Algoritmi e Strutture Dati
&
Laboratorio di Algoritmi e Programmazione

— Appello del 19 Giugno 2007 —

Esercizio 1 (ASD)

Si risolvano le seguenti ricorrenze, giustificando la risposta.

- $T(n) = 3T(\frac{n}{4}) + 3n$
- $T(n) = 2T(\frac{n}{4}) + T(\frac{n}{3}) + n$

Esercizio 2 (ASD)

Si disegni un albero R/B che contiene le chiavi: $\{8, 28, 3, 10, 12, 20, 30, 40, 1, 4, 6, 9, 11, 15, 45, 50\}$.

Esercizio 3 (ASD)

Si sviluppi un algoritmo per calcolare il grado massimo dei nodi di un albero generale rappresentato tramite gli attributi: `fratello`, `figlio` e `padre`. (Ricordiamo che il grado di un nodo di un albero è pari al numero dei suoi figli.)

Esercizio 4 (ASD)

Si sviluppi un algoritmo che dato un intero x ed una lista di interi L costruisce una nuova lista L' che contiene tutti gli elementi di L che sono multipli di x. Si definisca la struttura dati utilizzata per rappresentare le liste, si dimostri la correttezza dell'algoritmo sviluppato e se ne valuti la complessità.

[continua sul retro]

Esercizio 5 (LAB)

Sia data la seguente interfaccia per alberi binari con struttura a nodi: ogni nodo hai una chiave e puntatori ai figli destro e sinistro (*senza puntatore al padre*).

```
interface BinTree  {
      Node root();               // la radice dell'albero
      Node left(Node p);         // figlio sinistro di p nell'albero
      Node right(Node p);        // figlio destro di p nell'albero
      int key(Node p);           // la chiave contenuta nel nodo p
}
```

Per tutti i metodi con parametro di tipo `Node` assumiamo la la precondizione `p != null`, mentre tutti i metodi con risultato di tipo `Node`, restituiscono `null` quando il loro valore è indefinito (ad esempio, `root()` su un albero vuoto). Definite, in Java, l'implementazione per il metodo `succ()` descritto dalla seguente specifica.

```
public static Node succ(BinTree T, int k) throws NoSuchElementException
/**
 * PRE:  T != null e' un albero binario di ricerca con chiavi distinte, k e' una chiave contenuta in T.
 *
 * POST: restituisce un riferimento al nodo che contiene la minima tra le chiavi
 * maggiori (il successore) di k in T. Se tale nodo non esiste lancia l'eccezione.
 */
```

L'implementazione deve garantire una complessità $O(h)$, dove h è l'altezza dell'albero.

Esercizio 6 (LAB)

Fornite l'implementazione del metodo `checkSum()` specificato qui di seguito.

```
 public static bool checkSum(int[] A, int[] B, int x)
/**
 * PRE:  A e B sono due array ordinati
 *
 * POST: restituisce true se esistono a in A e b in B tali che a+b = x
 */
```

L' implementazione deve garantire una complessità $O(n \log n)$ dove n è il numero complessivo degli elementi in `A` e `B`.