



UNIVERSITÀ  
degli STUDI  
di CATANIA

# **Algoritmi di ordinamento: Bubble sort, Selection sort, Insertion sort**

Corso di programmazione I (A-E / O-Z) AA 2025/26

Corso di Laurea Triennale in Informatica

---

Fabrizio Messina

[fabrizio.messina@unict.it](mailto:fabrizio.messina@unict.it)

Dipartimento di Matematica e Informatica

Il problema dell'ordinamento consiste nel sistemare gli elementi dello array in un preciso ordine. Ad esempio:

- ordinamento crescente;
- ordinamento decrescente.

Algoritmi di ordinamento:

- **Bubblesort**
- **SelectionSort**
- **InsertionSort**
- QuickSort
- MergeSort
- HeapSort
- ShellSort

Si chiama così perchè gli elementi dello array raggiungono la giusta posizione nell'ordinamento finale come fossero *bolle* che salgono in superficie. **Ordinamento crescente:**

- si effettua un certo numero di **visite dell'intero array**.
- Ad ogni visita si **confrontano coppie di elementi** contigui:
  - se il primo valore è maggiore del secondo, la loro posizione sarà invertita **swap**;
- Se durante una visita **non avviene alcuno scambio**, ciò significa che **l'array è ordinato**, dunque lo algoritmo può terminare.

# BubbleSort

Implementazione del BubbleSort.

Flag *swapped* utile a capire se vanno fatti altri cicli.

```
1    bool swapped = true;
2    while(swapped){
3        swapped = false;
4        for(int i = 0; i < length - 1; i++ )
5            if ( array[i] > array[i+1] ){
6                swap( array, i, i+1 );
7                swapped = true;
8            }
9    }
```

**Esempio di output** (le parentesi quadre indicano le coppie di elementi in fase di confronto ed eventualmente di scambio/swap).

1: [8 4] 6 1 2 7 5 3

1: 4 [8 6] 1 2 7 5 3

1: 4 6 [8 1] 2 7 5 3

1: 4 6 1 [8 2] 7 5 3

1: 4 6 1 2 [8 7] 5 3

1: 4 6 1 2 7 [8 5] 3

1: 4 6 1 2 7 5 [8 3]

2: [4 6] 1 2 7 5 3 8

In pratica, nel BubbleSort, **dopo la k-esima visita dello array**, il k-esimo elemento più grande trova il giusto posto nello ordinamento finale.

1: [8 4] 6 1 2 7 5 3

1: 4 [8 6] 1 2 7 5 3

1: 4 6 [8 1] 2 7 5 3

...

2: [4 6] 1 2 7 5 3 8

**Worst case:**  $(n - 1) \cdot (n - 1)$  iterazioni.

Esempi svolti

`bubble_sort.c` – Bubblesort.

Implementare una versione ottimizzata del BubbleSort sulla base della precedente osservazione:

*dopo la  $k$ -esima visita dello array, il  $k$ -esimo elemento più grande trova il giusto posto nello ordinamento finale.*

- ricerca il minimo dello intero array ( $A[0 \dots n-1]$ )
- scambia il minimo con l'elemento di posto zero;
- ricerca il minimo nel sottoarray  $A[1 \dots n-1]$ ;
- scambia il minimo con l'elemento di posto 1;
- ...

## SelectionSort

```
1  for (int index = 0; index < length - 1; index++)
2      {
3          //selezione o ricerca del minimo
4          min = index;
5          for (int i = index + 1; i < length; i++)
6              if (array[i] < array[min])
7                  min = i;
8
9          //scambia minimo ed elemento di indice index
10         swap(array, min, index);
11     }
```

Esempio di output (le parentesi quadre indicano le coppie di elementi che si scambieranno la posizione).

[8] 4 6 [1] 2 7 5 3

1 [4] 6 8 [2] 7 5 3

1 2 [6] 8 4 7 5 [3]

1 2 3 [8] [4] 7 5 6

1 2 3 4 [8] 7 [5] 6

1 2 3 4 5 [7] 8 [6]

1 2 3 4 5 6 [8] [7]

Esempi svolti

`selection_sort.c` – SelectionSort

Si basa sull'inserimento di ogni elemento in un **sottoarray ordinato**. Descrizione (informale) dello algoritmo:

- si consideri il **primo elemento** dello array. Esso rappresenta un sottoarray di lunghezza 1;
- si **inserisca** il **secondo elemento** al posto giusto nel sottoarray ordinato di lunghezza 1:
  - se questo è minore del primo e unico elemento, quest'ultimo si sposterà a destra;
- si **inserisca** al posto giusto il **terzo elemento**, spostando li elementi del sottoarray, se necessario, per mantenere l'ordinamento;
- ...

# InsertionSort

```
1  for (int index = 1; index < length; index++){
2      int key = array[index];
3      int position = index;
4
5      // shift valori piu' grandi di key a destra
6      while (position > 0 && array[position - 1] > key){
7          //shift a destra
8          array[position] = array[position - 1];
9          position--;
10     }
11     array[position] = key; //inserimento
12 }
```

Esempio di output.

[8] 4 6 1 2 7 5 3

[4 8] 6 1 2 7 5 3

[4 6 8] 1 2 7 5 3

[1 4 6 8] 2 7 5 3

[1 2 4 6 8] 7 5 3

[1 2 4 6 7 8] 5 3

[1 2 4 5 6 7 8] 3

Esempi svolti

`insertion_sort.c` – InsertionSort

**L'efficienza di un algoritmo di sorting** può essere stabilita **analiticamente** contando il numero di confronti, **in funzione della dimensione  $n$  dello input**.

Gli algoritmi di ordinamento {Bubble, Selection, Insertion} Sort operano in modo simile:

- ciclo esterno con numero di iterazioni approssimativamente uguale alla lunghezza dello array;
- ciclo interno che scandisce approssimativamente tutti gli elementi dello array;
- di conseguenza, **circa  $n^2$  confronti**

FINE