ASD1: Correzione della Prima Provetta (2002-2003)

Esercizio 1. *Dire quali delle seguenti affermazioni sono vere. Ove false, fornire un controesempio.*

1. $f(n) = o(g(n))$ *implica* $f(n) = O(g(n))$.
2. $f(n) = O(g(n))$ *implica* $f(n) = o(g(n))$.
3. $f(n) = O(f(2n))$.
4. $f(n) = O(g(n))$ *implica* $2^{f(n)+3} = O(2^{g(n)+2})$.

Svolgimento Esercizio 1

1. $f(n) = o(g(n))$ implica $f(n) = O(g(n))$?
   Vero. La dimostrazione formale, non richiesta, sarebbe la seguente. Sappiamo che, per ogni $c > 0$, esiste un n_0 tale che $\forall n \geq n_0, 0 \leq f(n) \leq cg(n)$. Risulta partento vero che esiste un $c > 0$, ed un n_0, tale che $\forall n \geq n_0, 0 \leq f(n) \leq cg(n)$. Pertanto, $f(n) = O(g(n))$.
2. $f(n) = O(g(n))$ implica $f(n) = o(g(n))$.
   Falso, come si vede prendendo $f(n) = g(n) = n$.
3. $f(n) = O(f(2n))$.
   Falso. Basta infatti prendere una qualsiasi $f(n)$ che non sia asintoticamente non-negativa per rendere falso che $f(n) = O(f(2n))$. Questo controesempio era però un poco come barare. Si consideri allora $f(n) = \frac{1}{2^n}$. Questa funzione è sempre non negativa e tuttavia $\lim_{n\to\infty} \frac{f(n)}{f(2n)} = \lim_{n\to\infty} \frac{1/2^n}{1/2^{2n}} = \lim_{n\to\infty} 2^n = \infty$.
4. $f(n) = O(g(n))$ implica $2^{f(n)+3} = O(2^{g(n)+2})$.
   Falso, come si vede prendendo $f(n) = 2n$ e $g(n) = n$.

Esercizio 2. *Si assuma* $f(n) = O(g(n))$. *Dimostrare che* $f(n) + g(n) = \Theta(g(n))$.

Svolgimento Esercizio 2
Sappiamo che esiste una costante $c > 0$ ed un n_0 tale che $\forall n \geq n_0, 0 \leq f(n) \leq cg(n)$. Pertanto, per ogni $n \geq n_0$, $0 \leq g(n) \leq f(n) + g(n) \leq (c + 1)g(n)$. Ciò dimostra che $f(n) + g(n) = \Theta(g(n))$.

Esercizio 3. *Ordinare le seguenti funzioni per ordine di crescita asintotico non decrescente. Ve ne sono alcune che presentano lo stesso ordine di crescita?* $f(n) = 3^{\log_2 n}$, $f(n) = 3^{\log_3 n}$, $f(n) = 3^{\log_4 n}$, $f(n) = \log_2 n$, $f(n) = \log_3 n$,

Svolgimento Esercizio 3
Si noti che, per ogni $n \geq 1$,

$$\log_3 n \leq \log_2 n \leq 3^{\log_4 n} \leq 3^{\log_3 n} \leq 3^{\log_2 n}$$

Riponendo le funzioni proposte secondo tale ordine, abbiamo

$$
\begin{aligned}
f_1(n) &= \log_3 n = (\log_3 2)\log_2 n,\\
f_2(n) &= \log_2 n,\\
f_3(n) &= 3^{\log_4 n} = 3^{\log_4 3 \log_3 n} = \left(3^{\log_3 n}\right)^{\log_4 3} = n^{\log_4 3},\\
f_4(n) &= 3^{\log_3 n} = n,\\
f_5(n) &= 3^{\log_2 n} = 3^{\log_2 3 \log_3 n} = \left(3^{\log_3 n}\right)^{\log_2 3} = n^{\log_2 3}.
\end{aligned}
$$

Da cui risulta evidente che

$$
\begin{aligned}
f_1(n) &= \Theta(f_2(n)),\\
f_2(n) &= o(f_3(n)),\\
f_3(n) &= o(f_4(n)),\\
f_4(n) &= o(f_5(n)).
\end{aligned}
$$

Volendo verificarlo ad un più basso livello, abbiamo infatti che

$$
\lim_{n\to\infty} \frac{f_2(n)}{f_1(n)} = \lim_{n\to\infty} \frac{\log_2 n}{\log_3 2 \log_2 n} = \lim_{n\to\infty} \log_2 3 = \log_2 3 > 0.
$$

L'unico caso critico è il confronto tra $f_2(n) = \log_2 n$ ed $f_3(n) = n^{\log_4 3}$ che non differiscono per una costante moltiplicativa. La differenza nel loro ordine di crescita asintotico può essere messa in evidenza utilizzando il teorema dell'Hôpital, ed osservando che $\log_4 3 > 0$

$$
\lim_{n\to\infty} \frac{f_2(n)}{f_3(n)} = \lim_{n\to\infty} \frac{\log_2 n}{n^{\log_4 3}} = \lim_{n\to\infty} \frac{1/n}{(\log_4 3) n^{(\log_4 3)-1}} = \lim_{n\to\infty} \frac{1}{(\log_4 3) n^{\log_4 3}} = 0
$$

I restanti due casi sono banali, simili al primo, e del tutto analoghi tra di loro. Ne affronteremo pertanto solo uno, e lasciamo l'altro come esercizio mentale.

$$
\lim_{n\to\infty} \frac{f_3(n)}{f_4(n)} = \lim_{n\to\infty} \frac{n^{\log_4 3}}{n} = \lim_{n\to\infty} \frac{1}{n^{1-\log_4 3}} = 0
$$

poichè $\log_4 3 < 1$.

