STUDIO STATISTICO DI ALGORITMI DI ORDINAMENTO

Matteo Finazzi, Davide Tissino, Fabrizio Travagnin

SOMMARIO

INTRODUZIONE

La seguente relazione vuole essere uno studio statistico di alcuni algoritmi di ordinamento. I tre algoritmi che vengono confrontati sono Quicksort, Shellsort e Bubblesort.

Il confronto fra gli algoritmi di ordinamento di numeri si basa sul diverso tempo impiegato da questi a portare a termine il compito. Come è intuibile, nonché noto, tale tempo è dipendente dalla quantità di numeri da ordinare forniti al programma. Abbiamo indicato questo numero con la variabile "complessità", mentre il tempo necessario al conseguimento del compito, misurato in secondi, con la variabile "tempo". È doveroso aggiungere che la precisione di campionamento dei tempi del microcomputer è dell'ordine del microsecondo.

I dati sono stati ottenuti eseguendo un programma scritto in c++ su un Raspberry Pi 4 (i cui listati sono posti in fondo al documento). Infatti, a seguito di diversi tentativi su un pc è stato notato che l'alta velocità di calcolo di un normale computer combinata alla bassa precisione di campionamento dei tempi fornita da Windows non ha permesso di ottenere dati sufficientemente precisi per la nostra analisi.

Dai dati grezzi ottenuti dal Raspberry sono stati estratti 7 campioni, tutti di 101 elementi: innanzitutto ad ogni algoritmo sono stati fatti ordinare 8000 numeri, ripetendo le misure 101 volte, in modo da ottenere tre campioni aleatori, con le ipotesi che le 101 misure del tempo di esecuzione siano indipendenti e identicamente distribuite per ogni algoritmo; successivamente si è ripetuto questo procedimento misurando però solo il tempo di esecuzione di Quicksort nell'ordinare 10734 numeri (questo valore è stato scelto a caso tra quelli non forniti al modello di regressione lineare); infine negli ultimi tre campioni è stato raccolto il tempo di esecuzione in funzione della complessità per ogni algoritmo; in particolare si sono fatti ordinare ai tre algoritmi array da 5000 a 15000 elementi numerici casuali con uno step di 100 numeri. Per ogni misura i numeri da ordinare sono stati generati a caso dal programma stesso, scegliendoli nell'intervallo [0, SIZE), dove SIZE rappresenta di volta in volta la dimensione dell'array, nonché la "complessità" del problema.

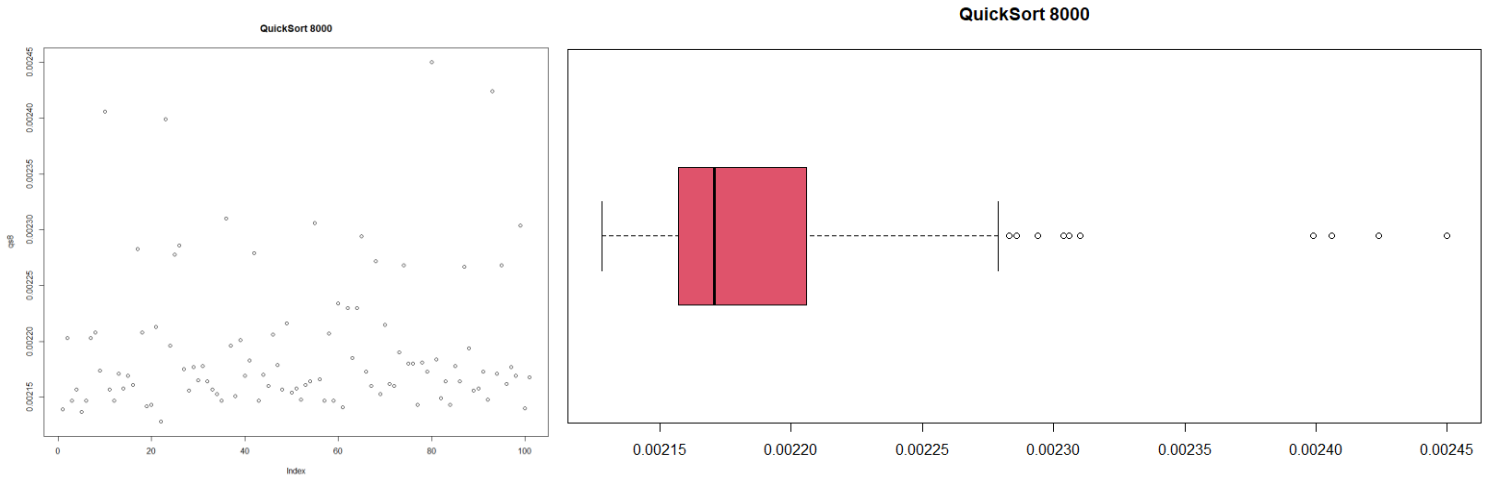
I primi tre campioni saranno studiati nella parte di statistica descrittiva per analizzare nel dettaglio le caratteristiche di ciascun algoritmo, gli ultimi tre campioni saranno alla base dei modelli di regressione lineare proposti per poter stimare il tempo di esecuzione in funzione della complessità e dell'algoritmo, infine il quarto campione tornerà utile nel paragrafo di

statistica inferenziale dove si confronterà un valore stimato dal miglior modello di regressione con il suo valore empirico.

STATISTICA DESCRITTIVA

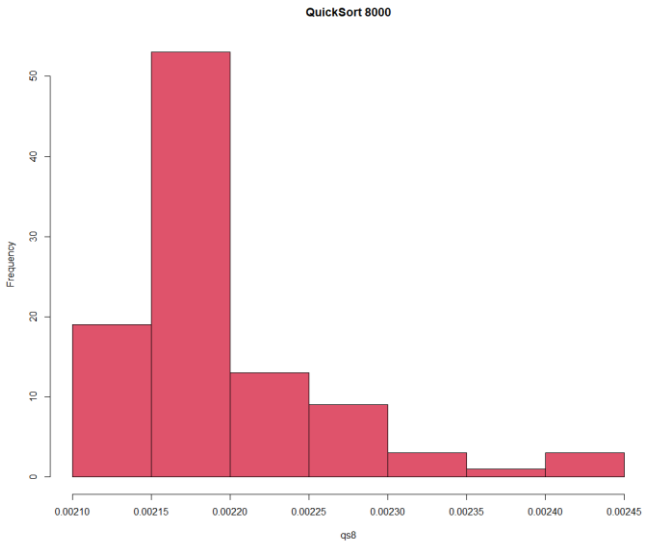
Si procede all'analisi dei campioni ottenuti. Nella trattazione si procede analizzando i primi 4 campioni (così come sono stati presentati nell'introduzione) ordinati per algoritmo. I tre campioni dipendenti dalla complessità verranno meglio trattati nel paragrafo sulla regressione.

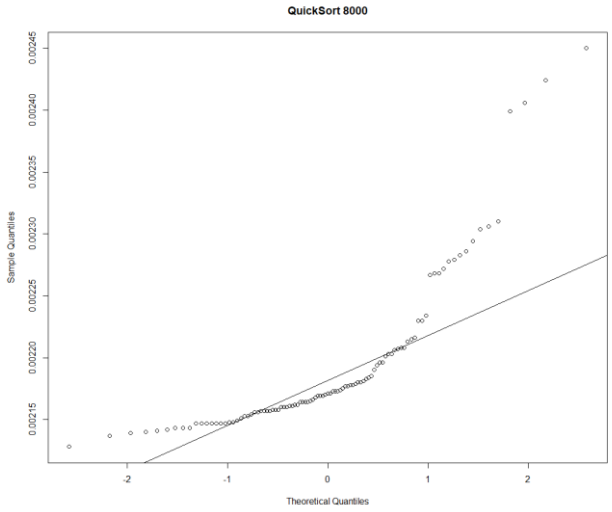
QUICKSORT - 8000




```
> summary(qs8)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.002128 0.002157 0.002171 0.002194 0.002206 0.002450

> sd(qs8)
[1] 6.339913e-05
```

