Algoritmi Insertion Sort e Bubble Sort

Andrea Ercoli, Filippo Ricotti, Lorenzo Francia, Michele Greco, Nicola Suzzi, Tommaso Brini

June 5, 2023

Introduzione

Scopo dell'elaborato è stato testare le principali differenze tempistiche fra gli algoritmi di ordinamento di Array *Insertion Sort* e *Bubble Sort*

Insertion Sort

```
public static long insertionSort(int[] array) {
        c = 0;
         long start = System.nanoTime();
         for (int i = 1; i < array.length; i++) {
              int value = array[i];
              int j = i - 1;
              while (j >= 0 && array[j] > value) {
                   array[j+1] = array[j];
                   j--;
                   array[j + 1] = value;
                   c++;}
         }
         long end = System.nanoTime();
         end = end - start;
         System.out.println("Numero di scambi "+ c);
                   return end; }
```

Bubble Sort

```
public static long bubbleSort(int[] array){
        c = 0;
        boolean scambio = true;
        long start = System.nanoTime();
        while(scambio){
            scambio = false;
            for(int i=0; i<array.length-1; i++){
                if(array[i] > array[i+1]){
                    int value = array[i];
                    array[i] = array[i+1];
                    array[i+1] = value;
                    scambio = true;
                    c++;}}}
        long end = System.nanoTime();
        end = end - start;
        System.out.println("Numero di scambi " + c);
        return end;}
```

Nel Main

Abbiamo generato arrays randomici di dimensioni variabili

```
final int smallDimension = 100;
final int midDimension = 1000;
final int bigDimension = 100000;

int[] arraySmall = randomArray(smallDimension);
int[] arrayMid = randomArray(midDimension);
int[] arrayBig = randomArray(bigDimension);
int[] arrayMidWorst = SpecialCase(midDimension, caseSet: false);
int[] arrayMidBest = SpecialCase(midDimension, caseSet: true);
```

```
public static int[] randomArray(int size) {
    int[] array = new int[size];
    Random random = new Random();
    for (int i = 0; i < size; i++) {
        array[i] = random.nextInt();
    }
    return array;
}
```

Conclusioni

Abbiamo osservato che

- in generale è più efficiente *Insertion Sort* di *Bubble Sort*
- nel caso migliore e peggiore per arrays di dimensioni "medie" *Boubble Sort* è più efficiente di *Insertion Sort*
- tramite un contatore dei passaggi necessari per riordinare gli arrays abbiamo verificato che il numero di passaggi è lo stesso per entrambi gli algoritmi

  Abbiamo stimato l'efficienza tramite un rapporto tra i tempi di esecuzione che riportiamo di seguito

```
Insertion Sort
Numero di scambi 2410
-Small dimension time = 318267
Numero di scambi 253741
-Mid dimension time = 50758407
Numero di scambi 499500
-Worst Mid dimension time = 10489112
Numero di scambi 0
-Best Mid dimension time = 8800
Numero di scambi 2497397845
-Big dimension time = 4027745356
```

```
 BubbleSort
Numero di scambi 2410
-Small dimension time = 910311
Numero di scambi 253741
-Mid dimension time = 30706138
Numero di scambi 499500
-Worst Mid dimension time = 9743068
Numero di scambi 0
-Best Mid dimension time = 7822
Numero di scambi 2497397845
-Big dimension time = 43574072822
```