7 Algoritmi di ricerca e di ordinamento

- Strategie di ordinamento di un Array
  1. * Scambio o Ingenuo; Bubble-Sort
     (Sono sicuramente le tecniche più semplice da implementare ma risultano le più scadenti in termini di efficenza)
  2. * Selection-Sort
     (Di semplice implementazione con efficienza ancora bassa se usato per notevoli quantità di dati)
  3. Insertion-Sort
     (Più efficiente delle precedenti ma ancora di bassa qualità se usato per notevoli dati)
  4. * Merge-Sort; Quick-Sort (Sono sicuramente le implementazioni più efficenti e ottimizzate)

(Nota: Vedi Simulazione Java)

7.1 Ordinamento per Scambio o Ingenuo

- L'idea di fondo è di confrontare sistematicamente ogni elemento di un Array con tutti gli elementi successivi per collocare nella posizione considerata l'elemento voluto (il più grande/piccolo).
  - Si può ottenere anche lo stesso risultato confrontando ogni elemento di un Array con tutti gli elementi che lo precedono (partendo dall'ultimo)

7.1 Ordinamento per Scambio o Ingenuo

- Il cuore dell'algoritmo consiste in un doppio ciclo annidato che all'occorrenza richiama una funzione per l'eventuale scambio di elementi.
- Nel caso si voglia ordinare un Array Vettore[n] in ordine crescente si avrà:

```
for (i=0; i<n-1; i++)
    for(k=i+1; k<n; k++)
        if (Vettore[i] > Vettore[k])
          Scambio(Vettore[i], Vettore[k]);
```

7.1 Ordinamento per Scambio o Ingenuo

- Come già detto: Si può ottenere anche lo stesso risultato confrontando ogni elemento di un Array con tutti gli elementi che lo precedono (partendo dall'ultimo)

```
for (i=n-1; i>0; i--)
    for(k=i-1; k>=0; k--)
        if (Vettore[i] < Vettore[k])
            Scambio(Vettore[i], Vettore[k]);
```

7.2 Ordinamento Bubble-Sort

- L'idea di fondo è di confrontare sistematicamente ogni elemento con quello successivo e troncare i confronti quando l'elemento più grande (o piccolo) è "emerso" verso la porzione prossima alla porzione di Array già ordinato.
  - Si può ottenere anche lo stesso risultato effettuando un confrontando sistematico ogni elemento con quello che lo precede fino a quando il più piccolo è "sceso" verso la porzione di Array già ordinato (partendo dall'ultimo)

7.2 Ordinamento Bubble-Sort

- Il cuore dell'algoritmo consiste in un doppio ciclo annidato che all'occorrenza richiama una funzione per l'eventuale scambio di elementi.
- Nel caso si voglia ordinare un Array Vettore[n] in ordine crescente si avrà:

```
for (i=n-1; i>0; i--)
    for(k=0; k<i; k++)
        if (Vettore[k] > Vettore[k+1])
            Scambio(Vettore[K], Vettore[k+1]);
```

Nota: da Destra

7.2 Ordinamento Bubble-Sort

- Come già detto: Si può ottenere anche lo stesso risultato effettuando un confrontando sistematico ogni elemento con quello che lo precede fino a quando il più piccolo è "sceso" verso la porzione di Array già ordinato (partendo dall'ultimo)

```
for (i=0; i<n-1; i++)
    for(k=n-1; k>i; k--)
        if (Vettore[k-1] > Vettore[k])
            Scambio(Vettore[k], Vettore[k-1]);
```

Nota: da Sinistra

7.3 Ordinamento Selection-Sort

- L'idea di fondo è di analizzare le porzioni di array posti a sinistra della parte già ordinata, ricercare fra questi l'indice della cella che contiene l'elemento più grande (piccolo) cercato e solo al termine dell'analisi si scambiano gli elementi posizionando quello trovato vicino alla porzione di array già ordinato.

```
for (i=n-1; i>0; i--)
    {
    M=0;
    for(k=1; k<=i; k++)
        if (Vettore[k] > Vettore[M])
           M=k;
    Scambio(Vettore[M], Vettore[i]);
    }
```

7.4 Ordinamento Merge-Sort e Quick-Sort

- Vedi Simulazione per apprezzarne l'efficenza