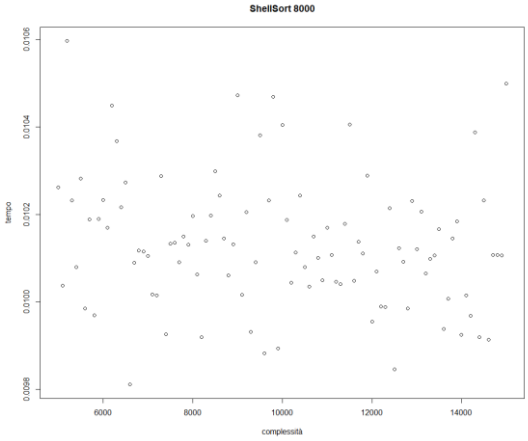


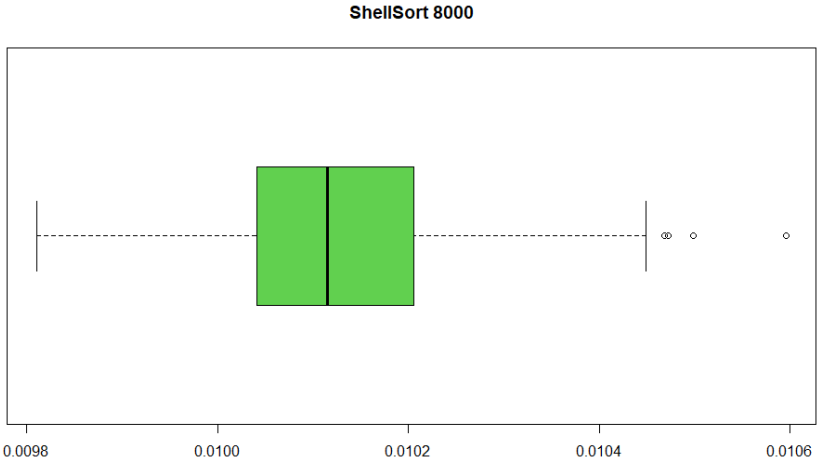


Lo scatterplot non mette in evidenza pattern particolari ma si può notare che la maggior parte dei valori è compresa nella fascia [0.00215, 0.00225], mentre i restanti valori sono disposti oltre il margine destro di tale intervallo, creando quindi una coda ben visibile anche dall'istogramma. Si evidenzia inoltre un discreto numero di outlier sempre in senso positivo; queste due caratteristiche, come si vedrà in seguito, saranno molto ricorrenti nel resto dei campioni analizzati. È sufficiente osservare il qqplot per decretare che non si tratta certamente di un campione normale.

```
> shapiro.test(qs8)
Shapiro-Wilk normality test
data:  qs8
W = 0.74239, p-value = 5.077e-12
```

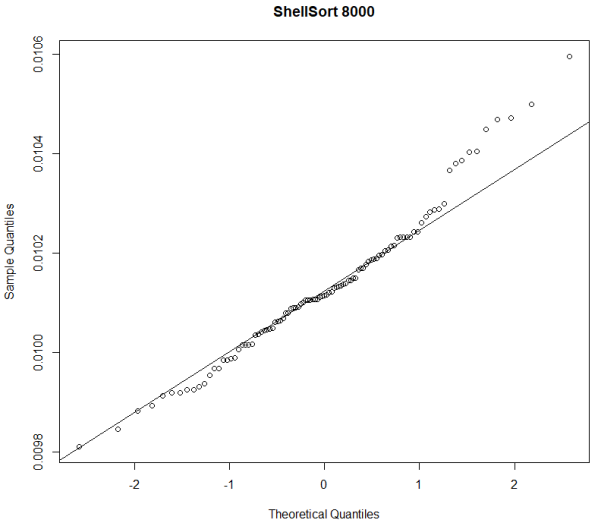
SHELLSORT - 8000

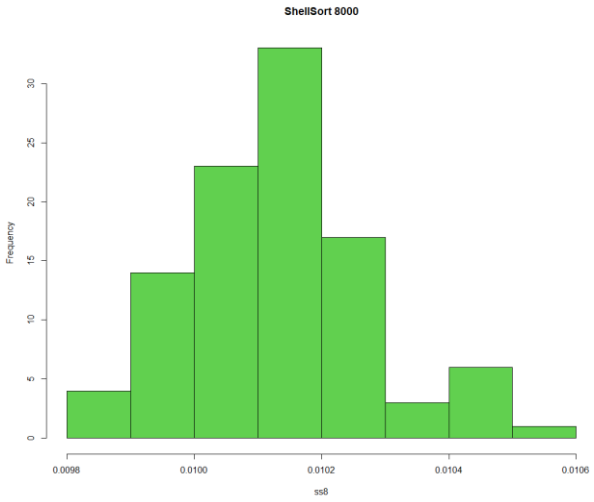




```
> summary(ss8)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.009811 0.010041 0.010115 0.010131 0.010206 0.010596

> sd(ss8)
[1] 0.0001486827
```

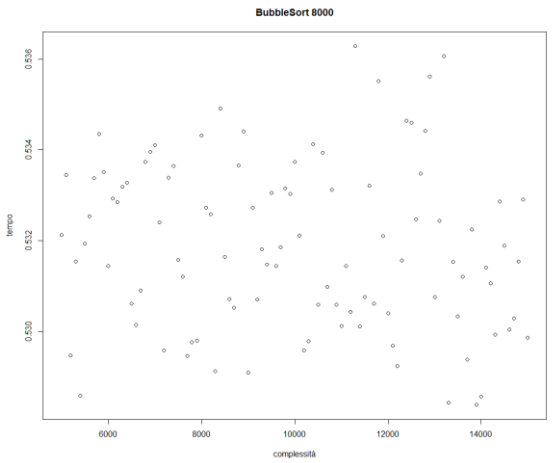


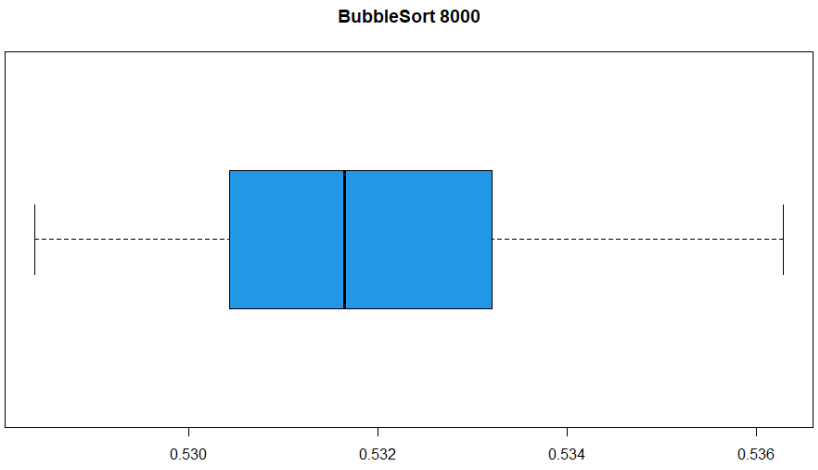


Dallo scatterplot non sono evidenti pattern particolari; si segnala sempre la presenza di una coda verso destra (valori di tempo più elevati) e di 4 outliers sempre verso destra. Nel qqplot I punti si allineano bene alla qqline nella parte centrale ma divergono molto sulle code: dal test di Shapiro-Wilk il campione risulta infatti non normale.

