Prova scritta di Algoritmi e Strutture Dati (1^o anno)

Luglio 2000

NOTE:

- L'ordine in cui vengono svolti gli esercizi non è rilevante (in altre parole: se avete problemi a svolgerne uno potete passare tranquillamente ai successivi e farlo dopo).
- I punti previsti per ogni esercizio si riferiscono ad uno svolgimento completamente corretto.

NOME:

COGNOME:

MATRICOLA:

==

Non scrivere qui sotto

Esercizio	Punti previsti	Punti assegnati
1	3	
2	3	
3	11	
4	8	
5	5	
Totale	30	

Esercizio 1 (punti 3)

Consideriamo il seguente codice C:

```
#include <stdio.h>
#include <stdlib.h>

int n = 10;

void na(int **a);
int fff (int *x);
void ppp (int x[]);

main()
{
  int *a; int i;
  na(&a);
  for (i=0;i<n;i++) printf("%d ",a[i]);
  printf("\n");
  ppp(a);
  for (i=0;i<n;i++) printf("%d ",a[i]);
  printf("\n");
}

void na(int **a)
{
   int i, *b;
   b = (int *) calloc(n,sizeof(int));
   for (i=0;i<n;i++) b[i] = n-i;
   *a = b;
 }

int fff (int *x)
{ *x = *x + 1;
  return( *x );
}

void ppp (int x[])
{
  int i;
  for (i=1;i<n;i++)
     x[i]=fff(&x[i-1]);
}
```

Domanda: Dire qual'è l'output del programma

Esercizio 2 (punti 3)

Consideriamo il seguente codice C:

```
#include <stdio.h>
#include <math.h>

int a = 10;
int b = 20;

void ppp (int x, int *y);

void qqq (int x);

main()
{
  int a = 1;
  int b = 100;
  int c = -1;
  ppp(a, &b);
  qqq(c);
  printf("%d, %d, %d\n", a,b,c);
}

void ppp (int x, int *y)
{
  x = b * 2;
  *y = x - a;
}

void qqq (int x)
{ float r, s;
  int i;
  x = a + b;
  s = (float)sqrt(x * 2.5);
  if (s < a/5) r = s; else r = -s;
  for (i=0;i<b;i++)
     x = (int) x * r;
  x = (int)sqrt(x/s);
}
```

Domanda: Dire qual'è l'output del programma

Esercizio 3 (punti 11)

Consideriamo il tipo di dato *Dizionario* di numeri interi (senza duplicati) implementato con:

- alberi binari di ricerca (con puntatori);
- liste dinamiche (con puntatori).

Domande:

1. Definire in pseudo-codice i tipi utilizzati per l'implementazione del tipo di dato nei due casi;
2. Definire in pseudo-codice la primitiva **Tree-to-List** che prende in input un dizionario rappresentato con un albero e produce lo stesso dizionario implementato con lista **ordinata**; l'albero in input deve essere demolito dalla primitiva.
3. Definire in pseudo-codice la primitiva **List-to-Tree** che prende in input un dizionario rappresentato con lista **non ordinata** e produce lo stesso dizionaro implementato con albero binario; la lista in input deve essere demolita dalla primitiva.
4. Descrivere come risulterebbe l'albero restituito dalla primitiva al punto precedente (dal punto di vista del bilanciamento) se la lista in input fosse ordinata.

N.B.: se nell'implementazione delle primitive ai punti 2 e 3 si utilizzano primitive standard sui tipi i dato in oggetto (scelta consigliata) è necessario definire in pseudo-codice l'implementazione di **tutte** le primitive utilizzate.

Esercizio 4 (punti 8)

Consideriamo il seguente framento di pseudo-codice (che scrive la parte intera del logaritmo in base 2 di tutti i numeri minori o uguali a n):

```
per i = 1,...,n :
{
   j ← 1;
   k ← -1;
   while j ≤ i do
   {
      k ← k + 1;
      j ← 2 * j;
   }
   scrivi(i, k);
}
```

Domanda: determinare la complessità della procedura in $\Omega()$ e in $O()$ in funzione di n.

Aiuto: ricordare le seguenti proprietà dei logaritmi e del fattoriale (servono tutte)

- $\log_2 a + \log_2 b = \log_2(ab)$
- $\log_2(a^b) = b \log_2 a$
- $n! \in \Omega(2^n)$
- $n! \in O(n^n)$

Esercizio 5 (punti 5)

Consideriamo la seguente funzione ricorsiva che lavora su liste dinamiche di interi implementate con puntatori (campo next per il puntatore all'elemento successivo).

```
function conta(l : lista): integer;
{
  if (l = NULL) then
     return(0)
  else
     return(1 + conta(l->next));
}
```

Domande: Stimare la complessità della funzione in $\Theta()$ in funzione del numero di elementi n della lista:

1. utilizzando il metodo dell'albero di ricorsione;
2. utilizzando le equazioni alle ricorrenze.