-02 1.080948e-02
## 9      800  Sequential  10 0.0001589 8.798163e-05 5.564447e-05
## 10    1000    Built-in  10 0.0002564 4.427189e-06 2.800000e-06
## 11    1000    Parallel  10 0.1489734 3.476853e-02 2.198955e-02
## 12    1000  Sequential  10 0.0001853 6.948389e-05 4.394547e-05
## 13    4000    Built-in  10 0.0008529 1.508826e-05 9.542653e-06
## 14    4000    Parallel  10 0.1867944 1.381444e-02 8.737016e-03
## 15    4000  Sequential  10 0.0007487 3.019952e-05 1.909985e-05
## 16    5000    Built-in  10 0.0010449 1.760335e-05 1.113333e-05
## 17    5000    Parallel  10 0.2259011 2.244976e-02 1.419848e-02
## 18    5000  Sequential  10 0.0009795 2.876437e-05 1.819218e-05
## 19   10000    Built-in  10 0.0021134 2.278986e-05 1.441357e-05
## 20   10000    Parallel  10 0.2672344 2.375610e-02 1.502467e-02
## 21   10000  Sequential  10 0.0025239 1.284060e-03 8.121109e-04
## 22  100000    Built-in  10 0.0243268 5.792073e-04 3.663228e-04
## 23  100000    Parallel  10 0.2725303 2.287437e-02 1.446702e-02
## 24  100000  Sequential  10 0.0279665 7.560262e-03 4.781529e-03
## 25 1000000    Built-in  10 0.2787734 2.660901e-03 1.682902e-03
## 26 1000000    Parallel  10 0.4092811 2.570134e-02 1.625496e-02
## 27 1000000  Sequential  10 0.2966316 1.542580e-02 9.756131e-03
## 28 2500000    Built-in  10 0.7450571 6.725889e-03 4.253826e-03
## 29 2500000    Parallel  10 0.6680882 2.657049e-02 1.680465e-02
## 30 2500000  Sequential  10 0.7590628 1.754608e-02 1.109711e-02
## 31 5000000    Built-in  10 1.5603784 6.502567e-03 4.112585e-03
## 32 5000000    Parallel  10 1.1663412 6.940289e-02 4.389424e-02
## 33 5000000  Sequential  10 1.5728140 2.112513e-02 1.336071e-02
```

```
ggplot(data=df1_sum,aes(x=Size, y=mean, ymin=mean-se, ymax= mean+se, color=Type))+geom_errorbar()+geom_
```



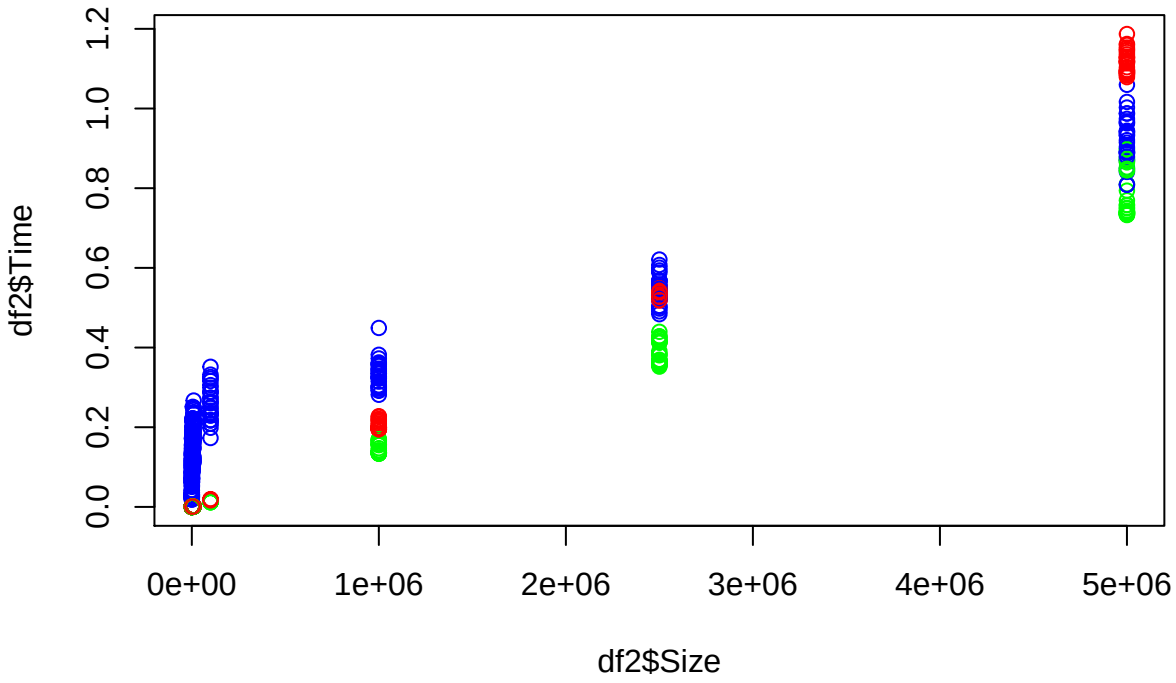
Here we can see after applying the confidence interval that there is the very slight difference between the built in and the sequential algorithms for a little array size but when the size is increasing, the built in algorithm is a little bit better. For the parallel quick sort,like we said before, it is only efficient starting a certain array size and then we can notice the exact opposite of time evolution compared to the other 2 algorithms.

The GCC compiler options

