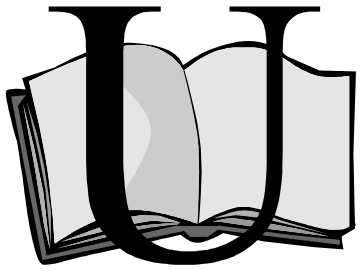
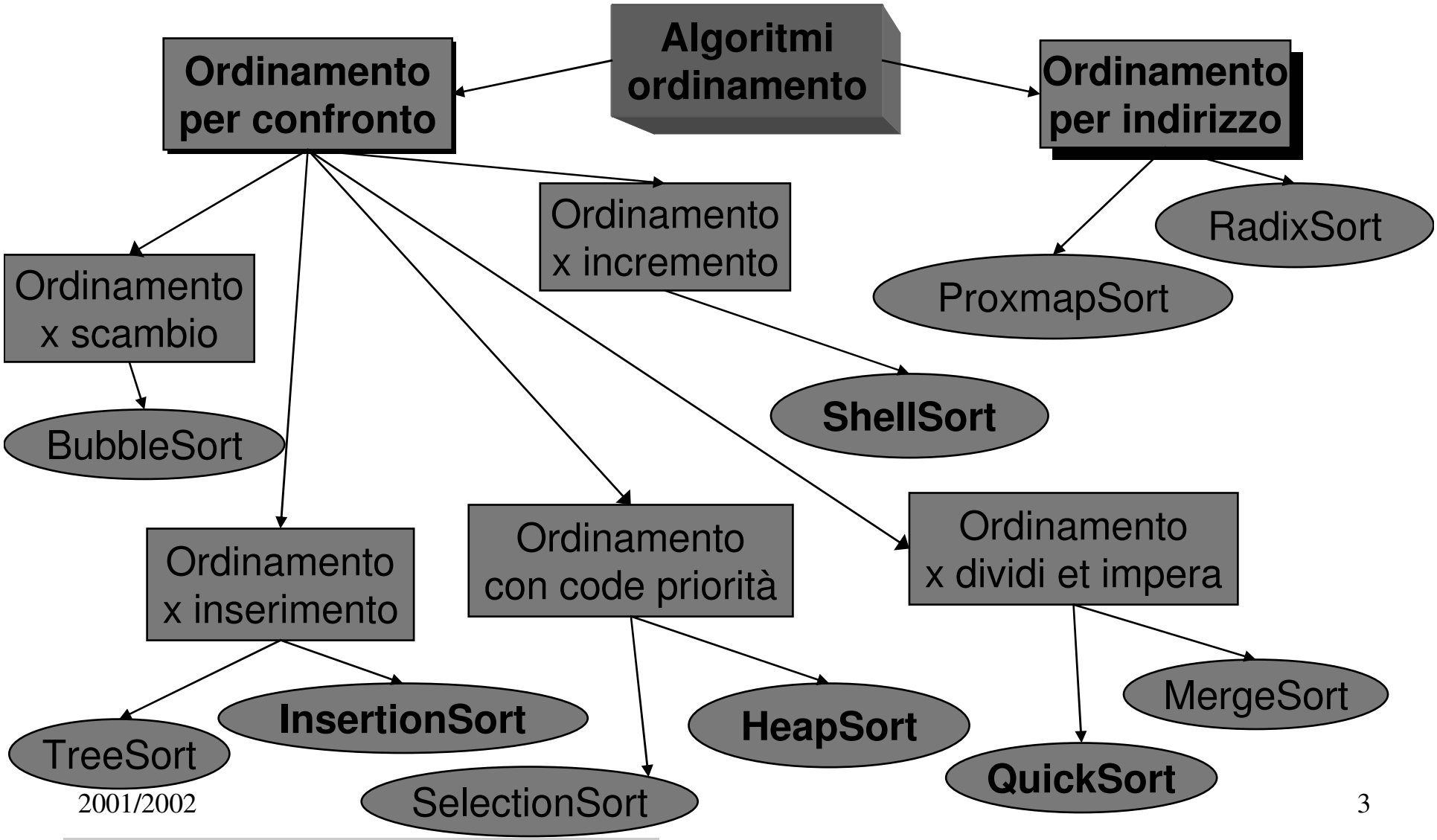
Algoritmi di ordinamento (I parte)



