LVQuickSort: considerazioni

Di Giampietro Andrea, s5208458

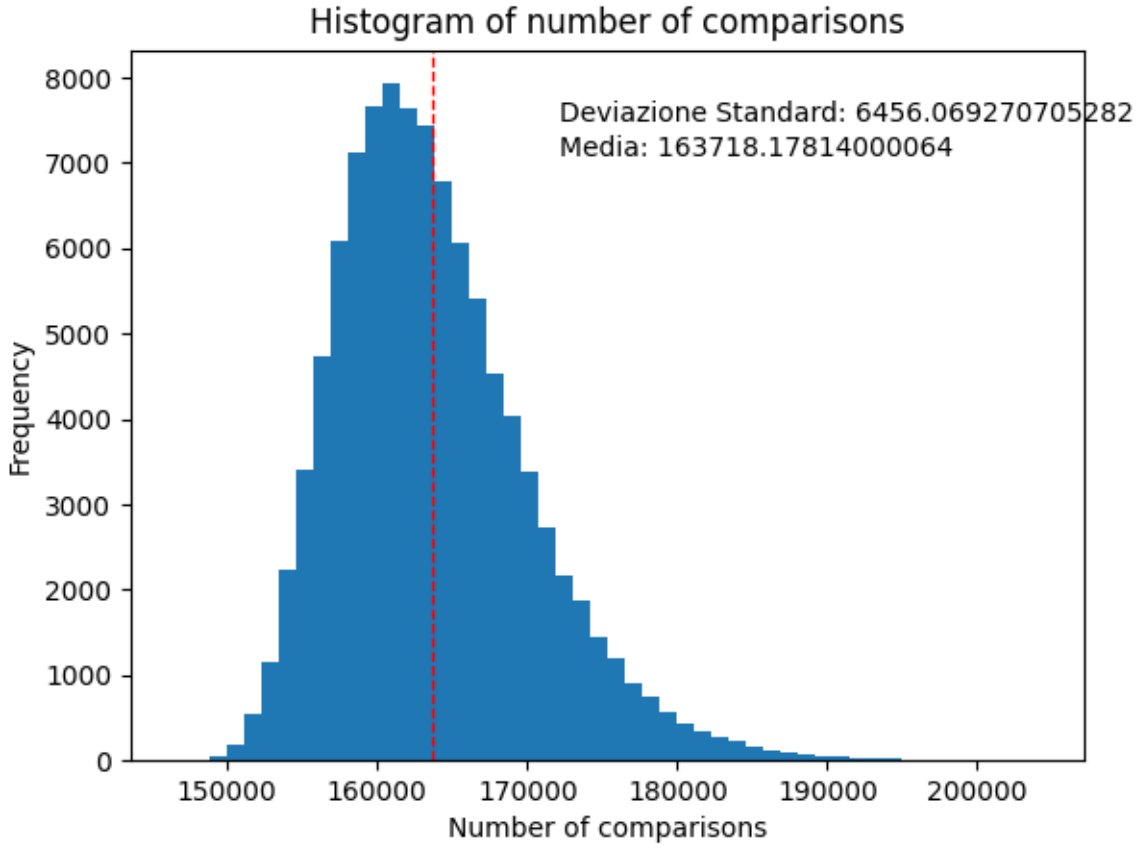
La richiesta

Compito 2.1. *Implementazione di LVQuickSort*
Costruisci una sequenza S di numeri con $|S| = 10^4$. Implementa *LVQuickSort* e conta il numero X_r di confronti effettuati in ogni singolo *run* r per ordinare la sequenza S. Calcola il valore medio $\hat{\mu}$ e la deviazione standard empirica $\hat{\sigma}$ usando le formule

$$\hat{\mu} = \frac{1}{R}\sum_{r=1}^{R} X_r \quad \text{e} \quad \hat{\sigma}^2 = \frac{1}{R-1}\sum_{r=1}^{R}(X_r - \hat{\mu})^2$$

del numero di confronti effettuati su $R = 10^5$ *run*. Produci un istogramma di 50 bin con i valori ottenuti. Limita dall'alto la probabilità con la quale *LVQuickSort* effettua il doppio e il triplo del valore atteso dei confronti mediante le disuguaglianze (6) e (7) con $\hat{\mu}$ e $\hat{\sigma}$ al posto di μ e σ. Confronta la frequenza empirica con la quale hai ottenuto il doppio e il triplo di $\hat{\mu}$ con i limiti delle disuguaglianze (6) e (7). **Commenta i risultati che ottieni**.

I dati ottenuti



Considerazioni sui dati ottenuti

L'algoritmo LVQuickSort dimostra un'elevata efficienza nel mantenere il numero di confronti vicino al valore medio nella maggior parte dei casi. I risultati ottenuti indicano che, in media, l'algoritmo esegue circa 163.718 confronti, con una deviazione standard di circa 6.456.

