

## Ordinamento

Un esempio di algoritmi:  
ordinamento

## Dati primitivi: ordinamento

- ◆ Un insieme di dati di tipo primitivo  $\{a_0, a_1, a_2, a_3, \dots, a_n\}$  si dice ordinato in ordine crescente se si ha
  - ◆  $a_0 \leq a_1 \leq a_2 \leq a_3 \leq \dots \leq a_n$
- ◆ I dati sono generalmente memorizzati in vettori.

## Metodo di selezione diretta (ordinamento crescente)

- ◆ L'algoritmo ricerca l'elemento minore della regione di coda del vettore da ordinare e lo sposta all'inizio della regione stessa.
- ◆ Ad ogni scansione della regione di coda viene spostato un elemento del vettore nella posizione corretta.
- ◆ L'ordinamento ha termine quando la regione di coda considerata è costituita da un elemento.

## Esempio: metodo di selezione diretta

1	23	4	-56	65	21	32	15	0	-3
-56	23	4	1	65	21	32	15	0	-3
-56	-3	4	1	65	21	32	15	0	23
-56	-3	0	1	65	21	32	15	4	23
-56	-3	0	1	4	21	32	15	65	23
-56	-3	0	1	4	15	32	21	65	23
-56	-3	0	1	4	15	21	32	65	23
-56	-3	0	1	4	15	21	23	32	65

```

/**
 Questa classe ordina un array e ne restituisce l'elenco dei
 valori sotto forma di stringa.
 */
public class SelezioneDiretta {
 private int[] a;
 /**
  Costruisce un ordinatore per selezione.
  @param anArray array da ordinare.
  */
 public SelezioneDiretta(int[] anArray) {
  a = anArray;
 }

 /**
  Ordina l'array per selezione diretta.
  */
 public void selectionSort() {
  for (int i = 0; i < a.length - 1; i++){
   int minPos = i;
   for (int j = i + 1; j<a.length; j++)
    if (a[j] < a[minPos])
     minPos = j;
   swap(minPos, i);
  }
 }
}

```

```

/**
 Scambia due elementi dell'array.
 @param i prima posizione da scambiare
 @param j seconda posizione da scambiare
 */
private void swap(int i, int j) {
 int temp = a[i];
 a[i] = a[j];
 a[j] = temp;
}

/**
 Genera una stringa contenente gli elementi del
 vettore.
 */
public String toString() {
 String s = "";
 for(int i=0; i<a.length; i++)
  s += a[i]+" ";
 s += "\n";
 return s;
}

```

```

public static void main(String[] args) {
 int dim = 10;
 if(args.length == 1)
  dim = Integer.parseInt(args[0]);
 int[] vet = new int[dim];
 for(int i=0; i<vet.length; i++)
  vet[i] = (int)(Math.random() * 100);
 SelezioneDiretta sorter = new SelezioneDiretta(vet);
 System.out.println(sorter);
 sorter.selectionSort();
 System.out.println(sorter);
}
}

```

## Metodo di ordinamento a bolle (ordinamento crescente)

- ◆ L'algoritmo scandisce dal fondo la regione di coda del vettore e confronta due dati consecutivi.
- ◆ Scambia due elementi se non sono ordinati.
- ◆ Ad ogni scansione l'elemento più piccolo si trova all'inizio della regione stessa.
- ◆ L'ordinamento ha termine quando la regione di coda considerata è costituita da un elemento.

## Esempio: metodo di ordinamento a bolle (bubble sort)

1	23	4	-56	65	21	32	15	0	-3
-56	1	23	4	-3	65	21	32	15	0
-56	-3	1	23	4	0	65	21	32	15
-56	-3	0	1	23	4	15	65	21	32
-56	-3	0	1	4	23	15	21	65	32
-56	-3	0	1	4	15	23	21	32	65
-56	-3	0	1	4	15	21	23	32	65
-56	-3	0	1	4	15	21	23	32	65
-56	-3	0	1	4	15	21	23	32	65
-56	-3	0	1	4	15	21	23	32	65

```
/**
 * Metodo di ordinamento alternativo: ordinamento a bolle.
 * Ad ogni iterazione l'elemento più piccolo della regione di coda
 * Viene spostato all'inizio della regione stessa.
 */
public void bubbleSort()
{
    for(int i=0; i<a.length-1; i++)
        for(int j=a.length-1; j>i; j--)
            if(a[j-1] > a[j])
                swap(j-1, j);
}
```

## Metodo di ordinamento a bolle (ordinamento crescente) II versione

- ◆ L'algoritmo scandisce dal fondo la regione di coda del vettore e confronta due dati consecutivi.
- ◆ Scambia i due elementi se non sono ordinati.
- ◆ Alla fine di ogni scansione l'elemento più piccolo della regione si trova all'inizio della regione stessa.
- ◆ L'ordinamento ha termine quando durante l'ultima scansione non sono stati effettuati scambi.