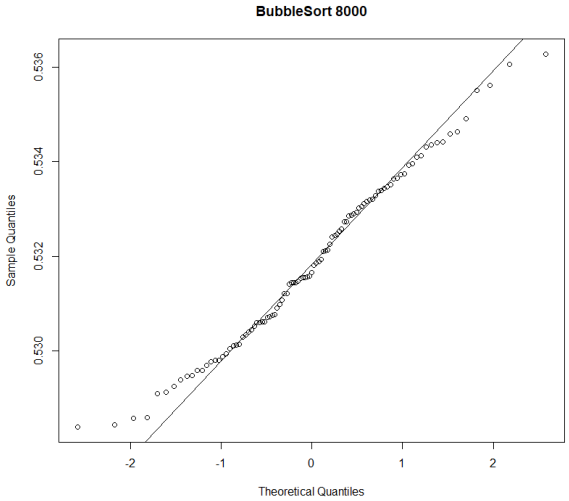
```
> shapiro.test(ss8)
Shapiro-Wilk normality test
data:  ss8
W = 0.97247, p-value = 0.03279
```

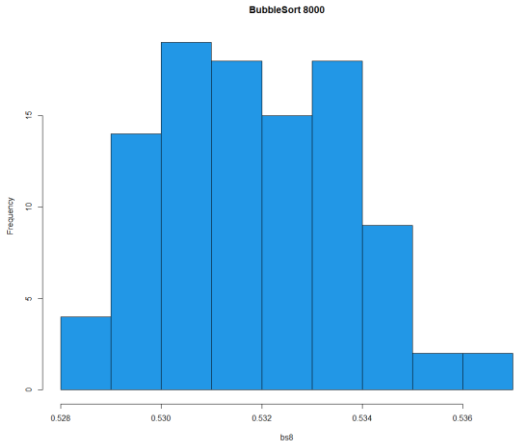
BUBBLESORT - 8000





```
> summary(bs8)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.5284  0.5304  0.5316  0.5319  0.5332  0.5363
> sd(bs8)
[1] 0.00184089
```

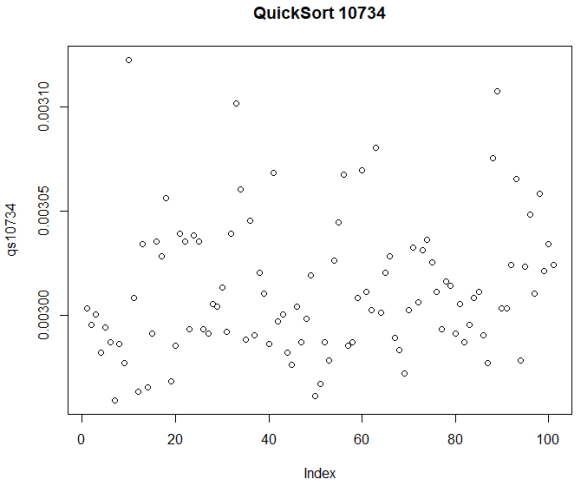


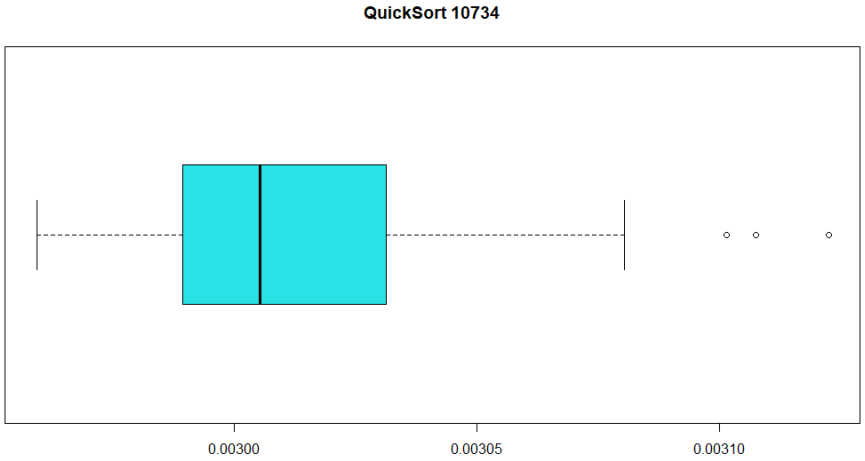


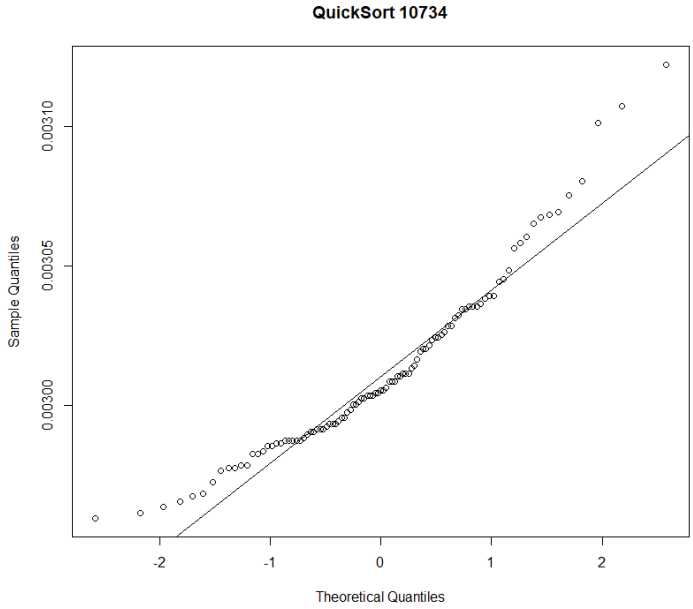
Anche in questo caso lo scatterplot mette in evidenza la presenza di una piccola coda verso destra, il campione è però esente da outliers e non possiamo escludere che sia normale, visti l'andamento in buona parte lineare dei punti del qqplot e l'elevato p-value del test di Shapiro-Wilk. La gaussianità è anche suggerita dall'istogramma.

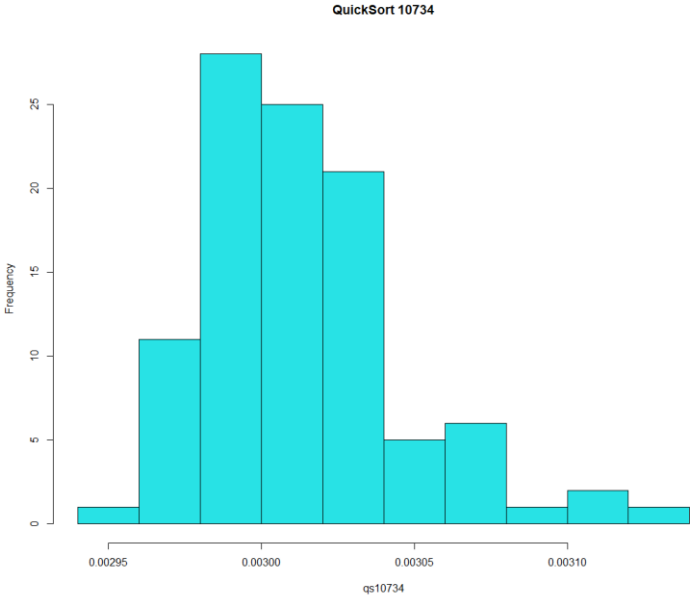
```
> shapiro.test(bs8)
Shapiro-Wilk normality test
data:  bs8
W = 0.98307, p-value = 0.223
```

QUICKSORT - 10734










```
> summary(qs10734)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.002959 0.002989 0.003005 0.003012 0.003031 0.003122

> sd(qs10734)
[1] 3.237344e-05
```

Pure quest'ultimo campione, ottenuto nuovamente mediante l'utilizzo dell'algoritmo Quicksort, ripresenta le tendenze viste in precedenza: pronunciata coda verso destra e discreta presenza di outliers nello stesso senso. La gaussianità del campione va rifiutata visto il bassissimo valore del p-value del test di Shapiro-Wilk (decisione forte).

```
> shapiro.test(qs10734)
Shapiro-Wilk normality test
data:  qs10734
W = 0.93682, p-value = 0.0001145
```

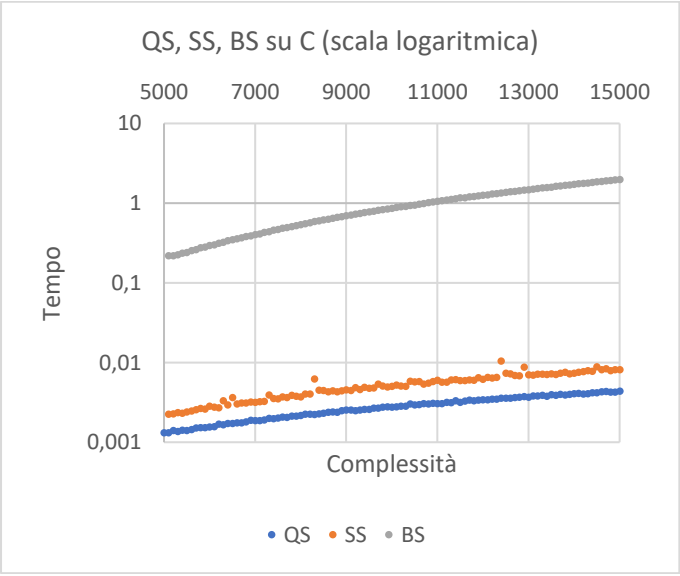
Come prevedibile, la media campionaria per questo campione risulta maggiore di quella relativa al medesimo algoritmo sottoposto a minore complessità.