E2: sommario

- Classificazione degli algoritmi di ordinamento
- Studio di due implementazioni di algoritmi che utilizzano interfaccia Comparable
  - Algoritmo per inserimento (Insertion Sort)
    - rappresentante classe di algoritmi di ordinamento semplici
  - Algoritmo a passo calante (Shell Sort)
    - rappresentante classe di algoritmi di ordinamento efficienti

E2: Classificazione



E2: Insertion Sort (richiamo)

- algoritmo di ordinamento semplice:

```
insertionSort(data[])
   for (i=1; i< data.lenght; i++)
      tmp = data[i];
      sposta di 1 posizione a dx tutti elementi
                        j<i && data[j] > tmp;
      metti tmp nella posizione corretta
      (quella lasciata libera = j)
```

E2: Insertion Sort (richiamo)

- Esempio
  - input A =

0	1	2	3	4	5	6	7
M	**H**	**D**	**K**	**B**	**F**	**J**	**L**

  - dopo 1 ciclo =

0	1	2	3	4	5	6	7
H	**M**	**D**	**K**	**B**	**F**	**J**	**L**

  - dopo 2 ciclo =

0	1	2	3	4	5	6	7
D	**H**	**M**	**K**	**B**	**F**	**J**	**L**

  - dopo 3 ciclo =

0	1	2	3	4	5	6	7
D	**H**	**K**	**M**	**B**	**F**	**J**	**L**

  - ...
  - dopo 7 ciclo =

0	1	2	3	4	5	6	7
B	**D**	**F**	**H**	**J**	**K**	**L**	**M**

E2: Insertion Sort (richiamo)

- Vantaggi:
  - ordina array solo se necessario (spostamento)
- Svantaggi:
  - inserimento non è localizzato
- Complessità (# confronti+#spostamenti)
  - ciclo esterno eseguito n-1 volte
  - caso migliore: n-1 confronti + 2(n-1) spostamenti--> O(n)
  - caso peggiore: $O(n^2)$ confronti + $O(n^2)$ spostam. --> $O(n^2)$
  - caso medio: $O(n^2)$ confronti + $O(n^2)$ spostam. --> $O(n^2)$

E2: ShellSort (richiamo)

- Idea algoritmo
  1. dato un array A[0..n],
  2. si fissa una sequenza decrescente di valori $<i_1,\ldots,i_k>$ t.c. $i_1<n/2$ e $i_k=1$
  3. Per ogni Delta i_a si partizione l'array in sottoseq.
     - $a_0, a_{0+ia}, a_{0+2ia},\ldots$
     - $a_1, a_{1+ia}, a_{1+2ia},\ldots$
     - …
     - $a_{ia-1}, a_{ia-1+ia}, a_{ia-1+2ia},\ldots$

  e si ordinano con il metodo InsertSort()
  4. si restituisce l'array A (ordinamento crescente)

E2: ShellSort (richiamo)

- Esempio
  - input A =

0	1	2	3	4	5	6	7
M	**H**	**D**	**K**	**B**	**F**	**J**	**L**

  - fisso sequenza Delta = 3,2,1
  - Delta = 3
    - Sottosequenze:

0	1	2	3	4	5	6	7
M	**H**	**D**	**K**	**B**	**F**	**J**	**L**
M	-	-	K	-	-	J	
	H	-	-	B	-	-	L
		D	-	-	F		

    - ordino sottosequenze

0	1	2	3	4	5	6	7
J	**B**	**D**	**K**	**H**	**F**	**M**	**L**
J	-	-	K	-	-	M	
	B	-	-	H	-	-	L
		D	-	-	F		

E2: ShellSort (richiamo)

- Esempio, continua
  - Delta = 2
    - Sottosequenze:

0	1	2	3	4	5	6	7
J	**B**	**D**	**K**	**H**	**F**	**M**	**L**
J	-	D	-	H	-	M	
	B	-	K	-	F	-	L

    - ordino sottosequenze

0	1	2	3	4	5	6	7
D	**B**	**H**	**F**	**J**	**K**	**M**	**L**
D	-	H	-	J	-	M	
	B	-	F	-	K	-	L

E2: ShellSort (richiamo)