Utilizzando le disuguaglianze di Chebyshev, possiamo stimare la probabilità che il numero di confronti superi il doppio e il triplo del valore medio. In entrambi i casi, la probabilità calcolata è estremamente bassa. Per il caso in cui il numero di confronti supera il doppio del valore medio, la probabilità è inferiore o uguale a 0,001494. Mentre per il caso in cui il numero di confronti supera il triplo del valore medio, la probabilità è inferiore o uguale a 0,000373: ciò fornisce una stima superiore delle probabilità e ci consente di valutare l'efficacia dell'algoritmo.

I risultati empirici ottenuti sono coerenti con i limiti teorici previsti, indicando che l'algoritmo offre performance affidabili nella gestione dei confronti.

Questi dati indicano che l'algoritmo LVQuickSort rimane coerentemente all'interno delle aspettative. Situazioni in cui si verificano un numero significativamente superiore di confronti rispetto al valore medio sono estremamente rare. Ciò conferma l'efficienza e la stabilità dell'algoritmo nella gestione di array di grandi dimensioni.

Il codice

```
#QUICKSORT WITH RANDOM PIVOT
#COUNTING COMPARISONS
import random
import matplotlib.pyplot as plt
import numpy as np

# Global variable to count the number of comparisons
counter = 0
# Number of times quicksort is run
RUN = 100000

# Function to partition the array on the basis of pivot element
def partition(array, low, high):
    global counter
    # Select the pivot element
    pivot = array[random.randint(low, high)]
    i = low - 1
    # Put the elements smaller than pivot on the left and greater than pivot on the right of pivot
    for j in range(low, high + 1):
        counter += 1
        if array[j] < pivot:
            i += 1
```

```
            array[i], array[j] = array[j], array[i]
    array[i + 1], array[high] = array[high], array[i + 1]
    return (i + 1)

# Generates a 10K array of random numbers
def generateArray():
    array = []
    for i in range(10000):
        array.append(random.randint(0, 10000))
    return array

# Function to perform quicksort
def quickSort(array, low, high):
    if low < high:
        # Select pivot position and put all the elements smaller
        # than pivot on left and greater than pivot on right
        pi = partition(array, low, high)
        # Sort the elements on the left of pivot
        quickSort(array, low, pi - 1)
        # Sort the elements on the right of pivot
        quickSort(array, pi + 1, high)


# Function to calculate the avarage value of comparisons
def average(comparisons):
    sum = 0
    for i in range(RUN):
        sum += comparisons[i]/RUN
    return sum


# Function to calculate the standard deviation of comparisons
def standardDeviation(comparisons):
    sum = 0
    for i in range(RUN):
        sum += (((comparisons[i] - average(comparisons)) ** 2 )/RUN)
    return (sum) ** 0.5


#Function to produce an histogram of the number of comparisons - 50 bins
def histogram(comparisons):
    bins = [0] * 50
    for i in range(RUN):
        bins[comparisons[i] // 100] += 1
    return bins


#Function to calculate the chebyshev's inequality
def chebyshev(avg, SD, k):
    n = pow(SD, 2) / (pow(k-1, 2) * pow(avg, 2))
```

```
    return n if n < 1 else 1

# Driver code for RUN runs
if __name__ == '__main__':

    # Create an empty array to store the number of comparisons
    comparisons = []
    a = generateArray()

    for i in range(RUN):
        # Reset the counter
        counter = 0
        array = generateArray()
        array = a.copy()
        n = len(array)
        quickSort(array, 0, n - 1)

        # Store the number of comparisons
        comparisons.append(counter)


    avg = average(comparisons)
    # Print the avaerage number of comparisons
    print("Average number of comparisons: ", avg)
    # Print the standard deviation of comparisons
    SD = standardDeviation(comparisons)
    print("Standard deviation of comparisons: ", SD)
    # Print the Chebyshev's inequality for k = 2
    print("Chebyshev's inequality: ", chebyshev(avg, SD, 2))
    # Print the Chebyshev's inequality for k = 3
    print("Chebyshev's inequality: ", chebyshev(avg, SD, 3))

    # Hisogram
    hist, bins = np.histogram(comparisons, bins=50)

    # Printing
    plt.hist(comparisons, bins=50)
    plt.xlabel('Number of comparisons')
    plt.ylabel('Frequency')
    plt.title('Histogram of number of comparisons')

    #Avg and sd
    plt.axvline(avg, color='r', linestyle='dashed', linewidth=1)
    plt.annotate(f'Deviazione Standard: {SD}', xy=(0.45, 0.9), xycoords='axes fraction')
    plt.annotate(f'Media: {avg}', xy=(0.45, 0.85), xycoords='axes fraction')

    plt.show()
    plt.axvline(avg, color='r', linestyle='dashed', linewidth=1)
```