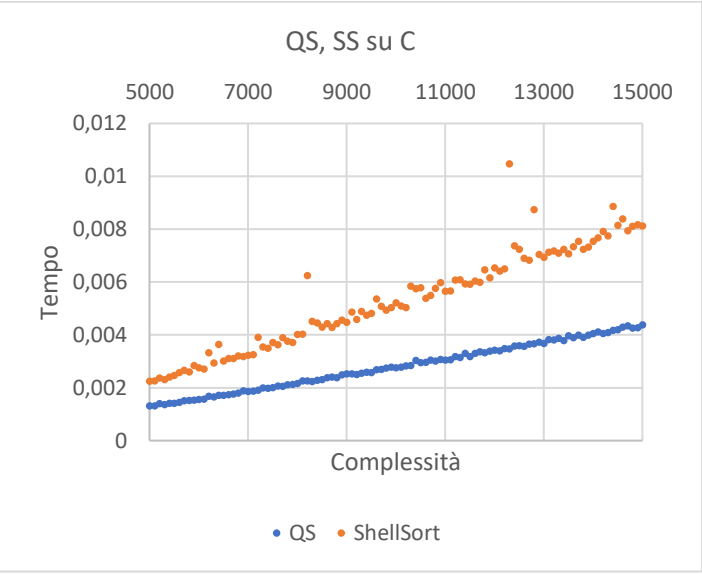
OSSERVAZIONE SULLE DEVIAZIONI STANDARD

Le tre deviazioni standard dei tre campioni sono significativamente diverse (almeno di un ordine di grandezza) e sembrano aumentare con la complessità del campione. Nella parte di inferenza statistica verranno riprese ed ampliate queste osservazioni.

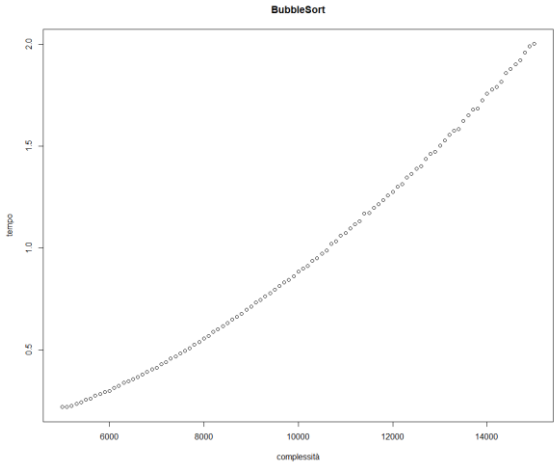
REGRESSIONE LINEARE

Nelle seguenti pagine verranno proposti alcuni modelli di regressione lineare relativi ai tre algoritmi in analisi. Lo scopo di questi modelli è quello di indagare la relazione che lega la complessità di un problema al suo tempo di risoluzione da parte dei tre diversi algoritmi al fine di poterne valutare l'efficienza (per semplicità qui misurata soltanto in termini di tempo di esecuzione). Inoltre la messa a punto di un modello consistente per Quicksort ci consentirà di stimare il tempo di esecuzione per l'ordinamento di 10734 numeri, che verrà confrontato con quello ricavato dal campione Quicksort-10734 (non utilizzato nella creazione del modello).
A questo scopo sono stati utilizzati i tre set di input (uno per algoritmo) di 101 misure. Ogni misura differisce dalla precedente per complessità, la quale va da 5000 a 15000 con uno step di 100 numeri. Per ottenere queste misure si è usata una versione modificata del programma, il cui listato è fornito in fondo al documento.
Le curve così ottenute sono state plottate qui di seguito: dapprima tutte insieme su scala logaritmica, vista la grande differenza di velocità, apprezzabile anche ad occhio, tra i più veloci Quicksort e Shellsort, e il lento Bubblesort, in seguito mettendo a confronto, su scala lineare, Quicksort e Shellsort, i cui tempi di esecuzione sono relativamente simili.





BUBBLESORT

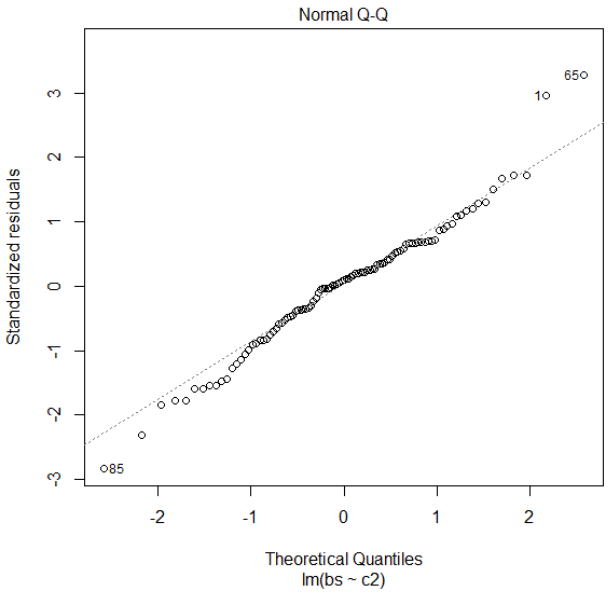


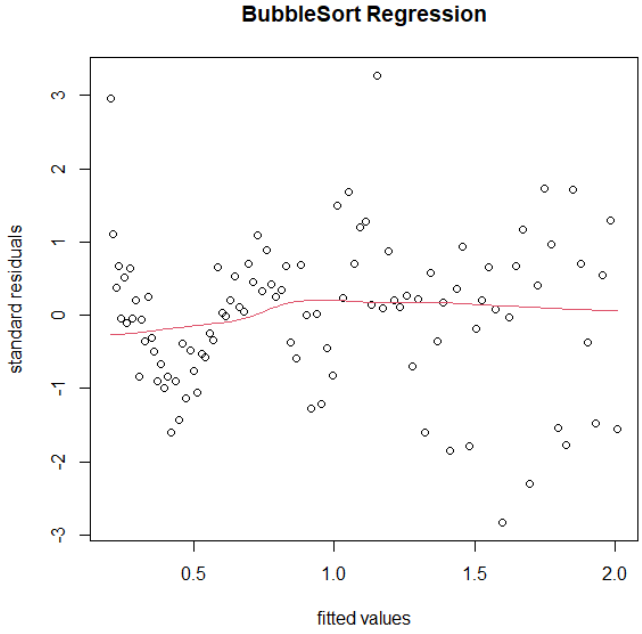
Osservando il grafico a lato è facile ipotizzare una correlazione di tipo quadratico tra la complessità e il tempo: i punti sembrano molto ben allineati sul grafico di una parabola e a prima vista non saltano all'occhio potenziali outlier.
Il modello proposto è il seguente:

$tempo_i = \beta_0 + \beta_1 complessità_i^2 + E_i$ con $E_i \sim N(0, \sigma^2)$

```
summary(bsregr)
Call:
lm(formula = bs ~ c2)
Residuals:
       Min         1Q     Median         3Q        Max
-0.0148473 -0.0030117  0.0005341  0.0033601  0.0172799
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -2.164e-02  1.109e-03  -19.51   <2e-16 ***
c2           9.028e-09  8.987e-12 1004.57   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.005311 on 99 degrees of freedom
Multiple R-squared:  0.9999,    Adjusted R-squared:  0.9999
F-statistic: 1.009e+06 on 1 and 99 DF,  p-value: < 2.2e-16
```

Osservando il p-value dell'F-test e l'r^2 si deduce l'ottima significatività globale del modello nonché il fatto che la quasi totalità della variabilità viene spiegata. I regressori sono entrambi significativi, tuttavia non sono ben rispettate le ipotesi del modello lineare gaussiano. Infatti, nonostante dall'analisi del qq plot e dal p-value del test di Shapiro-Wilk si evinca la normalità dei residui, essi non risultano omoschedastici: per i valori di complessità minore appare un andamento sinusoidale che va, per valori di complessità superiori, a disperdersi nel tipico "grafico a nuvola" indice di un buon modello.



