Trioli Davide – S5316731 a.a. 2022/2023

Analisi e Progettazione di Algoritmi

Esercizio 2.1
Lontani dal caso peggiore: Las Vegas QuickSort

Parte 1. ***Implementazione e codice***
L'implementazione dell'algoritmo proposto è stata sviluppata interamente in linguaggio *Python*, per facilitare la comprensibilità del codice e sfruttare la creazione di grafici semplificata dalle apposite librerie esistenti.
Nelle pagine seguenti il codice prodotto, suddiviso per funzionalità implementate.

Sotto: le librerie importate, `matplotlib.pyplot` *per la generazione del grafico dei dati,* `numpy` *per una gestione migliore delle sequenze dei numeri e* `random` *per la generazione di numeri casuali; oltre alle variabili istanziate per tenere traccia di confronti effettuati per* run*, numero di* run *totali e lunghezza delle sequenze.*

```
import matplotlib.pyplot as plt
import numpy as np
import random

############################################################

# global var for comparisons num each run
xnum = 0

# num of runs executed
runs = 100000

# length of each sequence to order
length = 10000

############################################################
```

```python
def LVQuickSort(s):
    qs_aux(s, 0, s.size-1)
    return s

def qs_aux(s, start, end):
    if(start < end):

        # pivot index taken random
        p_ind = random.randint(start, end)

        # switching pivot to start of sequence
        s[start], s[p_ind] = s[p_ind], s[start]
        p_ind = start

        # comparisons for
        i = start+1
        for j in range(i, end+1):

            #comparisons sum
            global xnum
            xnum+=1

            if s[j] <= s[p_ind]:
                s[i], s[j] = s[j], s[i]
                i+=1

        # switching pivot back to its place
        s[p_ind], s[i-1] = s[i-1], s[p_ind]
        p_ind = i-1

        # recursive function callings
        qs_aux(s, p_ind+1, end)
        qs_aux(s, start, p_ind-1)

    return s

############################################################
```

Sopra: l'implementazione effettiva di `LVQuickSort` *tramite funzione ausiliaria* `qs_aux`*, in modo da poter chiamare inizialmente la funzione esterna su una sequenza iniziale in maniera "pulita", priva di dati ulteriori in input utilizzati solo per la realizzazione della ricorsione interna.*

```
# init of empty x axis for values
xvals = np.empty(0, dtype=int)

# init sequence of numbers [1..length]
seq = np.arange(1, length+1, 1)

for i in range(runs):

    # shuffling every seq before ordering
    random.shuffle(seq)

    seq = LVQuickSort(seq)

    # adding result to x axis for graphing later
    xvals=np.append(xvals, xnum)

    xnum = 0

    # console print of completion %
    if(i%(runs/100) == 0):
        print(100*i/runs,"% complete...")


# manual calc of mu and sigma2 not needed since more precise functions already
exist
'''
for i in range(runs):
    mu += xvals[i]
mu = mu/(runs-1)

for i in range(runs):
    sigma2 += ((xvals[i] - mu)**2)
sigma2 = sigma2/(runs-1)
'''
mu = np.average(xvals)
sigma2 = np.var(xvals)
```

Sopra: inizializzazione e randomizzazione di sequenze numeriche da ordinare volta per volta, e salvataggio in apposito array del valore relativo al numero di confronti effettuati; calcolo di μ e σ^2 attraverso funzioni esistenti più ottimizzate rispetto all'implementazione "a mano" (opzione iniziale, commentata).

```python
print("")
print("# # # # # # # STATISTICS # # # # # # #")
print("")
print("- - average num of comparisons µ:", mu)
print("- - variance σ²:", round(sigma2, 5))
print("- - standard deviation σ:", round(sigma2**0.5, 5))
print("- - min num of comparisons:", xvals.min())
print("- - max num of comparisons:", xvals.max())
print("- - comparisons ratio max/µ:", round(xvals.max()/mu, 5))
print("- - comparisons ratio min/µ:", round(xvals.min()/mu, 5))

# inequalities used:
# (6) P{X ≥ vµ} ≤ (1/v)  with v = 2, 3
# (7) P{X ≥ vµ} ≤ (σ²/v²µ²)  with v = 2, 3

pr_count_v1=0
pr_count_v2=0
pr_count_v3=0
pr_count_ms=0
for i in range(runs):
    if(xvals[i] >= mu):
        pr_count_v1+=1
    if(xvals[i] >= (2*mu)):
        pr_count_v2+=1
    if(xvals[i] >= (3*mu)):
        pr_count_v2+=1
    if(xvals[i] <= (mu+(sigma2**0.5)) and xvals[i] >= (mu-(sigma2**0.5))):
        pr_count_ms+=1
```

