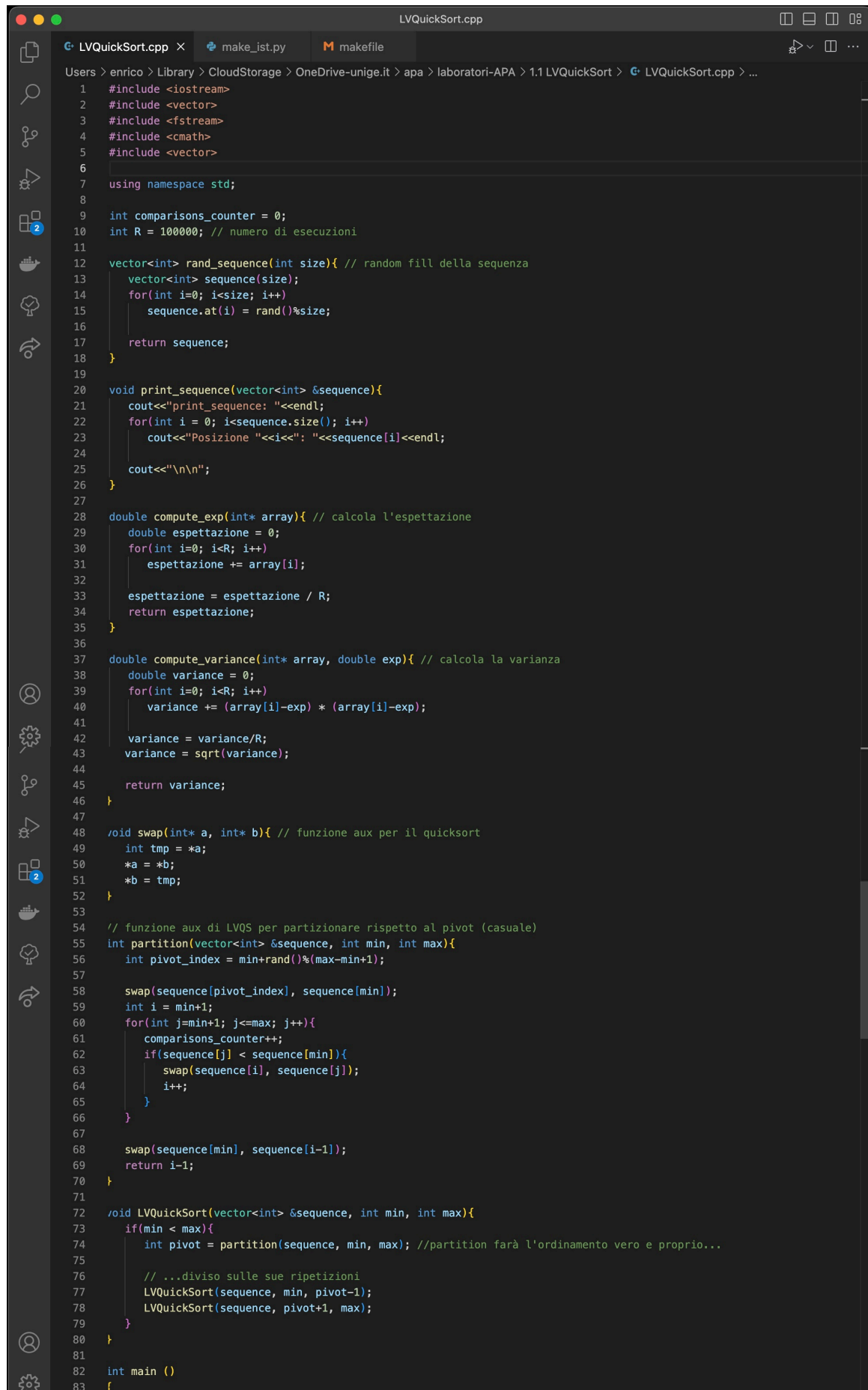


# Relazione LasVegas QuickSort

Pezzano Enrico



```
LVQuickSort.cpp
Users > enrico > Library > CloudStorage > OneDrive-unige.it > apa > laboratori-APA > 1.1 LVQuickSort > LVQuickSort.cpp > ...
1  #include <iostream>
2  #include <vector>
3  #include <fstream>
4  #include <cmath>
5  #include <vector>
6
7  using namespace std;
8
9  int comparisons_counter = 0;
10 int R = 100000; // numero di esecuzioni
11
12 vector<int> rand_sequence(int size){ // random fill della sequenza
13     vector<int> sequence(size);
14     for(int i=0; i<size; i++)
15         sequence.at(i) = rand()%size;
16
17     return sequence;
18 }
19
20 void print_sequence(vector<int> &sequence){
21     cout<<"print_sequence: "<<endl;
22     for(int i = 0; i<sequence.size(); i++)
23         cout<<"Posizione "<<i<<": "<<sequence[i]<<endl;
24
25     cout<<"\n\n";
26 }
27
28 double compute_exp(int* array){ // calcola l'aspettazione
29     double aspettazione = 0;
30     for(int i=0; i<R; i++)
31         aspettazione += array[i];
32
33     aspettazione = aspettazione / R;
34     return aspettazione;
35 }
36
37 double compute_variance(int* array, double exp){ // calcola la varianza
38     double variance = 0;
39     for(int i=0; i<R; i++)
40         variance += (array[i]-exp) * (array[i]-exp);
41
42     variance = variance/R;
43     variance = sqrt(variance);
44
45     return variance;
46 }
47
48 void swap(int* a, int* b){ // funzione aux per il quicksort
49     int tmp = *a;
50     *a = *b;
51     *b = tmp;
52 }
53
54 // funzione aux di LVQS per partizionare rispetto al pivot (casuale)
55 int partition(vector<int> &sequence, int min, int max){
56     int pivot_index = min+rand()%(max-min+1);
57
58     swap(sequence[pivot_index], sequence[min]);
59     int i = min+1;
60     for(int j=min+1; j<=max; j++){
61         comparisons_counter++;
62         if(sequence[j] < sequence[min]){
63             swap(sequence[i], sequence[j]);
64             i++;
65         }
66     }
67
68     swap(sequence[min], sequence[i-1]);
69     return i-1;
70 }
71
72 void LVQuickSort(vector<int> &sequence, int min, int max){
73     if(min < max){
74         int pivot = partition(sequence, min, max); //partition farà l'ordinamento vero e proprio...
75
76         // ...diviso sulle sue ripetizioni
77         LVQuickSort(sequence, min, pivot-1);
78         LVQuickSort(sequence, pivot+1, max);
79     }
80 }
81
82 int main ()
83 {
```

```
84 cout << "LVQuickSort by Enrico Pezzano."<<endl<<endl;
85 srand(time(NULL));
86
87 int size = 10000; // dimensione delle sequenze
88 int comparisons_array[R]; // array per il numero dei confronti per run
89
90 for(int i=0; i<R; i++){
91     // sequenza di numeri interi con |S|=10^4; una per ogni run
92     vector<int> S = rand_sequence(size);
93
94     // chiamo LVQuickSort, tenendo conto del numero di confronti effettuati in ogni run
95     LVQuickSort(S,0,size-1);
96
97     comparisons_array[i] = comparisons_counter;
98     comparisons_counter = 0;
99 }
100
101 // calcolo il valore medio del numero di confronti