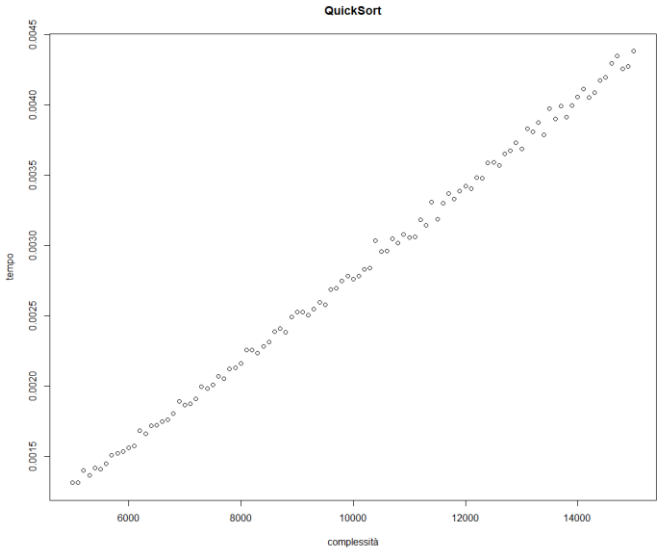



```
> shapiro.test(bsres)
Shapiro-Wilk normality test
data:  bsres
W = 0.97905, p-value = 0.1084
```

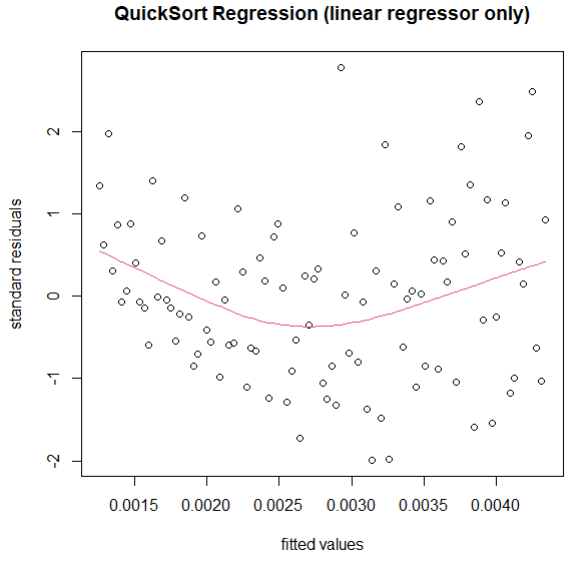
QUICKSORT

Il plot dei dati a fianco suggerirebbe un andamento lineare del tempo in funzione della complessità. I dati risultano ben allineati e non si segnala la presenza di outlier; il modello ipotizzato è il seguente:

$$tempo_i = \beta_0 + \beta_1 complessità_i + E_i$$
con $E_i \sim N(0, \sigma^2)$




```
> summary(qsregr)
 Call:
lm(formula = qs ~ c)
Residuals:
       Min         1Q     Median         3Q        Max
-7.958e-05 -2.776e-05 -1.316e-06  2.491e-05  1.113e-04
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -2.838e-04  1.428e-05  -19.87   <2e-16 ***
c            3.084e-07  1.371e-09  224.91   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 4.018e-05 on 99 degrees of freedom
Multiple R-squared:  0.998,Adjusted R-squared:  0.998
F-statistic: 5.058e+04 on 1 and 99 DF,  p-value: < 2.2e-16
```

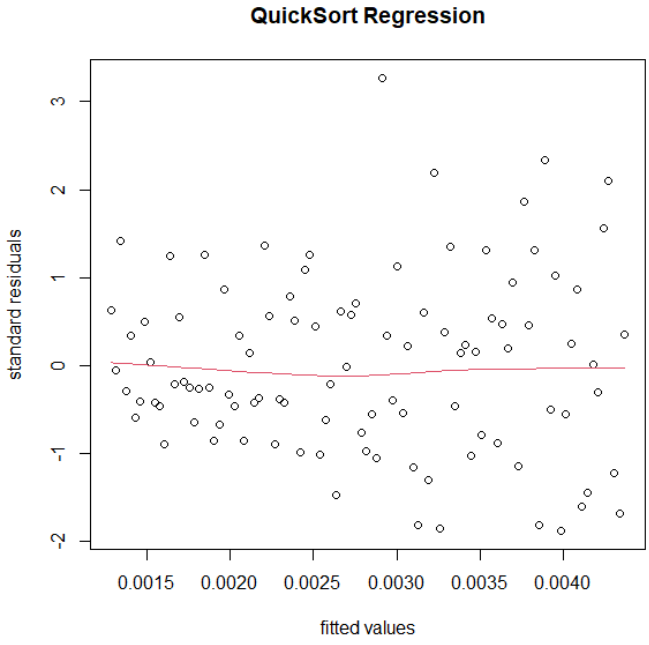


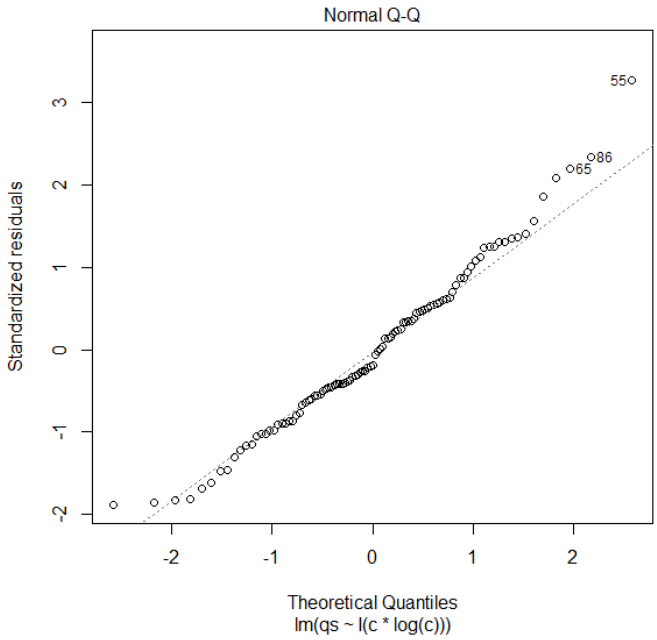

```
> shapiro.test(qsres)
Shapiro-Wilk normality test
data:  qsres
W = 0.98355, p-value = 0.2428
```

Nonostante non si possa rifiutare la gaussianità dei residui, è evidente che questi non siano omoschedastici: lo

scatterplot mette in evidenza una dipendenza di tipo non lineare. Dato che però il plot dei punti somiglia molto ad una retta, si è ipotizzato un andamento del tipo *xlogx.*
Il modello affinato è il seguente:
$tempo_i = \beta_0 + \beta_1 complessità_i \times \log(complessità_i) + E_i$ con $E_i \sim N(0, \sigma^2)$

```
> summary(qsregr2)
Call:
lm(formula = qs ~ I(c * log(c)))
Residuals:
       Min         1Q     Median         3Q        Max
-7.073e-05 -2.430e-05 -6.967e-06  2.169e-05  1.240e-04
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)   -2.406e-06  1.241e-05  -0.194    0.847
I(c * log(c))  3.029e-08  1.277e-10 237.238   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 3.809e-05 on 99 degrees of freedom
Multiple R-squared:  0.9982,    Adjusted R-squared:  0.9982
F-statistic: 5.628e+04 on 1 and 99 DF,  p-value: < 2.2e-16
```



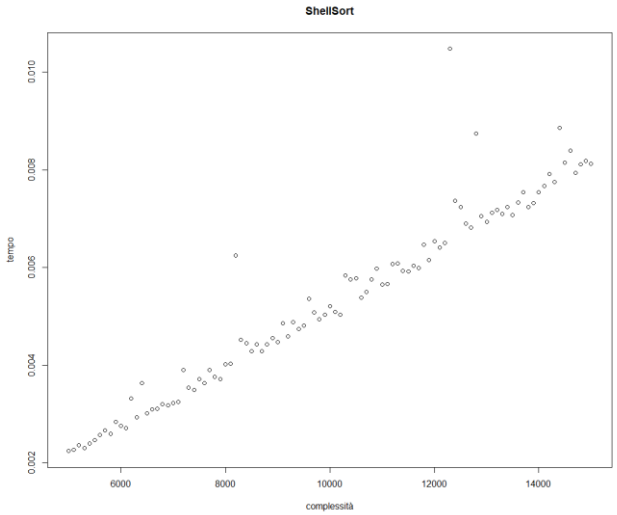


Nuovamente significatività globale e variabilità spiegata sono ottime, tuttavia in questo caso sono soddisfatte tutte le ipotesi del modello lineare gaussiano: si può notare infatti dallo scatterplot dei residui che i punti si dispongono in una forma a nuvola centrata sullo 0; inoltre circa il 95% di essi è contenuto nell'intervallo [-2, 2];

```
> shapiro.test(qsres2)
Shapiro-Wilk normality test
data:  qsres2
W = 0.98168, p-value = 0.1744
```

infine, sia l'analisi del qq-plot che del p-value dello Shapiro test non permettono di rifiutare l'ipotesi di gaussianità dei residui. Il modello ottenuto si può quindi considerare valido.