## Esempio: metodo di ordinamento a bolle (seconda versione)

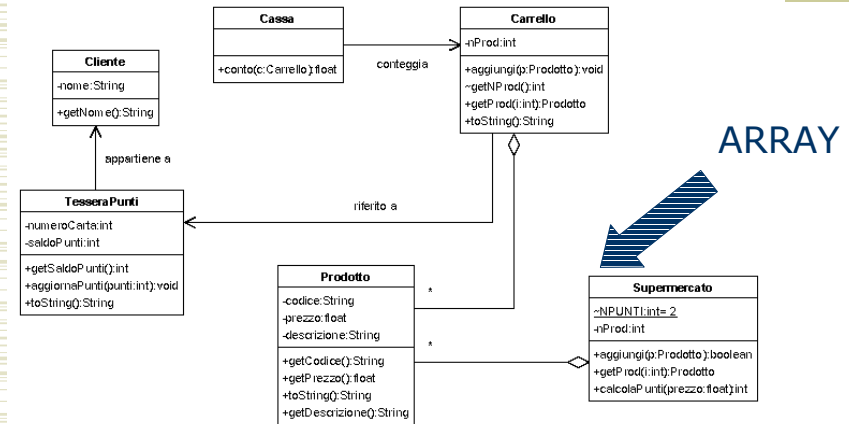
1	23	4	-56	65	21	32	15	0	-3
-56	1	23	4	-3	65	21	32	15	0
-56	-3	1	23	4	0	65	21	32	15
-56	-3	0	1	23	4	15	65	21	32
-56	-3	0	1	4	23	15	21	65	32
-56	-3	0	1	4	15	23	21	32	65
-56	-3	0	1	4	15	21	23	32	65
-56	-3	0	1	4	15	21	23	32	65

```

/**
Seconda versione del metodo di ordinamento a bolle.
Ad ogni iterazione l'elemento più piccolo della regione di coda
viene spostato all'inizio della regione stessa.
Quando il vettore risulta ordinato il ciclo esterno ha termine.
*/
public void bubbleSort() {
    boolean sorted = false;
    for(int i=0; i<a.length-1 && !sorted; i++){
        sorted = true;
        for(int j=a.length-1; j>i; j--){
            if(a[j-1] > a[j]){
                sorted = false;
                swap(j-1, j);
            }
        }
    }
}

```

## Il modello



## Classe Supermercato

```

public class Supermercato {
    // numero di punti per euro di spesa
    static final int NPUNTI = 2;
    private Prodotto lista[];
    private int nProd;

    public Supermercato(int n){
        lista = new Prodotto[n];
        nProd = 0;
    }

    public boolean aggiungi(Prodotto p){
        if(nProd < lista.length){
            lista[nProd++] = p;
            return true;
        } else
            return false;
    }

    public Prodotto getProd(int i){
        return lista[i];
    }
}

```

## Classe Prodotto

```

public class Prodotto {
    private String codice;
    private float prezzo;
    private String descrizione;

    public Prodotto(String laDescrizione, float ilPrezzo, String ilCodice) {
        codice = ilCodice;
        prezzo = ilPrezzo;
        descrizione = laDescrizione;
    }

    public String getCodice() {
        return codice;
    }

    public float getPrezzo() {
        return prezzo;
    }

    public String toString(){
        return "Codice: " + codice + "\tPrezzo: " + prezzo
            + "\tDescrizione: " + descrizione;
    }

    public String getDescrizione() {
        return descrizione;
    }
}

```

## Ordinamento di una collezione di oggetti.

1. Si deve definire quale sia l'informazione rispetto a cui effettuare l'ordinamento, detta *chiave di ordinamento*.
2. Si deve definire un metodo nella *classe degli oggetti* da ordinare che ne effettui il *confronto* sulla base della chiave scelta.
3. Vanno inseriti nella *classe che implementa la collezione* il metodo di *ordinamento* e il metodo di *scambio* tra elementi della collezione.

## Esempio: Supermercato

- ◆ *Chiave di ordinamento*: nome del prodotto (*codice*);
- ◆ Metodo di confronto, da inserire nella classe *Prodotto*:

```
public int confronta(Prodotto prod){  
    return this.codice.compareTo(prod.codice);  
}
```

Restituisce un valore negativo se la chiave dell'oggetto corrente precede la chiave dell'oggetto passato come parametro.

```
public int compareTo(String anotherString)  
    Compares two strings lexicographically.  
Parameters:  
    anotherString - the String to be compared.  
Returns:  
    the value 0 if the argument string is equal to this string;  
    a value less than 0 if this string is lexicographically less than the string  
    argument;  
    a value greater than 0 if this string is lexicographically greater than the  
    string argument.
```

## Esempio: Supermercato

- ◆ Metodo di ordinamento, da inserire nella classe *Supermercato*:

```
public void bubbleSort() {  
    for(int i = 0; i < count-1; i++)  
        for(int j = count-1; j > i; j--)  
            if(lista[j-1].confronta(lista[j]) > 0)  
                swap(j-1, j);  
}
```

## Esempio: la collezione di Supermercato

- ◆ Metodo di scambio, da inserire nella classe *Supermercato*:

```
private void swap(int i, int j){  
    Prodotto temp = lista[i];  
    lista[i] = lista[j];  
    lista[j] = temp;  
}
```

## Chiave di ordinamento numerica: prezzo

oppure

```
public int confronta(Prodotto prod){
    if (this.prezzo < prod.prezzo)
        return -1;
    else if (this.prezzo == prod.prezzo)
        return 0;
    else return 1;
}
```

## Chiavi di ordinamento: prezzo e codice

```
public int confronta(Prodotto prod){
    if (this.prezzo < prod.prezzo)
        return -1;
    else if (this.prezzo == prod.prezzo)
        return
        (this.codice.compareTo(prod.codice));
    else
        return 1;
}
```