

TESTO E SOLUZIONI

(nota: tra i punteggi dati agli esercizi, alcuni sono un po' gonfiati)

Esercizio 1 (1 punto)

Consideriamo il seguente codice C:

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

main() {
    int n;
    scanf("%d", &n);

    { char *y;
      int k;

      y = (char *) calloc( n, sizeof(char) );
      for (k = 0; k < n; k++) *(y+k) = k;
      for (k = n-1; k >= 0; --k) printf("%d, ", y[k]);
    }
}
```

Domanda: qual'è l'output se in input, per n, diamo 9 ?

Risposta: 8, 7, 6, 5, 4, 3, 2, 1, 0,

Esercizio 2 (3 punti)

Consideriamo il seguente codice C:

```
#include <stdio.h>

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

main () { int a = 2;
          ppp (&a, &a);
          printf ("%d", a);
        }

void ppp (int * x, int *y) { *y = - *y ;
                             *x = *x + *y ;
                          }
```

Domanda: qual'è l'output ?

Risposta: - 4

Spiegazione (NON richiesta allo scritto):

L'effetto della chiamata `ppp (&a, &a)` è:

-- passaggio parametri: assegnare ai parametri `x` ed `y` il valore "indirizzo di `a`"

-- istruzioni: `*y = -*y;` equivale a: `a = -a;`
`*x = *x + *y;` equivale a: `a = a + a;`

Esercizio 3 (5 punti: 1 + 1 + 3)

1) $\log_3 n \in \Omega(\log_2(n^3))$? **Risposta : sí**

2) $n^{2n} \in O(n^n)$? **Risposta : no**

3) **A** è un algoritmo con input n (n intero positivo).

La sua funzione complessità, in funzione di n , è

$T_1(n) = 100$ se n è dispari

$3n$ se n è pari

Consideriamo ora la complessità di **A** in funzione della lunghezza della rappresentazione binaria di n e indichiamo la nuova funzione complessità con **T_2**. Dare una stima per **T_2** in $\Theta(\dots)$.

Sul foglio risposte scrivere solo : $T_2(k) \in \Theta(\dots)$

Risposta: $T_2(k) \in \Theta(2^k)$

Spiegazione (NON richiesta allo scritto):

1) $\log_2(n^3) = 3 \log_2 n = 3 c \log_3 n$ dove $c = \log_2 3$

2) $n^{2n} = n^n n^n$ e non c'è nessuna costante c tale che $n^n n^n \leq c n^n$

3) indicando con k la dimensione dell'input (cioè la lunghezza ...), il legame tra

k e input m di dimensione k è: $k \approx \log_2 m$ cioè: $m \approx 2^k$;

per un dato k , ci sono input (di lunghezza k) sia pari che dispari;

la complessità in funzione di k è quella dell'input "peggiore", cioè pari,

e quindi: $T_2(k) \in \Theta(2^k)$.

(Vedere anche dispense, discussione complessità della funzione fattoriale)

Esercizio 4 (punti 3 in prima approssimazione)

Consideriamo espressioni composte da: lettere, %, #, parentesi tonde.

L'insieme Exp delle espressioni è dato dalla seguente definizione (induttiva), dove:

$Lett = \{a, b, c, \dots, z\}$, l'insieme delle lettere minuscole.

- (base) $Lett \subset Exp$
- (passo_1) se $x \in Lett$ e $exp \in Exp$ allora $x \% (exp) \in Exp$
- (passo_2) se $exp1, exp2 \in Exp$ allora $exp1 \# exp2 \in Exp$

Domanda_1: Scrivere una stringa che appartiene ad Exp , di lunghezza ≥ 8 , contenente sia il simbolo % che il simbolo #

Risposta: $a \% (a \# a \# a)$
oppure: $a \% (b \# c) \# c$ $a \# b \% (a \# b)$ eccetera

Domanda_2: Dare la successione di insiemi che definisce Exp .

In altre parole, vogliamo, sul foglio risposte:

$Exp_0 = \dots\dots\dots$ $Exp_{k+1} = \dots\dots\dots$
(con gli Exp_k tali che $Exp = \bigcup \{ Exp_k \mid k \geq 0 \}$)

Risposta: $Exp_0 = Lett$
 $Exp_{k+1} = Exp_k \cup \{ x \% (e) \mid x \in Lett, e \in Exp_k \}$
 $\cup \{ e1 \# e2 \mid e1 \in Exp_k, e2 \in Exp_k \}$

(non è l'unica successione di insiemi possibile; per una variante, vedere dispense)

Esercizio 5 (4 punti)

Consideriamo:

- il tipo di dato **successioni di caratteri di lunghezza $\leq K$** dove K è una costante fissata;
- con le 3 sorti: succ, char, bool ed i corrispondenti insiemi SUCC, CHAR, BOOL;
- l'operazione $chop : SUCC \rightarrow SUCC$ definita da
 $chop (<a_1, \dots, a_n >) = <a_1, \dots, a_{n-1}>$ se $n > 0$
 $chop (<>) = \text{indefinito}$ altrimenti
- l'implementazione delle successioni con "array di caratteri + lunghezza"

Sul foglio risposte:

- a) scrivere, in C, Pascal, pseudo-codice, il tipo corrispondente alla sorte succ
- b) precisare come si rappresentano la successione vuota e la successione <a, b>
- c) scrivere, in pseudo-codice, un'implementazione dell'operazione chop , con le seguenti caratteristiche:
 - non si produce una nuova successione, ma si modifica l'argomento
 - si gestisce in modo esplicito la situazione di "indefinito"

Risposte, in pseudo-codice (quello delle dispense):

- a) alla sorte succ corrisponde il tipo Succ dei record a 2 campi:
 - campo cont di tipo array di caratteri, con indici da 1 a K
 - campo lung di tipo integer che assume valori tra 0 e K
- b) alla successione vuota corrispondono i record con campo lung = 0
 alla succ <a, b> corrispondono i record con: lung = 2; cont[1] = 'a' ; cont[2] = 'b'
- c) per implementare l'operazione chop, come richiesto, ci sono varie possibilità;
 - la piú generale è:

```

procedura CHOP (      s : Succ          parametro IN-OUT
                    undef: boolean     parametro OUT      )
if s.lung = 0      then undef ← true
else { s.lung ← s.lung - 1 ; undef ← false }

```

Esercizio 6 (8 punti)

Considerate il "pezzo di programma" che segue (e che non fa nulla di interessante),

dove: aa è un array [1.. n] of integer e p, k, med sono variabili intere e usiamo le { } come in C.

Pezzo di programma

- (1) p ← n
- (2) med ← (1 + p) div 2
- (3) **while** p > med **do**
 - { (3.1) **per** k = med, med+1, ..., p : scrivi (aa[k])
*equivale al Pascal: **for** k:= med **to** p **do** write(aa[k])*
 - (3.2) vai a capo nell'output
 - (3.3) p ← med
 - (3.4) med ← (1 + p) div 2

Domanda:

Determinare la complessità del pezzo di programma, nel caso peggiore, possibilmente in $\Theta(\dots)$.

Non fare conti troppo dettagliati (esplicitando tutte le costanti,...),
ma non limitarsi nemmeno a dare il risultato, o a quattro chiacchiere;
in particolare: precisare se c'è un caso peggiore (o caso pessimo) e qual'è.

Risposta: la complessità è in $\Theta(n)$ e non c'è un caso peggiore, infatti:

Si vede subito che l'esecuzione non dipende dai valori contenuti nell'array e il costo dipende solo dal loro numero; non c'è dunque un caso peggiore.

Complessità:

- le istruzioni (1), (2), (3.2), (3.3), (3.4) sono a costo costante; valutare la condizione del while ha costo costante;
- la (3.1) è lineare in p , più precisamente: il costo è $a p/2 + b$ (con $a > 0$)
- allora tutto il corpo del while ha un costo che è: $a p/2 + c$ (con $a > 0$)
- il corpo del while si esegue per: $p = n$, $p = n/2$ (circa), $p = n/4$ (circa), $p = n/8$ (circa),... e quindi si esegue per circa $\log_2 n$ volte
- sommando tutte i costi delle esecuzioni di corpo del while + valutazione della condizione (diciamo che il costo è c') si ha:

$(a n/2 + c + c') + (a n/4 + c + c') + (a n/8 + c + c') + \dots$ [circa $\log_2 n$ addendi]

$= a (n/2 + n/4 + n/8 + \dots) + d \log_2 n + d'$ (d, d' costanti opportune)

- $n/2 + n/4 + n/8 + \dots = n$ (circa)
- quindi il tutto è in $\Theta(n)$

Non erano richiesti tutti i passaggi e tutti i dettagli, ma almeno alcuni sí !

Alternativa, più semplice:

- tutte le istruzioni "elementari" hanno costo costante e quindi la complessità dipende solo dal numero di volte che si esegue il corpo del while e, per ciascuna di queste, quello del for;
- se stabiliamo il numero di volte che si esegue l'istruzione `scrivi(aa[k])` sappiamo qual'è l'effetto combinato dei due cicli; cioè, l'esecuzione di `scrivi(aa[k])` è operazione dominante;

- il corpo del while si esegue per: $p = n$, $p = n/2$ (circa), $p = n/4$ (circa), $p = n/8$ (circa),.... e, per ciascun valore di p , l'istruzione `scrivi(aa[k])` si esegue $p/2$ volte (circa)
- quindi il numero di volte che si esegue `scrivi(aa[k])` è (circa):
$$n/2 + n/4 + n/8 + \dots = n$$
- dunque la complessità è in $\Theta(n)$.

Esercizio 7 (8 punti)

Consideriamo alberi binari con etichette intere, implementati nel modo solito:

un albero è un puntatore a nodo;

un nodo è un record con tre campi: info (di tipo integer), sin, des (di tipo albero).

La procedura che segue effettua una "visita parziale" (usiamo { } e -> come in C).

procedura vis (t : albero binario come sopra)

istruzioni:

