Contents

Insertion Sort

L'insertion sort itera gli elementi e gli sposta nella posizione giusta.

```
void insertion(int[] a, int i) {
    int insert = a[i];
    int moveItem = i;
    // I.C.: (a[0..moveItem-1],a[moveItem+1..i] ordinato) && (insert <
     ↪  a[moveItem+1..i])
    while (moveItem > 0 && a[moveItem - 1] > insert) {
        a[moveItem] = a[moveItem-1];
        --moveItem;
    }
    a[moveItem] = insert;
}

void insertionSort(int[] a, int n) {
    for (int i = 1; i <= n-1; i++) {
        // Inserisce a[i] al posto giusto in a[0...i-1],
        // spostando in avanti gli elementi in a[0...i-1] che sono > di a[i],
         ↪  per fare posto
        insertion(a,i);
    }
}
```

Selection Sort

Scambia il massimo con l'ultimo indice disponibile al momento

```
void selectionSort(int[] a, int n) {
    // I.C.: (a[0..n-pass] <= a[n-pass+1..n-1]) && (a[n-pass...n-1] ordinato)
```

```
    for (int pass = 1; pass <= n-1; pass++) {
        int iMax = findLastMax(a, n-pass+1);    // Trova il massimo
        swap(a, iMax], n-pass); // Mette il massimo ad a[n-pass]
    }
}
```

Bubble Sort

In questo caso il massimo per iterazioni successive va avanti all'ultima posizione.

```
void bubbleMax(int[] a, int m) {
    // I.C.: ???
    for (int i = 0, i <= m - 2; ++i) {
        if (a[i] > a[i+1]) {    // Swappo se a[i] > a[i+1]
            swap(a,i,i+1);
        }
    }
}
void mysterySort(int[] a, int n) {
    // I.C.: (a[0..n-pass] <= a[n-pass+1..n-1]) && (a[n-pass+1...n-1]
     ↪  ordinato)
    for (int pass = 1; pass <= n-1; pass++) {
    // sposta in a[n-pass] il massimo di a[0..n-pass]
        bubbleMax(a, n-pass+1);
    }
}
```

Stabilità di un algoritmo di ordinamento

Un metodo di ordinamento si dice stabile se preserva l'ordine relativo degli elementi con chiavi uguali all'interno della sequenza da ordinare.

Quindi due elementi con lo stesso valore, devono preservare l'ordine relativo.