```
df2 <- read.csv("/home/yacine/Documents/performance/M2R-ParallelQuicksort/data/yacine-S550CA_2016-02-0
 head(df2)
```

```
##   Size Compilation        Type     Time
## 1 1000          -O1  Sequential 0.000290
## 2 1000          -O1    Parallel 0.172405
## 3 1000          -O1    Built-in 0.000251
## 4 1000          -O2  Sequential 0.000082
## 5 1000          -O2    Parallel 0.108279
## 6 1000          -O2    Built-in 0.000247
```

```
 plot(df2$Size,df2$Time,col=c("red","blue","green")[df2$Type])
```

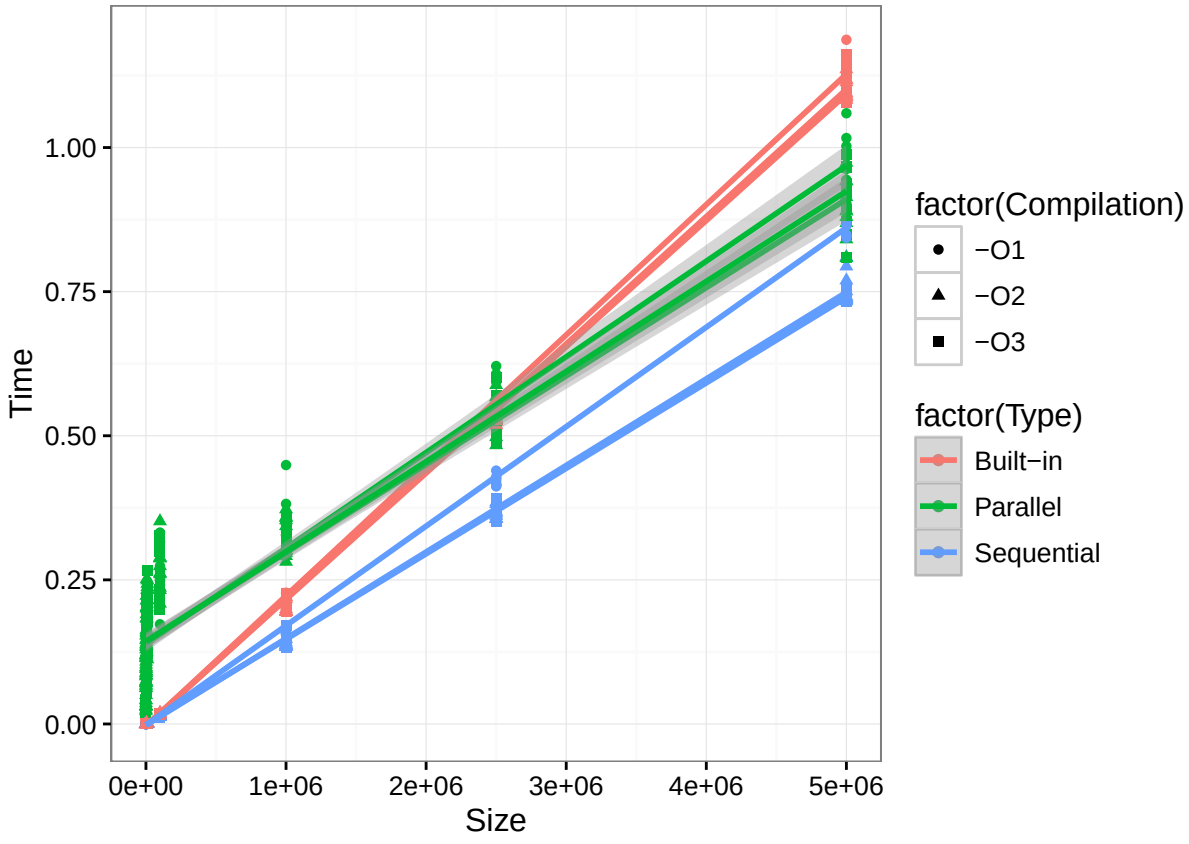


Let's see the different execution times with ggplot

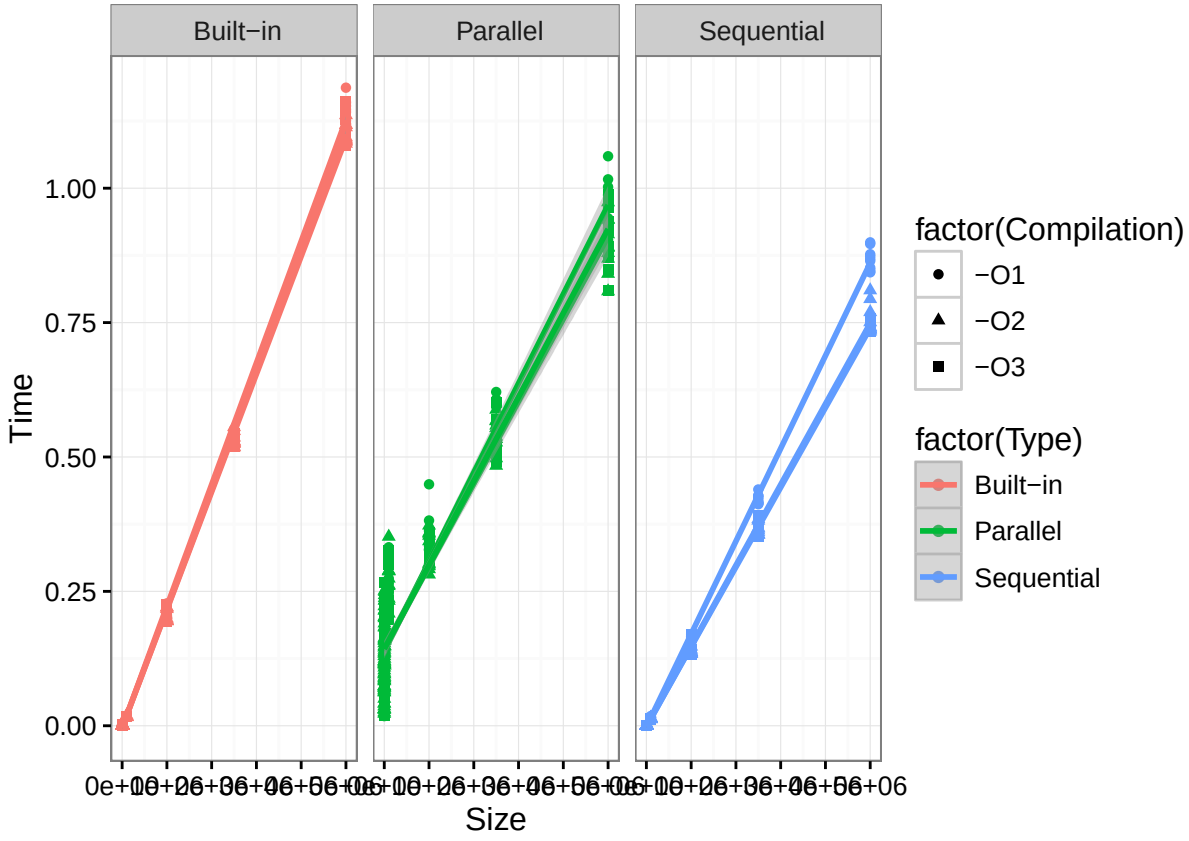
```
library(dplyr)
```

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

## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

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

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

```
ggplot(data=df2, aes(x=Size, y=Time, color=factor(Type), shape= factor(Compilation)))+geom_point()+theme
```




```
ggplot(data=df2, aes(x=Size, y=Time, color=factor(Type), shape= factor(Compilation)))+geom_point()+theme
```


Built−in
Parallel
Sequential
Time
1.00
0.75
0.50
0.25
0.00
Size
factor(Compilation)
−O1
−O2
−O3
factor(Type)
Built−in
Parallel
Sequential