Compito 2.1

Il compito riguarda l'implementazione di un algoritmo Quick Sort con selezione del pivot randomica, in modo da renderlo di tipo Las Vegas.
"Mescolare le carte" secondo quanto visto a lezione dovrebbe difenderci dalla possibilità che determinati input negativi influenzino negativamente la complessità temporale dell'esecuzione.
In particolare, in questo compito gli esperimenti sono stati fatti su una sequenza disordinata casuale da 10000 numeri eseguendo 100000 volte il programma.

Il Codice:

Per la codifica dell'algoritmo ho scelto il linguaggio c++, utilizzando termini in inglese per il codice vero e proprio e commenti in italiano.
La funzione a cui ho affidato il compito di fare da "dealer" delle nostre carte è rand() della stdlib di c++.
Per depositare i risultati ho deciso di scriverli su due file separati: uno contenente il numero di confronti ogni run e uno contenente valore medio e deviazione standard.
Ho realizzato il LVQuickSort applicando lo pseudocodice contenuto nelle note:

```
vector<int> LVQuickSort(vector<int> list){
    if(list.size() <= 1){
        return list;
    }
    int pivotIndex = rand() % list.size();
    int pivot = list[pivotIndex];
    vector<int> left;
    vector<int> right;
    for(int i = 0; i < list.size(); ++i){
        compCount++;
        if(i == pivotIndex){
            continue;
        }
        if(list[i] < pivot){
            left.push_back(list[i]);
        } else {
            right.push_back(list[i]);
        }
    }
```

```
    left = LVQuickSort(left);
    right = LVQuickSort(right);
    left.push_back(pivot);
    left.insert(left.end(), right.begin(), right.end());
    return left;
}
```

Per gestire le diverse esecuzioni, ho reputato più interessante l'analisi del numero di confronti tenendo la lista da ordinare sempre uguale, variando randomicamente solo la selezione del pivot:

```
void ManySorts(vector <int> list){

    //Preparazione File Output
    ofstream outputFile(outputFileName);
    if(!outputFile.is_open()){
        cerr << "error opening Output file";
        return;
    }

    //Preparazione lista
    vector<int> backupList = list;

    for(int i=0; i < RunNumber; ++i){
        compCount = 0;
        list = LVQuickSort(list);
        outputFile << compCount << endl;
        list = backupList;
    }
}
```

Il programma ha infine calcolato valore medio e deviazione standard con le seguenti funzioni:

```
float averageValue(){
    ifstream inputFile(outputFileName);
    if(!inputFile.is_open()){
        cerr << "error opening Output file";
        return -1;
    }
```

```
    float sum = 0;
    int temp;

    for(int i = 0; i < RunNumber; ++i){
        inputFile >> temp;
        sum += temp;
    }

    int avgVal = sum/RunNumber;

    ofstream outputFile("values.txt", ios::app);
    if(!outputFile.is_open()){
        cerr << "error opening Output file";
        return -1;
    }
    outputFile << "Average number of comparisons: " << avgVal << endl;
    return avgVal;
}

float empiricalVariance(float avgVal){
    ifstream inputFile(outputFileName);
    if(!inputFile.is_open()){
        cerr << "error opening Output file";
        return -1;
    }

    float sum = 0;
    int temp;
    for(int i = 0; i < RunNumber; ++i){
        inputFile >> temp;
        sum +=(temp - avgVal)*(temp - avgVal);
    }

    float empVariance = sum/(RunNumber-1);

    ofstream outputFile("values.txt", ios::app);
    if(!outputFile.is_open()){
        cerr << "error opening Output file";
        return -1;
```