- Esempio, continua
  - Delta = 1
    - La sottosequenza è l'array stesso.
    - L'ordinamento è l'InsertSort sull'array

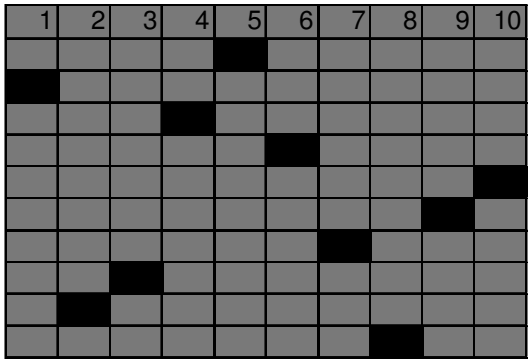
0	1	2	3	4	5	6	7
D	**B**	**H**	**F**	**J**	**K**	**M**	**L**
B	D	F	H	J	K	L	M

0	1	2	3	4	5	6	7
B	**D**	**F**	**H**	**J**	**K**	**L**	**M**

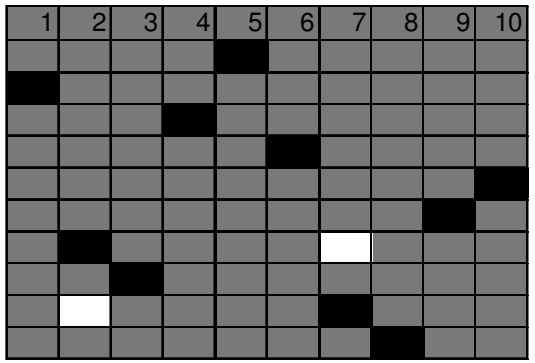
E2: ShellSort: punto di vista altern.

- Dato A = { i | 1<=i<=10}

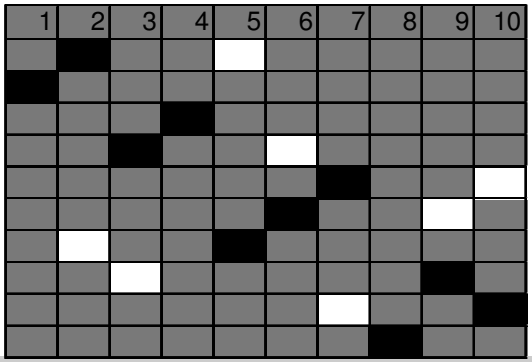
Array *A*



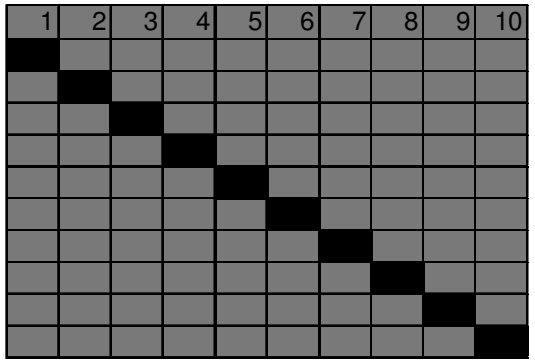
Delta = 5



Delta = 3



Delta = 1



E2: ShellSort: parametro Delta

- Diversi modi per scegliere sequenza Delta:
  - anche in soli due passi (Delta: $(16n/\pi)^{1/3}$ e 1) complessità ShellSort = $O(n^{5/3})$ (Knuth)
  - Sequenza Delta Non deve essere pari o dispari
- Euristica:
  - $delta_1 = 1$;
  - $delta_{i+1} = 3delta_i + 1$
  - si arresta a $delta_t$ t.c. $delta_{t+1} >= n$

E10: ShellSort

- Costo computazionale
  - risultato analitico difficile
  - costo computazionale InsertSort = $O(n^2)$
  - risultati sperimentali ShellSort = $O(n^{1,25})$

E2: implem.: confronto fra oggetti

- Si vuole dare un'implementazione degli algoritmi che ordini oggetti generici (Object)
- Necessario quindi confrontare oggetti
- Che tipo di confronto si deve utilizzare tra due oggetti x e y?
  - x == y è vero sse i due oggetti sono il medesimo
    - troppo forte… poco utile…