Sopra: stampe di statistiche ricavate dai dati raccolti durante le esecuzioni precedenti, oltre al calcolo di quante singole run *abbiano effettuato un numero di confronti particolare (sopra 1, 2 o 3 volte il valore atteso* μ *e all'interno dell'intervallo* $[\mu - \sigma, \mu + \sigma]$.

```
pr_v1 = pr_count_v1/runs
print("")
print("-", round(pr_v1*100, 5), "% of runs over µ comparisons")
print("")

pr_v2 = pr_count_v2/runs
print("-", pr_v2*100, "% of runs over 2µ comparisons")
print("- - from Markov inequality should be ≤ 1/2 = 50 %")
print("- - from Chebyshev inequality should be ≤",
round(100*sigma2/(4*(mu**2)), 5), "%")
print("")

pr_v3 = pr_count_v3/runs
print("-", pr_v3*100, "% of runs over 3µ comparisons")
print("- - from Markov inequality should be ≤ 1/3 = 33.33333 %")
print("- - from Chebyshev inequality should be ≤",
round(100*sigma2/(9*(mu**2)), 5), "%")
print("")

pr_ms = pr_count_ms/runs
print("-", pr_ms*100, "% of runs inside µ ± σ")
print("")

print("# # # # # # # # # # # # # # # # # # #")

# graph plotting
plt.grid(linestyle=':', color='#cccccc')
plt.hist(xvals, bins=50, color='#ff5555')
line_mu = plt.axvline(mu, linestyle='--', color='#700918')
line_sigma = plt.axvline(mu+(sigma2**0.5), linestyle=':', color='#dddddd')
plt.axvline(mu-(sigma2**0.5), linestyle=':', color='#dddddd')
plt.xlabel("#comparisons")
plt.ylabel("#occurrences")
plt.title("LVQuickSort - Statistics for " + str(runs) + " runs ordering " +
str(length) + " elements")
plt.legend([line_mu, line_sigma], ["µ", "µ ± σ"])
plt.show()
```

Sopra: stampe relative a statistiche quantitative precedentemente calcolate e plotting dell'istogramma corrispondente ai dati raccolti.

Parte 2. *Risultati e statistiche*

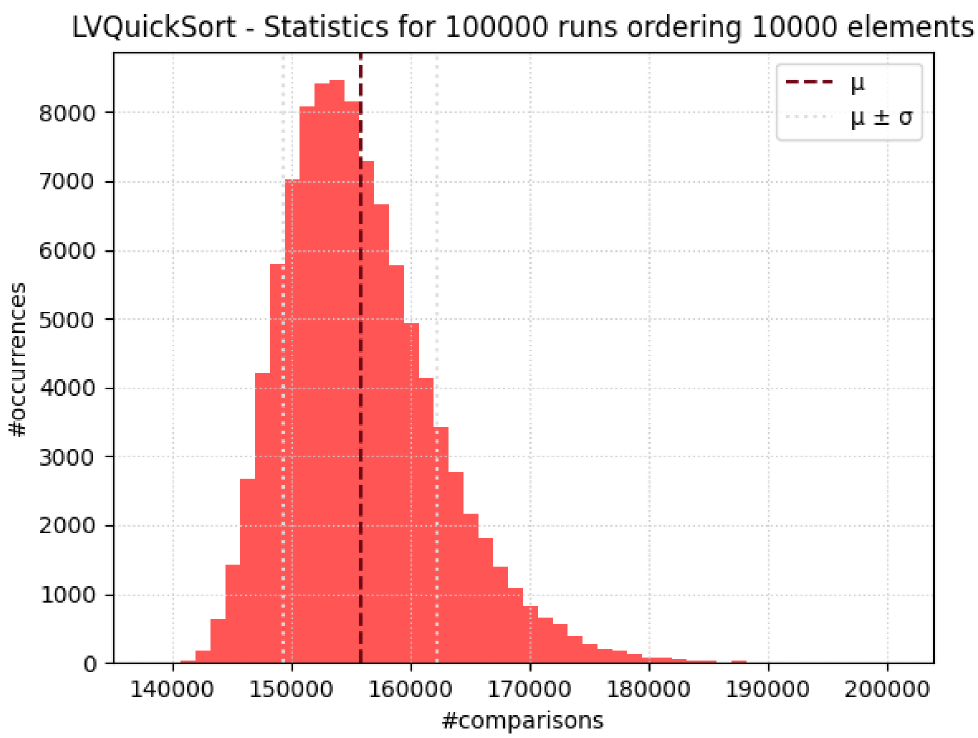
I risultati prodotti per una esecuzione di 10^5 ordinamenti su sequenze randomizzate volta per volta lunghe 10^4 elementi sono, direttamente dall'output del codice, i seguenti:

```
# # # # # # # STATISTICS # # # # # # #

- - average num of comparisons μ: 155768.11339
- - variance σ²: 42013275.83299
- - standard deviation σ: 6481.76487
- - min num of comparisons: 138100
- - max num of comparisons: 200786
- - comparisons ratio max/μ: 1.28901
- - comparisons ratio min/μ: 0.88657

- 44.157 % of runs over μ comparisons

- 0.0 % of runs over 2μ comparisons
- - from Markov inequality should be ≤ 1/2 = 50 %
- - from Chebyshev inequality should be ≤ 0.04329 %

- 0.0 % of runs over 3μ comparisons
- - from Markov inequality should be ≤ 1/3 = 33.33333 %
- - from Chebyshev inequality should be ≤ 0.01924 %

- 70.341 % of runs inside μ ± σ

# # # # # # # # # # # # # # # # # # #
```