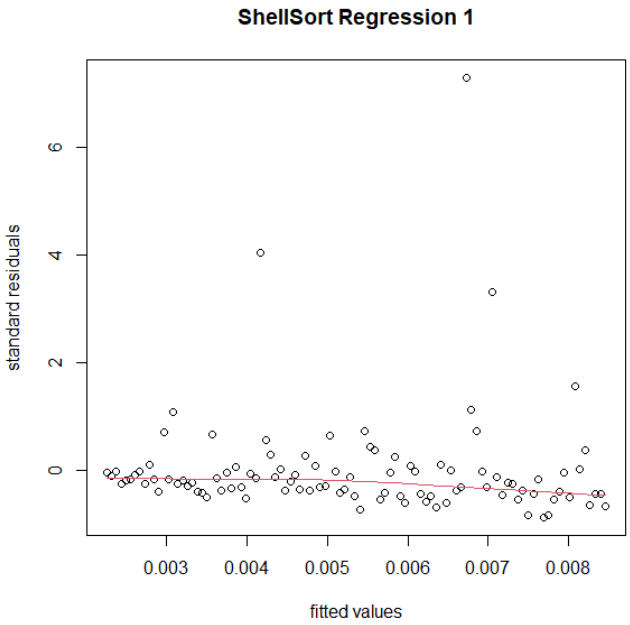
SHELLSORT



Già osservando il plot dei dati si può prevedere una consistente presenza di outlier nel verso crescente dell'asse dei tempi. Il primo modello proposto, che per analogia presenta lo stesso regressore del precedente, ha un r^2, seppur buono (>81%), inferiore ai precedenti. I residui di tale modello sono omoschedastici ma presentano almeno 3 forti outlier e non sono distribuiti simmetricamente rispetto alla retta orizzontale passante per 0; tutto ciò ne inficia la gaussianità come si vede dal qqplot e dal p-value del test di Shapiro-Wilk.

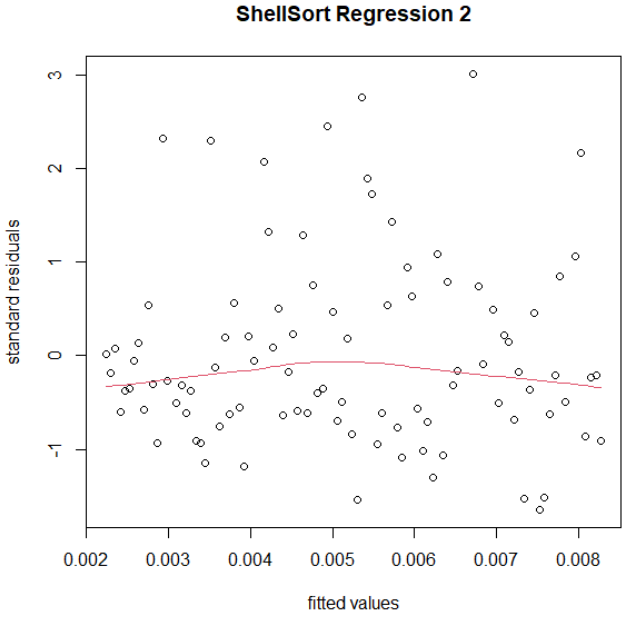
```
> summary(ssregr)
Call:
lm(formula = ss ~ I(c * log(c)))
Residuals:
       Min         1Q     Median         3Q        Max
-0.0004454 -0.0002175 -0.0001096  0.0000032  0.0037468
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)   -3.352e-04  1.685e-04   -1.99   0.0494 *
I(c * log(c))  6.098e-08  1.734e-09   35.17   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.0005173 on 99 degrees of freedom
Multiple R-squared:  0.9259,     Adjusted R-squared:  0.9252
F-statistic:  1237 on 1 and 99 DF,  p-value: < 2.2e-16
```

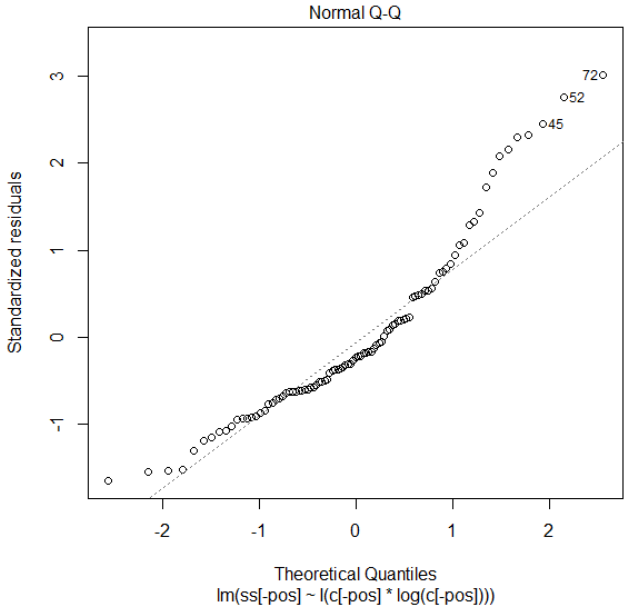
```
shapiro.test(rstandard(ssregr))
Shapiro-Wilk normality test
data:  rstandard(ssregr)
W = 0.50965, p-value < 2.2e-16
```



Si è dunque cercato di affinare tale modello rimuovendo gli outlier nelle posizioni [15, 33, 74, 75, 79, 95]:

```
> summary(ssregr2)
Call:
lm(formula = ss[-pos] ~ I(c[-pos] * log(c[-pos])))
Residuals:
       Min         1Q     Median         3Q        Max
-2.825e-04 -1.083e-04 -4.002e-05  8.599e-05  5.198e-04
Coefficients:
                           Estimate Std. Error t value Pr(>|t|)
(Intercept)               -2.878e-04  5.830e-05  -4.936 3.49e-06 ***
I(c[-pos] * log(c[-pos]))  5.938e-08  6.042e-10  98.281  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.0001744 on 93 degrees of freedom
Multiple R-squared:  0.9905,     Adjusted R-squared:  0.9904
F-statistic:  9659 on 1 and 93 DF,  p-value: < 2.2e-16
```