E2: implem.: confronto fra oggetti

  - sarebbe meglio avere x ‘uguale’ y sse rappresentano lo stesso oggetto, quindi...
  - Il confronto deve essere basato sul valore e non sul riferimento
- La classe Object implementa metodo equals()
  - realizza l’operatore == tra reference…
  - è il candidato ideale per l’overriding (si può ridefinire cosa significa ==)
  - ma non è sufficiente per dire se x > o < y

E2: implem.: confronto fra oggetti

- è necessario quindi che gli oggetti mettano a disposizione un metodo per confrontarli
- solo attraverso questo metodo è possibile eseguire un ordinamento

- interfaccia java.lang.Comparable
  - semplice interfaccia per richiedere un metodo di confronto (compareTo())
  - gli algoritmi richiedendo come parametro un array di Comparable si garantiscono di poter confrontare gli oggetti in modo corretto.

E2: implem.: Interfaccia Comparable

```
public interface Comparable {
  /**
   * Restituisce un intero negativo o 0 o un intero positivo a seconda che
   * l'oggetto sia minore o uguale o maggiore dell'argomento (elemento).
   */
   public int compareTo(Object elemento);
}
/** Contratto
 * Impone un ordine totale sugli oggetti della classe che la implementa.
 * Questo ordinamento è detto anche ordinamento naturale della classe
 *
 * Un ordinamento è detto consistente con l'uguaglianza sse
 * (e1.compareTo((Object)e2)==0) ha valori identico a
 *  e1.equals((Object)e2) per qualsiasi e1 e e2 della classe C.
 *
 * È raccomandato implementare il metodo in modo consistente
 */
```

E2: implem.: Interfaccia Comparable

- I metodi delle classi insertionSort e shellSort devono utilizzare compareTo()
- Si deve imporre all'utente che gli oggetti dell'array in input a classi che implementano Comparable.

E2: implem.: Interfaccia Comparable

- Classi che implementano Comparable()
  - Class — Natural Ordering
  - Byte — signed numerical
  - Character — unsigned numerical
  - Long — signed numerical
  - Integer — signed numerical
  - Short — signed numerical
  - Double — signed numerical
  - Float — signed numerical
  - BigInteger — signed numerical
  - BigDecimal — signed numerical
  - File — system-dep. lexicographic on pathname.
  - String — lexicographic
  - Date — chronological

