# Laboratorio di Algoritmi e Strutture dati

15 marzo 2015

## Esercizio 1 – Ricerca binaria *(5 pt)*

### IntSortedArray (array ordinato di interi parzialmente riempito)

In un file *IntSortedArray.java* si definisca una classe *IntSortedArray* contenente:

1. un campo privato `elements` di tipo `int[]`, che conterrà gli elementi e sarà in generale solo parzialmente riempito;
2. un campo privato `size` di tipo `int` che contiene il numero di elementi effettivamente presenti nell'array (non la capacità!);
3. un costruttore pubblico di default `IntSortedArray()` che crea un array di capacità iniziale 16;
4. un costruttore pubblico `IntSortedArray(int initialCapacity)` che crea un array della data capacità iniziale; `initialCapacity` può essere un intero positivo o anche 0; se è negativo viene lanciata una eccezione `IllegalArgumentException`;
5. un costruttore pubblico `IntSortedArray(int[] a)` che prende come argomento un array `a` di `int`, non necessariamente ordinato, e costruisce un *IntSortedArray* avente come capacità la lunghezza di `a` incrementata di 16, e contenente tutti gli elementi di `a`;
6. un metodo pubblico `int size()`, che restituisce il numero di elementi effettivamente presenti nell'array (non la capacità!);
7. un metodo privato `int binarySearch(int x)` che realizza la procedura di ricerca binaria iterativa con tutti i raffinamenti visti a lezione, compresa la **restituzione della posizione di inserimento se l'elemento non è presente**;
8. un metodo pubblico `int insert(int x)` che inserisce `x` in `elements` mantenendolo ordinato, e restituisce l'indice a cui è stato inserito: lo inserisce in ogni caso, anche se l'elemento è già presente (in tal caso si avrà evidentemente un elemento ripetuto); se l'array è pieno, prima rialloca gli elementi in un array di dimensione doppia e poi inserisce l'elemento;

9. un metodo pubblico `int get(int i)` che restituisce l'i-esimo elemento, oppure solleva l'eccezione `ArrayIndexOutOfBoundsException` se l'i-esimo elemento non esiste;

10. un metodo pubblico `toString()` override dell'omonimo metodo di Object, che produca una stringa rappresentante l'array ordinato, con gli elementi racchiusi fra parentesi quadre e separati da virgola e spazio; esempio: `[5, 13, 25, 25, 43, 61]`.

Si definisca e si esegua poi, tramite *JUnit*, lo unit testing dei costruttori e dei metodi *size*, *insert*, *toString*, *indexOf*, *get*.

## SortedArrayList (ArrayList generica ordinata)

Questo esercizio è una versione generica dell'esercizio precedente, realizzata per semplicità mediante la classe *ArrayList* di Java. In un file *SortedArrayList.java* si realizzi una classe *SortedArrayList* contenente:

1. un campo privato `elements` di tipo `ArrayList<E>`;

2. un costruttore pubblico di default `SortedArrayList<E>()` che crea una ArrayList<E> di capacità iniziale 16;

3. un costruttore pubblico `SortedArrayList<E>(int initialCapacity)` che crea una ArrayList<E> della data capacità iniziale;

4. un costruttore pubblico `SortedArrayList<E>(E[] a)` che prende come argomento un array `a` di `E`, non necessariamente ordinato, e costruisce una ArrayList<E> avente come capacità la lunghezza di `a` incrementata di 16, e contenente tutti gli elementi di `a`;

5. un metodo pubblico `int size()`, che restituisce il numero di elementi effettivamente presenti nell'ArrayList (banalmente richiama il metodo *size* di ArrayList!);

6. un metodo privato `int binarySearch(E x)` che realizza la procedura di ricerca binaria iterativa con i vari raffinamenti visti a lezione compresa la restituzione della posizione di inserimento;

7. un metodo pubblico `int insert(E x)` che inserisce `x` in `elements` mantenendolo ordinato e restituisce l'indice a cui è stato inserito;

8. un metodo pubblico `T get(int i)` che restituisce l'i-esimo elemento (banalmente richiama la *get* di *ArrayList*!)

9. un metodo pubblico `toString()` override dell'omonimo metodo di Object (banalmente richiama il *toString* di *ArrayList*!)