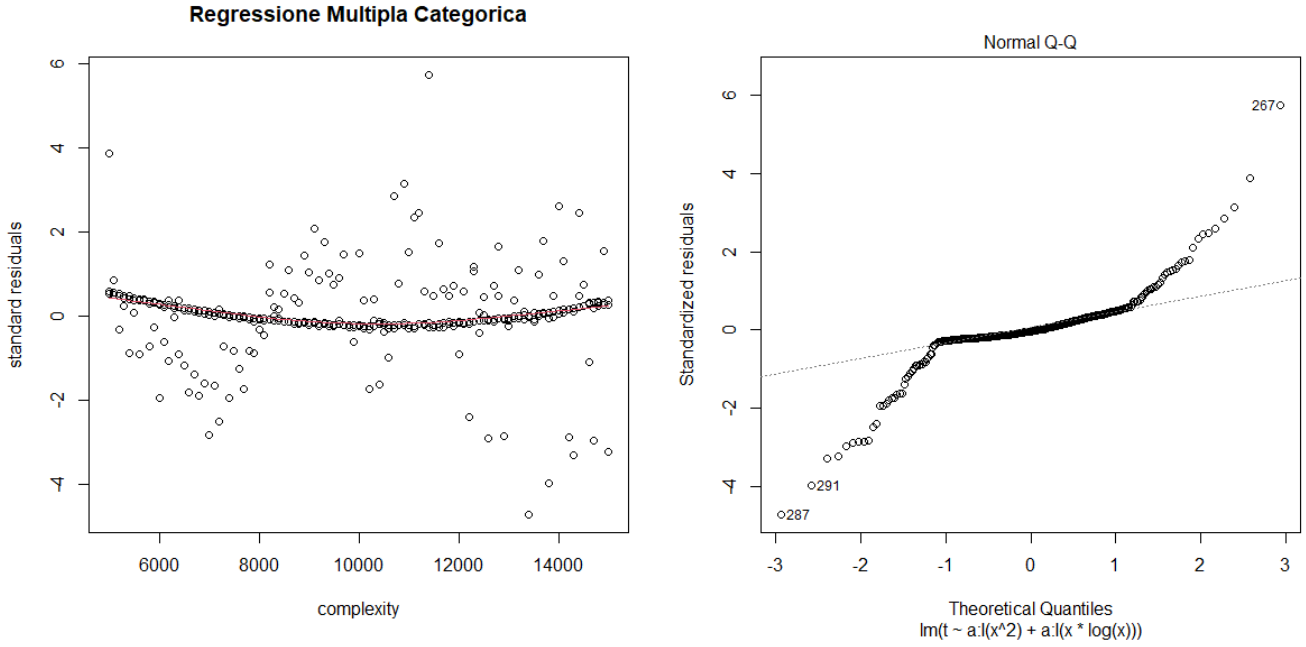

```
>shapiro.test(rstandard(ssregr2))
Shapiro-Wilk normality test
data:  rstandard(ssregr2)
W = 0.91967, p-value = 2.151e-05
```

Purtroppo si riconferma il trend asimmetrico precedente: i residui, seppur omoschedastici, si concentrano maggiormente sul lato positivo dello scatterplot e non risultano gaussiani, nonostante il p-value del test di Shapiro-Wilk sia aumentato significativamente. Il modello va quindi scartato perché non rispetta le ipotesi di gaussianità dei residui.
La tendenza dei residui a presentare una coda nel verso positivo dei tempi, riscontrata, seppur in grado molto minore, anche negli altri modelli, è stata causa di un'indagine aggiuntiva per cercare di comprenderne le motivazioni: le misure relative all'ultimo modello, dove il fenomeno era più evidente, sono state ripetute altre volte ottenendo risultati sorprendentemente simili. Alla luce di questo si è ipotizzato che la causa delle code sia dovuta al fatto che il processore sia impegnato nell'esecuzione di altri programmi che possono a tratti creare dei rallentamenti nello svolgimento dell'algoritmo; questo non spiega però come mai il fenomeno sia particolarmente accentuato nel caso dello Shellsort, fatto che ci costringe a supporre anche una causa intrinseca all'algoritmo e non facilmente spiegabile.

REGRESSIONE MULTIPLA CATEGORICA:

Nel tentativo di creare un modello unitario per tutti e tre i dataset e di migliorare la situazione riguardo i residui (pessima nella maggior parte dei casi precedenti), è stato tentato un modello di regressione multipla con l'aggiunta della variabile categorica "algoritmo" le cui categorie sono i tre diversi algoritmi (qs, ss, bs).

```
> summary(mregr)
Call:
lm(formula = t ~ a:I(x^2) + a:I(x * log(x)))
Residuals:
       Min         1Q     Median         3Q        Max
-0.0149932 -0.0006691 -0.0001071  0.0010242  0.0183490
Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)       -7.873e-03  2.089e-03  -3.769 0.000197 ***
aBS:I(x^2)         9.198e-09  2.938e-11 313.039  < 2e-16 ***
aQS:I(x^2)        -9.799e-11  2.938e-11  -3.335 0.000963 ***
aSS:I(x^2)        -9.435e-11  2.938e-11  -3.211 0.001469 **
aBS:I(x * log(x)) -3.474e-07  5.566e-08  -6.241 1.50e-09 ***
aQS:I(x * log(x))  2.295e-07  5.566e-08   4.124 4.84e-05 ***
aSS:I(x * log(x))  2.524e-07  5.566e-08   4.535 8.36e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 0.003229 on 296 degrees of freedom
Multiple R-squared:      1,Adjusted R-squared:      1
F-statistic: 1.434e+06 on 6 and 296 DF,  p-value: < 2.2e-16
```



Il caso è disastroso: la distribuzione dei residui è tutto fuorché normale, come si evince dal bassissimo valore del p-value dello Shapiro test e dalla forma a gradoni del qq-plot. Si vede inoltre come la maggior parte di questi si stringa lungo una parabola mentre i restanti lungo una linea sinusoidale che ricalca i residui trovati nel modello relativo al Bubblesort. La causa ipotizzata per questo comportamento è la grande differenza tra le varianze, come già è stato detto nel paragrafo sulla statistica descrittiva.

```
> shapiro.test(rstandard(mregr))
Shapiro-Wilk normality test
data:  rstandard(mregr)
W = 0.81135, p-value < 2.2e-16
```

INFERENZA STATISTICA:

Si intende ora testare le predizioni del modello relativo al Quicksort, l'unico che rispetta le ipotesi del modello lineare gaussiano. Per far ciò si fa riferimento al campione di 101 elementi con complessità di 10734. La sua media campionaria è stata calcolata nel paragrafo relativo alla statistica descrittiva.
Si procede ora calcolando tre intervalli di confidenza per il valore atteso ai tre livelli di confidenza usuali (90%, 95%, 99%).

```
> z.test(qs10734, sigma.x = sd(qs10734), conf.level=0.90)

      One-sample z-Test

data:  qs10734
z = 935.15, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
90 percent confidence interval:
 0.003007086 0.003017683
sample estimates:
  mean of x
0.003012385
```

```
> z.test(qs10734, sigma.x=sd(qs10734), conf.level=0.95)

      One-sample z-Test

data:  qs10734
z = 935.15, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.003006071 0.003018698
sample estimates:
  mean of x
0.003012385
```

```
> z.test(qs10734, sigma.x = sd(qs10734), conf.level=0.99)

      One-sample z-Test

data:  qs10734
z = 935.15, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
99 percent confidence interval:
 0.003004087 0.003020682
sample estimates:
  mean of x
0.003012385
```

Dagli output di R si evincono gli intervalli di confidenza:

- (0.003007086; 0.003017683) al 90%;
- (0.003006071 0.003018698) al 95%;
- (0.003004087 0.003020682) al 99%;

Attraverso il modello di regressione sopra ottenuto, si trova una stima puntuale del valore.

```
>newc = data.frame(c=10734)
> predict(qsregr2, newc)
            1
0.003014877
```

La media del campione (0.003012) è estremamente simile al valore stimato. Inoltre si può immediatamente notare che tale valore rientra in tutti e tre gli intervalli di confidenza sopra calcolati.
Si è infine condotto il seguente test sulla media per verificare l'uguaglianza con il valore stimato:

$$H_0: \mu = 0.003014877$$
$$H_1: \mu \neq 0.003014877$$

Trattandosi di un campione non normale ma numeroso a varianza incognita, si procede col seguente z-test.

```
> z.test(qs10734, sigma.x=sd(qs10734), mu=0.003014877, alternative = "two.sided")

One-sample z-Test

data:  qs10734
z = -0.77367, p-value = 0.4391
alternative hypothesis: true mean is not equal to 0.003014877
95 percent confidence interval:
 0.003006071 0.003018698
sample estimates:
  mean of x
0.003012385
```

Con un p-value così elevato non è possibile rifiutare H_0 (conclusione debole), così la nostra ipotesi risulta verificata.