102 double middle_comparison_value = compute_exp(comparisons_array);
103 cout<<"Il valore atteso del numero dei confronti è: "<<middle_comparison_value<<endl;
104
105 // calcolo la varianza del numero di confronti
106 double comparisons_variance = compute_variance(comparisons_array, middle_comparison_value);
107 cout<<"La varianza del numero dei confronti è: " << comparisons_variance<<"\n\n";
108
109 // output file filling for comparisons
110 ofstream comparison;
111 comparison.open ("comparison.dat");
112
113 for(int i=0; i<R; i++)
114     comparison << comparisons_array[i] << endl;
115
116 comparison.close();
117
118 // output file filling dell'aspettazione
119 ofstream aspettazione;
120 aspettazione.open ("aspettazione.dat");
121 aspettazione << middle_comparison_value << endl;
122 aspettazione.close();
123
124 // stima empiricamente la probabilita con la quale LVQS effettua il doppio e il triplo del valore medio dei confronti
125 int comparisons_double = 0;
126 int comparisons_triple = 0;
127
128 for(int i=0; i<R; i++){
129     if(comparisons_array[i] >= middle_comparison_value * 2)
130         comparisons_double++;
131
132     if(comparisons_array[i] >= middle_comparison_value * 3)
133         comparisons_triple++;
134 }
135
136 cout << "Stima empirica del numero di volte in cui LVQuickSort effettua: " << endl;
137 cout << "- Il doppio del valore medio dei confronti: " << comparisons_double << endl;
138 cout << "- Il triplo del valore medio dei confronti: " << comparisons_triple << endl<<endl;
139
140 return 0;
141 }
```

```
make_ist.py
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # prendo il valore atteso
5 e = open('aspettazione.dat', 'r')
6
7 e.seek(0)
8 exp = int(e.readline())
9
10 # prendo i confronti
11 f = open('comparison.dat', 'r')
12
13 # creerò un unico file con 50 bin
14 binCounts = [50]
15
16 # ciclo da 1 a 50
17 for binCount in binCounts:
18     values = []
19     f.seek(0)
20
21     # riempio values
22     for line in f.readlines():
23         values.append(int(line))
24
25     n, bins, patches = plt.hist(values, bins=binCount)
26
27     # guardo dove si trova il valore di aspettazione
28     # mi fermo quando l'aspettazione è nel bin giusto
29     # bins[0] limite sx del primo bin; bin[1] limite destro
30     # per questo parto a contare da count=-1, poiché se ad esempio mi trovasse
31     # nel primo bin entrerebbe nell'if una volta (perché è maggiore del limite sx di esso)
32     # alla fine count dovrebbe essere 0
33     count = -1;
34     for mayExp in bins:
35         if(exp > mayExp):
36             count = count + 1
37
38     plt.gca().set(title='Istogramma LVQuickSort', xlabel='Numero di Confronti', ylabel='Frequenza')
39     plt.savefig('Istogramma.png')
40     plt.close()
```

```
makefile
1 all:
2     g++ LVQuickSort.cpp -o startLVQS
3
4     @echo "Avvio il programma, la sequenza verrà ordinata..."
5
6     ./startLVQS
7
8     @echo "Creo gli istogrammi..."
9
10    python3 make_ist.py
11
12    @echo "Programma terminato."
13
14 clean:
15    rm -f Istogramma.png comparison.dat aspettazione.dat startLVQS
```