E2: InsertionSort: procedura

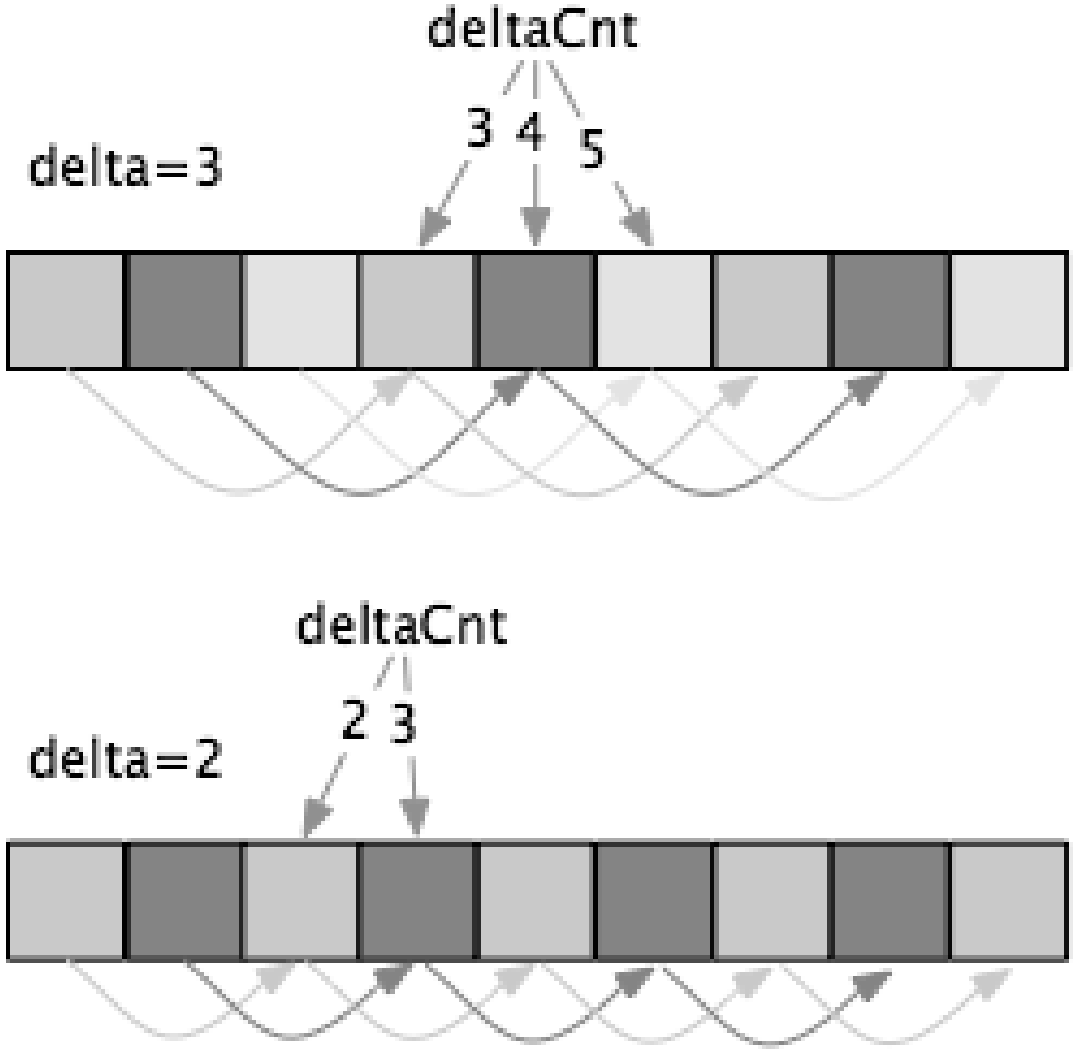
```
/** InsertionSort(data) ordina il vettore data di oggetti Comparable
*/
public static void insertionSort(Comparable data[]) {
    Comparable tmp;
    int i, j;
    for (i =1; i < data.lenght; i++) {
       tmp = data[i];
       for (j = i; j>0 && tmp.compareTo(data[j-1])<0; j--)
          data[j] = data[j-1];
       data[j] = tmp;
    }
}
```

E2: ShellSort: procedura

```
/** ShellSort(data) ordina il vettore data di oggetti Comparable
*/
public static void shellSort(Comparable a[]) {
   int i, j, k, delta, deltaCnt, increments[] = new int[20];
   Comparable tmp;
   // crea il numero corretto di incrementi delta
   for ( delta=1, i=0; delta < data.length && i < increments.length; i++) {
      increments[i] = delta;
      delta = 3* delta +1;
   }
   // itera per il numero di diversi delta
   for ( i--; i>=0; i--) {
      delta = increments[i];
```

E2: ShellSort: procedura

```
// itera per il numero di sottoarray delta-ordinati
for ( deltaCnt=delta; deltaCnt<2*delta; deltaCnt++)  {
   // ordinamento per inserimento nel sottoarray indicizzato con passo
   //delta
   for ( j= deltaCnt; j < data.length; ) {
      tmp = data[j];
      k = j;
      while ((k-delta)>=0 && tmp.compareTo(data[k-delta]) < 0) {
         data[k] = data[k-delta];
         k -= delta;
      }
   data[k] = tmp;
   j += delta;
} } }
}
```


deltaCnt
3 4 5
delta=3
deltaCnt
2 3
delta=2

E2: Insertion-Shell: esempio d'uso

…classe Test

```
...
    Integer[] a = new Integer[n];
    System.out.print("Array a: ");
    for (i=0; i<n; i++) {
            a[i] = new Integer(rnd.nextInt(n));
            System.out.print(a[i] + ",  ");
   }
   Sort.shellSort(a); //oppure Sort.insertionSort(a)
   System.out.print("\nArray ordinato: ");
...
```