OSSERVAZIONE SULLE DEVIAZIONI STANDARD

Le tre deviazioni standard dei tre campioni sono significativamente diverse (almeno un ordine di grandezza) e sembrano aumentare con la complessità del set di input. Queste affermazioni sono state verificate con 2 F-test opportuni (il terzo non è necessario, dato che la conclusione segue per la proprietà transitiva) i cui p-value sono risultati tutti inferiori a 2.2E-16:

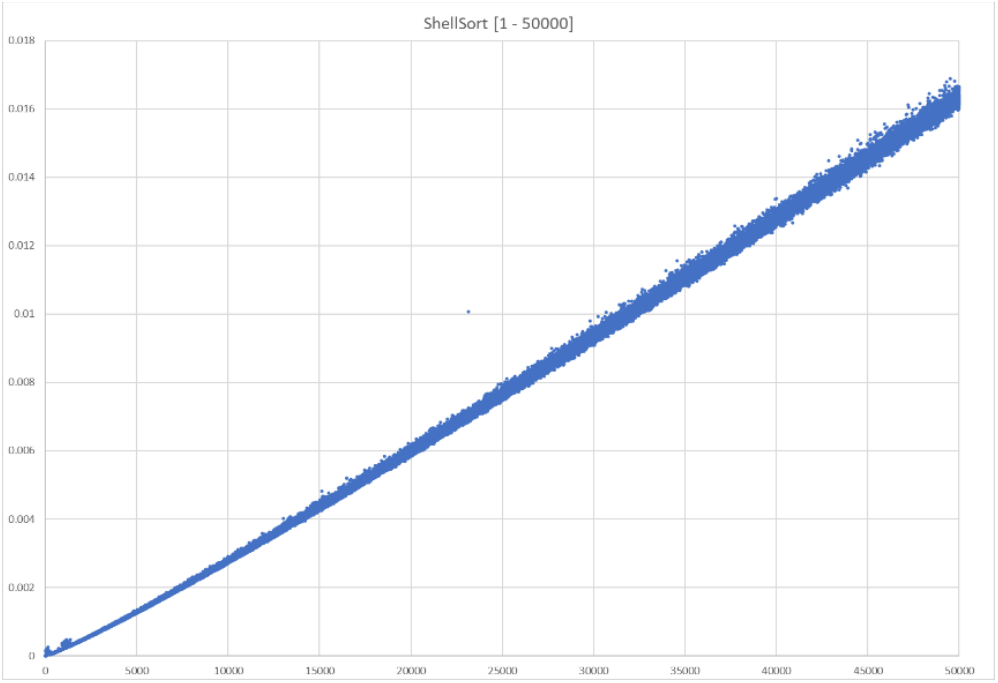
```
> var.test(qs8,ss8, alternative="less")
      F test to compare two variances
data:  qs8 and ss8
F = 0.16854, num df = 100, denom df = 100, p-value < 2.2e-16
alternative hypothesis: true ratio of variances is less than 1
95 percent confidence interval:
 0.000000 0.234555
sample estimates:
ratio of variances
         0.1685361
```

```
> var.test(ss8,bs8, alternative="less")
      F test to compare two variances
data:  ss8 and bs8
F = 0.0065233, num df = 100, denom df = 100, p-value < 2.2e-16
alternative hypothesis: true ratio of variances is less than 1
95 percent confidence interval:
 0.000000000 0.009078552
sample estimates:
ratio of variances
       0.006523263
```

Bisogna sottolineare il fatto che le ipotesi di gaussianità dei campioni, richieste dagli F-test, possono essere considerate rispettate soltanto, al limite, per il secondo dei due. Li abbiamo proposti con scopo puramente indicativo, forti del fatto che il rapporto tra le varianze (indicato nell'ultima riga dell'output) è bassissimo in entrambi i test.

La relazione tra le varianze e la complessità del campione si può spiegare a nostro avviso con la consistente differenza tra i tempi di elaborazione di ciascun algoritmo tenendo conto che il sistema di raccolta dati utilizzato è lontano dall'essere ideale: infatti bisogna ricordare che non è stato possibile riservare tutte le risorse del microcomputer all'esecuzione degli algoritmi di ordinamento e pertanto un più lungo tempo di esecuzione di un algoritmo potrebbe aver aumentato la probabilità di avere altri processi in esecuzione in parallelo che potrebbero aver causato improvvise variazioni (in senso positivo) del tempo di esecuzione; questo spiegherebbe inoltre la forte presenza di code verso destra nella maggior parte dei campioni. Per rafforzare quest'ipotesi si è eseguito un numero molto grande di misure

facendo ordinare in sequenza allo stesso algoritmo array di dimensioni da 1 a 50000. Dal plot dei dati ottenuti (qui sotto) si vede chiaramente come, anche utilizzando sempre lo stesso algoritmo (Shellsort in questo caso), la varianza aumenti con il tempo di esecuzione e di conseguenza con la complessità del problema.



SCRIPT:

Di seguito abbiamo incluso il listato del programma in c++ usato per ottenere i dati. Esso implementa i 3 algoritmi di sorting e due void principali che li chiamano (mainConst() e mainIncrease()) per ottenere sia i campioni aleatori costanti che i campioni usati per i modelli di regressione. I dati ottenuti vengono raccolti in colonna nel file “data.txt”.

```
#include<stdio.h>
#include <time.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string>
#include <unistd.h>

void bubbleSort(int a[]);

timespec diff(timespec start, timespec end);

void quickSort(int a[], int primo, int ultimo);

void shellSort(int arr[], int n);

int SIZE=8000;

int mainConst(void){

    srand(time(NULL));

        std::ofstream fout;

        fout.open("data.txt");


    int a[SIZE];

    for (int k = 0; k <=150; k++){

                for (int z = 0; z < SIZE; z++) {
                        a[z] = rand() % SIZE;
                }

                        struct timespec start, stop;

                        clock_gettime(CLOCK_REALTIME, &start);

                        //bubbleSort(a);

                        quickSort(a,0,SIZE-1);

                        //shellSort(a, SIZE);

                        clock_gettime(CLOCK_REALTIME, &stop);

                        double elTime = (stop.tv_sec - start.tv_sec ) + (double)( stop.tv_nsec - start.tv_nsec
) / (double)1000000000;

                        std::string out = std::to_string(elTime) + "\n";

                        fout << out;

                        std::cout << out;
        }

        fout.close();

}
```

```
int mainIncrease(void){

    srand(time(NULL));

        std::ofstream fout;

        fout.open("data.txt");

    for (int k = 100; k <=15000; k+=100){

                SIZE=k;

                int a[SIZE];

                for (int z = 0; z < SIZE; z++) {
                        a[z] = rand() % SIZE;
                }

                struct timespec start, stop;

                clock_gettime(CLOCK_REALTIME, &start);

                //bubbleSort(a);

                //quickSort(a,0,SIZE-1);

                shellSort(a, SIZE);

                clock_gettime(CLOCK_REALTIME, &stop);

                double elTime = (stop.tv_sec - start.tv_sec ) + (double)( stop.tv_nsec - start.tv_nsec ) /
(double)1000000000;

                std::string out = std::to_string(k) + "; " + std::to_string(elTime) + "\n";

                fout << out;

                std::cout << out;

        }

        fout.close();

}

void quickSort(int a[], int primo, int ultimo) {

  int i, j, pivot, temp;

  if (primo < ultimo) {
    pivot = primo;
    i = primo;
    j = ultimo;

    while (i < j) {
      while (a[i] <= a[pivot] && i < ultimo)
        i++;
      while (a[j] > a[pivot])
        j--;
      if (i < j) {
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
      }
    }

    temp = a[pivot];
    a[pivot] = a[j];
    a[j] = temp;
    quickSort(a, primo, j - 1);
    quickSort(a, j + 1, ultimo);
  }
}


void bubbleSort(int a[]) {
```

```
  int swap = 0;

  for (int c = 0; c < SIZE - 1; c++)
  {
    for (int d = 0; d < SIZE - c - 1; d++)
    {
      if (a[d] > a[d + 1])
      {
        swap = a[d];
        a[d] = a[d + 1];
        a[d + 1] = swap;
      }
    }
  }

}


void shellSort(int arr[], int n)
{

    for (int gap = n/2; gap > 0; gap /= 2)
    {

        for (int i = gap; i < n; i += 1)
        {

            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                arr[j] = arr[j - gap];
                             arr[j] = temp;
        }
    }

}
```