Il programma implementato per questo compito sfrutta l'algoritmo Quicksort di tipo Las Vegas per ricavare una serie di misurazioni del numero di confronti necessari ad ordinare una sequenza di 10 mila elementi.

In particolare è stato eseguito per 100 mila run, calcolando una buona stima del valore atteso, che si attesta a circa 156044.

Possiamo confermare che l'algoritmo ha complessità  $\Theta(n \log n)$ , in particolare il dato si avvicina a  $4 \cdot n \log n$ ; circa 160 mila confronti, maggiore il loro numero, maggiore sarà la trascurabilità del caso peggiore  $\Theta(n^2)$ .

Osservando l'istogramma sottostante possiamo notare come la grande maggioranza delle esecuzioni abbia generato un risultato molto vicino al valore atteso; lo si può dedurre anche dalla varianza, che è stata stimata a circa 6481.76.

Infine il programma effettua una stima empirica del numero di volte in cui LVQuickSort effettua il doppio e il triplo del valore medio dei confronti; in entrambi i casi il programma ha restituito 0. Per ottenere un risultato diverso si dovrebbe eseguire il programma per un numero di volte ancora più grande.

Di seguito uno screenshot delle stampe a console del programma e l'istogramma costituito da 50 bin, ottenuti dai calcoli dall'algoritmo.

```
LVQuickSort by Enrico Pezzano.
```

```
Il valore atteso del numero dei confronti è: 156044
```

```
La varianza del numero dei confronti è: 6481.76
```

```
Stima empirica del numero di volte in cui LVQuickSort effettua:
```

```
- Il doppio del valore medio dei confronti: 0
```

```
- Il triplo del valore medio dei confronti: 0
```