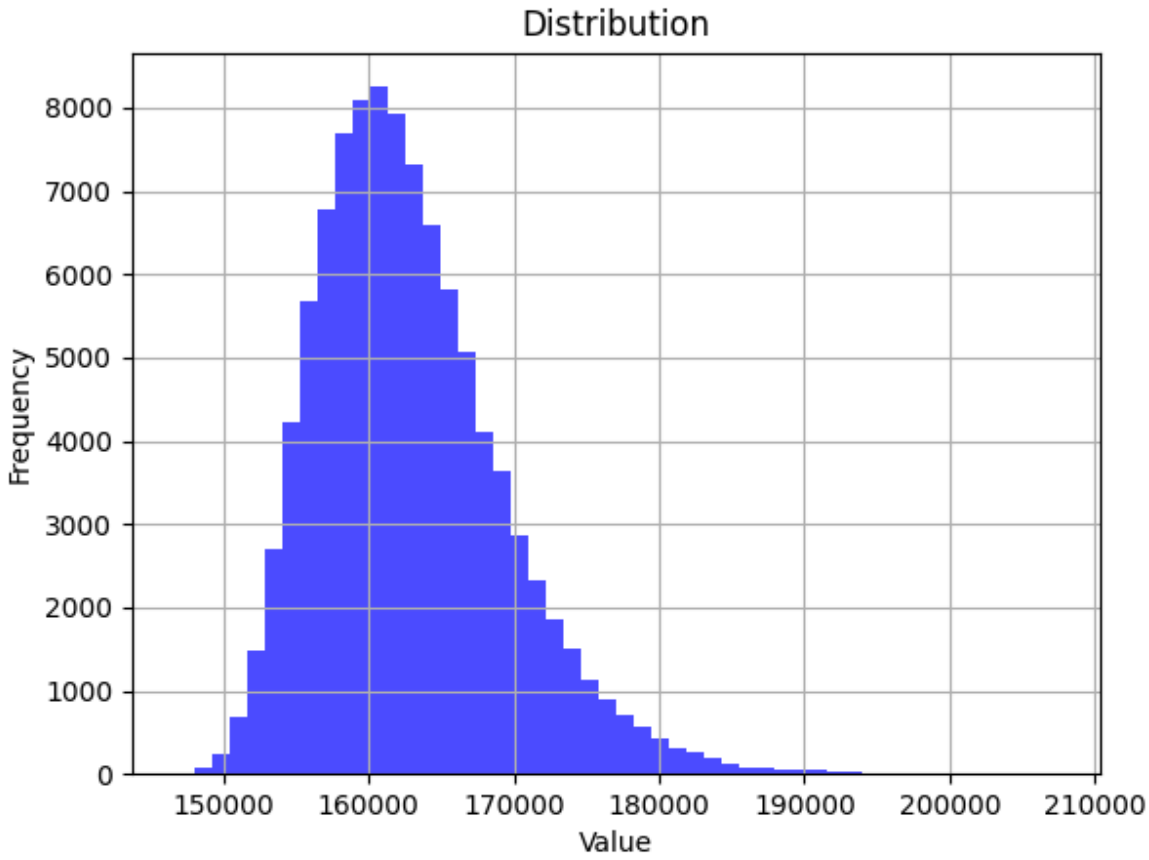
```
    }
    outputFile << "Empirical variance: " << empVariance << endl;
    outputFile << "Standard Deviation: " << sqrt(empVariance) << endl;
    return empVariance;

}
```

Il main ha semplicemente settato srand() e collegato i pezzi (salto la generazione della lista poichè poco rilevante):

```
int main(){
    srand(time(NULL));
    vector<int> list = generateList();
    cout << "Original List: ";
    printList(list);
    cout << endl;
    ManySorts(list);
    float avgVal = averageValue();
    cout << "Average number of comparisons: " << avgVal << endl;
    float empVar = empiricalVariance(avgVal);
    cout << "Empirical variance: " << empVar << endl;
}
```

I risultati e il grafico:

L'esecuzione del programma ha calcolato un valore medio di 162812 comparazioni e una deviazione standard di 6455.31,che ho rappresentato in questo istogramma:



Per valutare questi risultati, vediamo quanto è la probabilità di ottenere in un'esecuzione particolarmente sfortunata un valore pari al doppio del valore medio.

$$\Pr\{X \geq v\mu\} \leq \frac{\mu}{v\mu} = \frac{1}{v} \qquad \Pr\{X > v\mu\} < \frac{\sigma^2}{(v-1)^2\mu^2}$$

Sostituiamo quindi μ con il valore medio, σ^2 con la varianza (41671027,1961) e v con 2. Nel primo caso viene che la probabilità è sicuramente inferiore a ½, mentre nel secondo viene che è minore di 41671027,1961 / 26.507.747.344 ovvero 0,00157203200465588382137402946 57.

Queste disuguaglianze forniscono indicazioni comunque molto blande, come è possibile intuire dal grafico infatti la frequenza empirica con cui il numero di confronti raggiunge il doppio o il triplo del valore atteso è pari a 0.

Essendo il valore medio 162812 e così bassa la probabilità che venga effettuato un numero di comparazioni anche pari solamente al doppio, possiamo confermare che siamo ben lontani dal caso peggiore dell'algoritmo non randomizzato (n^2).
Confrontando questo valore medio con il calcolo della complessità nel caso migliore (circa 2 n log n) di quick sort deterministico otteniamo un risultato interessante: la complessità (2 n log n) con $n = 10^4$ corrisponde a $2 * 10^4 * \log_2 10^4 \approx 2*10^4 * 12 = 240000$, il quale è superiore al valore medio ottenuto dai nostri esperimenti.
Dal grafico è inoltre evidente come i vari valori assunti durante le diverse esecuzioni si scostino poco da questo valore calcolato e che quindi rendere l'algoritmo di tipo Las Vegas ci assicuri di ottenere un costo computazionale contenuto, molto simile a quello del caso migliore dell'algoritmo deterministico, mentre i casi in cui il numero di comparazioni si allontana dal centro sono irrisori, oltre che notevolmente distanti dal caso peggiore dell'algoritmo deterministico, il quale valore per essere rappresentato dal grafico di sopra richiederebbe una grandissima variazione della scala.