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

Giugno '99

**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 | 2 | |
| 4 | 5 | |
| 5 | 7 | |
| 6 | 6 | |
| 7 | 4 | |
| Totale | 30 | |

**Esercizio 1** (punti 3)

Consideriamo il seguente codice C:

```
#include <stdio.h>
#define  MAX 12

char s[MAX];

main()
{  int i,m;
   char c;
   for ( i=0 ; i<MAX ; i++ ) scanf( "%c", &s[i] );
   m = MAX/2;
   for ( i=0 ; i<m ; i++ )
   {
      c = s[i];
      s[i] = s[MAX-i-1];
      s[MAX-i-1] = c;
   }
   for ( i=0 ; i<MAX ; i++ ) printf( "%c", s[i] );
   printf("\n");
}
```

**Domande:**

1. Dire qual'è l'output per il seguente input: ARUTTETIHCRA
2. Descrivere in generale cosa fa il programma
3. Se MAX fosse dispari (es: MAX=11) il programma funzionerebbe ancora correttamente?

**Esercizio 2** (punti 3)

Consideriamo il seguente codice C:

```
#include <stdio.h>

void ppp (int * x, int * y);
int  fff (int x, int y);

main()
{
  int a = 4;
  int b = 2;
  int c;
  c = fff(a,b);
  printf ("%d, %d, %d\n", a, b, c);
  ppp(&a, &b);
  printf ("%d, %d\n", a, b);
}

void ppp (int * x, int * y)
{ int i, aux;
  aux = 1;
  for (i=0; i<*y; i++)
    aux = *x * aux;
  *y = aux / 2;
}

int fff (int x, int y)
{ int aux;
  aux = x;
  x = y * 3;
  return( x + aux );
}
```

**Domanda:** qual'è l'output stampato dalle printf?

**Esercizio 3** (punti 2)

Consideriamo le espressioni definite induttivamente come segue:

Lett $= \{A, B, C, \ldots, Z\}$, l'insieme delle lettere maiuscole.

- (base) Lett $\subset$ Exp
- (passo 1) x, y $\in$ Exp $\Rightarrow$ x & y $\in$ Exp
- (passo 2) p, q $\in$ Exp $\Rightarrow$ ( p @ q ) $\in$ Exp

**Domande:**

1. Scrivere una stringa che appartiene ad Exp, di lunghezza $\geq 7$ e contenente parentesi
2. (A & B) appartiene ad Exp ?

**Esercizio 4** (punti 5)

Consideriamo il tipo di dato *Dizionario* di numeri interi implementato con alberi binari di ricerca (con puntatori).

**Domande:**

1. Definire in pseudo-codice i tipi utilizzati per l'implementazione del tipo di dato;
2. Definire in pseudo-codice la primitiva **Insert** tenendo presente che non si ammettono duplicati in un insieme;
3. Disegnare l'albero che si ottiene partendo dall'albero vuoto e inserendo tramite la Insert i valori della sequenza di input: 5 18 3 2 23 44 15 17 31 5 6
(segnare il valore contenuto in ogni nodo dell'albero).

**Esercizio 5** (punti 7)

Consideriamo il tipo di dato **Successione** di lunghezza $\leq K$ di numeri interi, implementato con array + lunghezza.

Consideriamo l'operazione

$$\text{Insert-after} : \text{Int} \times \text{Succ} \times \text{Int} \longrightarrow \text{Succ}$$

Definita (a parole) come segue:

Insert-after(x,s,y) cerca in s la prima occorrenza di y (a partire dall'inizio) e:

- se trova y allora inserisce x in s nel posto successivo a y (spostando in avanti tutti gli elementi seguenti);
- se non trova y allora inserisce x in coda;
- segnala errore (e non inserisce x) se s è già piena;
- la successione s viene modificata dall'inserimento (nel senso che non se ne fa una copia).

**Domande:**

1. Definire in pseudo-codice il tipo utilizzato per implementare le successioni.
2. Definire in pseudo-codice l'implementazione di Insert-after come procedura.
3. Calcolare la complessità dell'implementazione fatta per Insert-after in funzione della lunghezza n di s, in $\Theta()$.
4. Indicare se esiste un caso peggiore (se sì, spiegare quale) oppure se la complessità non dipende dal particolare input ma solo da n (in questo caso spiegare perché).

**Nel calcolo di complessità: non fare conti troppo dettagliati esplicitando tutte le costanti, ma non limitarsi a neppure a dare solo il risultato: bisogna spiegare come ci si arriva.**

**Esercizio 6** (punti 6)

Consideriamo la procedura seguente in pseudo-codice:

```
procedure ppp(aa : array[1..n] of integer IN; bb : array[1..n] of integer OUT);
{
 i, j : integer;
 per i = 1,...,n :
 {
    b[i] ← 0
    j ← n
    while j ≥ i do
    {
       b[i] ← b[i] + a[j]
       j−−
    }
 }
}
```

**Domanda:** determinare la complessità della procedura in funzione della dimensione n dei parametri.

**Non fare conti troppo dettagliati esplicitando tutte le costanti, ma considerare il costo di tutte le parti di programma mettendo eventualmente in evidenza le operazioni dominanti (una volta che si sono individuate si possono ignorare le altre). Non limitarsi a dare solo il risultato: bisogna spiegare come ci si arriva.**

**Esercizio 7** (punti 4)

Assumiamo che la complessità di una procedura ricorsiva sia definita dal seguente sistema alle ricorrenze:

$T(1) = b$
$T(n) = a + 2\ T(n/2)$

dove a e b sono due costanti e n è la dimensione del problema.

**Domanda:** dare la complessità della procedura in $\Theta()$ spiegando come si ottiene (suggerimento: usare l'albero della ricorsione).