```
if t non è vuoto then
  { scrivi ( t -> info )
    if ( t -> info ) < 0 then vis ( t -> sin)    else vis(t -> des)
  }
```

Domande. Consideriamo una generica chiamata vis(t) :

a) stimare la complessità in funzione dell'altezza, sia h, di t

b) stimare la complessità in funzione del numero di nodi, sia n, di t

Sia in a) che in b):

- complessità nel caso peggiore, possibilmente in $\Theta(\dots)$.
- **non fare conti troppo dettagliati** (esplicitando tutte le costanti,...),
ma non limitarsi nemmeno a dare il risultato, o a quattro chiacchiere;
in particolare, precisare se c'è un caso peggiore (o caso pessimo) e qual'è.

Risposta; a) $T_{alt}(h)$ è in $\Theta(h)$
b) $T_{nod}(n)$ è in $\Theta(n)$

sia in a) che in b) è importante individuare "il caso peggiore".

Infatti:

a) La procedura esegue una visita e, dalla radice, arriva ad una foglia seguendo un ramo dell'albero (ramo = cammino radice- foglia).

Tutti i costi (passaggio parametri, istruzioni diverse dalle chiamate,...) sono costanti e quindi il costo totale dipende solo dal numero di chiamate ricorsive.

Il numero di chiamate è uguale al numero di nodi presenti sul ramo che viene visitato, quindi uguale a $lunghezza+1$ (ma possiamo ignorare il "+1").

Il ramo seguito dipende dal valore delle etichette contenute nei nodi.

Poichè l'albero è un generico albero binario (quindi non necessariamente bilanciato) i rami possono avere lunghezza diversa; nel caso peggiore dobbiamo supporre che venga scelto un ramo di lunghezza massima, cioè uguale all'altezza dell'albero.

(E' facile capire che questo caso peggiore è effettivamente possibile.)

Allora il costo totale, nel caso peggiore, è lineare in h , quindi la complessità in funzione dell'altezza, indicata con $T_{alt}(h)$ è in $\Theta(h)$.

b) Da quanto visto sopra, il costo della procedura non dipende tanto dal numero di nodi, quanto dalla lunghezza dei rami.

Allora, tra gli alberi con n nodi, i "peggiori" sono quelli con rami più lunghi possibili.

L'albero con un solo ramo, dove ci sono tutti i nodi, è il peggiore ed ha altezza $n-1$.

Scegliendo, ad esempio, l'albero "tutto sbilanciato a sinistra" [a destra] con etichette dei nodi negative [positive] otteniamo che la visita percorre tutto il ramo di lunghezza $n-1$.

Allora il costo totale, nel caso peggiore, è lineare in n , quindi la complessità in funzione del numero di nodi, indicata con $T_{nod}(n)$ è in $\Theta(n)$.