Riassumendo quindi si ha:

- $\hat{\mu} \approx 155768.1$
- $\hat{\sigma}^2 \approx 42013275.8$
  $\Leftrightarrow \hat{\sigma} \approx 6481.8$

Inoltre, graficamente si visualizza una situazione come quella illustrata qui sulla destra, dove le barre (*bins*) color salmone rappresentano quante istanze (asse y) di `LVQuickSort` hanno effettuato un certo numero di confronti (asse x); come da legenda, sono state graficate anche una linea tratteggiata color porpora rappresentante il valor medio di comparazioni per esecuzione e due linee punteggiate grigie rappresentanti la deviazione standard (calcolata come radice quadrata della varianza) a partire dal valor medio.



Successivamente, come indicato nell'output, utilizzando le disuguaglianze proposte

(6) $$\Pr\{X \geq v\mu\} \leq \frac{\mu}{v\mu} = \frac{1}{v}$$ e (7) $$\Pr\{X \geq v\mu\} \leq \frac{\sigma^2}{v^2\mu^2}$$

e implementando i risultati empirici sopra descritti per $v = 2$ e $v = 3$, si verifica che:

- Per $v = 2$:
    - Per la disuguaglianza (6) si ha che $Pr\{ X \geq 2\mu \} \leq \frac{1}{2}$
    - Per la disuguaglianza (7) si ha che $Pr\{ X \geq 2\mu \} \lesssim \frac{42013275.8}{2^2(155768.1^2)} \approx 0.00043$
- Per $v = 3$:
    - Per la disuguaglianza (6) si ha che $Pr\{ X \geq 3\mu \} \leq \frac{1}{3}$
    - Per la disuguaglianza (7) si ha che $Pr\{ X \geq 3\mu \} \lesssim \frac{42013275.8}{3^2(155768.1^2)} \approx 0.00019$

In effetti, sebbene l'informazione ottenuta attraverso la disuguaglianza (7) sia molto più elevata poiché sensibilmente più rivelativa, in entrambi i casi le probabilità così calcolate vengono rispettate in quanto l'algoritmo non effettua mai più di 2μ confronti: si ferma infatti a un massimo di circa 1.29 volte il valore medio (come indicato nei risultati statistici).

Altri dati interessanti che emergono sono ad esempio:

- Un limite inferiore a quanti confronti l'algoritmo esegue, qui sperimentalmente pari a circa 0.89μ;
- Il numero di esecuzioni che effettuano un numero di comparazioni superiore alla media, circa 44%, cioè meno della metà: la maggioranza delle *run* eseguirà quindi un numero di confronti più vantaggioso;
- La percentuale di *run* con un numero di confronti all'interno del range $[\mu - \sigma, \mu + \sigma]$ ossia l'intervallo vicino una deviazione standard dal valor medio, del 70.3% circa: più di 7 volte su 10 quindi lo scostamento dal valore atteso è molto basso, rendendo quindi l'algoritmo molto affidabile computazionalmente.