Esercizio 4. *Si dimostri che* $\log[(n+1)!] = \Theta(n \log n)$.

Svolgimento Esercizio 4

Avevamo visto a lezione che $\log(n!) = \Theta(n \log n)$. Come prima cosa, riconduciamoci pertanto a quanto per noi già noto. A tal fine, basta una piccola manipolazione algebrica:

$$
\log[(n+1)!] = \log[(n+1)(n!)] = \log(n+1) + \log(n!) = \Theta(\log n) + \Theta(n \log n) = \Theta(n \log n).
$$

Questa era già una risposta parziale all' esercizio, ma per completarlo, dobbiamo mostrare anche che $\log(n!) = \Theta(n \log n)$.

Chiaramente,

$$
\log(n!) = \log \prod_{i=1}^{n} i = \sum_{i=1}^{n} \log i \leq \sum_{i=1}^{n} \log n = n \log n,
$$

da cui $\log(n!) = O(n \log n)$ disegue. Resta pertanto da dimostrare che $\log(n!) = \Omega(n \log n)$.

$$
\log(n!) = \log \prod_{i=1}^{n} i = \sum_{i=1}^{n} \log i \geq \sum_{i=\lceil n/2 \rceil}^{n} \log i \geq \sum_{i=\lceil n/2 \rceil}^{n} [(\log n) - 1] = \Omega(n \log n),
$$

Esercizio 5. *Allo scopo di tabulare i valori della seguente ricorrenza*

$$C_n = C_{n-1} + C_{n-1} \qquad \text{per } n \geq 1, \text{ con } C_0 = 1$$

si consideri la seguente procedura iterativa.

```
#include<iostream.h>

int main() {
  int n; cout << "Dammi n: "; cin >> n; cout << endl;
  long long int T[n +1];   T[0] = 1;
  cout << "T[0] = "  << T[0] << endl;
  for(int k=1; k<=n; k++) {
    T[k] = T[k-1]+T[k-1];
    cout << "T[" << n << "] = "  << T[n] << endl;
  }
}
```

Si stabilisca l'ordine di crescita del tempo di calcolo della procedura proposta. Si stabilisca l'ordine di crescita della memoria impiegata dalla procedura proposta.

Svolgimento Esercizio 5

La memoria occupata è sicuramente $\Omega(n)$, causa l'allocazione `long long int T[n +1]` alla seconda linea del codice proposto. Infatti, tale linea viene eseguita incondizionatamente, ad ogni immissione di un input corretto.

La memoria occupata è di fatto $\Theta(n)$, poichè se andiamo a considerare le altre allocazioni di memoria esplicitamente od implicitamente comportate dal nostro codice, risulta evidente che, per ogni realizzazione ragionevole di un compilatore, esse saranno in numero costante $\Theta(1)$, e non dipenderanno cioè dal valore n in input.

Il tempo di calcolo impiegato è sicuramente $\Omega(n)$, causa il ciclo `for(int k=1; k<=n; k++)` presente alla quarta linea del codice proposto. Infatti tale ciclo for viene eseguito incondizionatamente, e per intero, ad ogni immissione di un input corretto.

Il tempo di calcolo impiegato è di fatto $\Theta(n)$, poichè le istruzioni contenute nel ciclo `for(int k=1; k<=n; k++)` sono eseguite $\Theta(n)$ volte, mentre quelle esterne al ciclo sono eseguite 1 volta. Inoltre, tutte le istruzioni, è ragionevole aspettarsi, richiedano $\Theta(1)$. A dire il vero, per l'istruzione, `long long int T[n +1]`, uno potrebbe chiedersi se essa non possa richiedere più di $\Theta(1)$ tempo. Tuttavia, è ragionevole ipotizzare quantomeno che il tempo richiesto da tale istruzione sia $O(n)$, e si noti che tale istruzione è esterna al *ciclo for* ed è pertanto eseguita un'unica volta.

Esercizio 6. *Il professor Tortoise, propone di utilizzare la seguente procedura ricorsiva allo scopo di tabulare la stessa ricorrenza ($C_n = C_{n-1} + C_{n-1}$).*

```
#include<iostream.h>

long long int T(int n) {
  if(n==0) return 1;
  return T(n-1) + T(n-1);
}

int main() {
  int n; cout << "Dammi n: "; cin >> n; cout << endl;
  cout << "T[" << n << "] = "  << T(n) << endl;
}
```

Si stabilisca l'ordine asintotico di crescita per il tempo di calcolo della procedura proposta dal professor Tortoise. Si stabilisca l'ordine di crescita della memoria impiegata dalla procedura di Tortoise.

Svolgimento Esercizio 6

Per il tempo di calcolo, possiamo scrivere la seguente ricorrenza,

$$T(n) = T(n-1) + T(n-1) + 1 = 2T(n-1) + 1$$

Ovviamente, $T(n) = \Theta(2^n)$. Possiamo infatti verificare per sostituzione sia che $T(n) \geq 2^n$:

$$T(n) = 2T(n-1) + 1 \geq 2T(n-1) \geq 22^{n-1} = 2^n$$

sia che $T(n) \leq 2^n - 1$:

$$T(n) = 2T(n-1) + 1 \leq 2(2^{n-1} - 1) + 1 = 2^n - 2 + 1 = 2^n - 1$$

La contraddizione $2^n \leq T(n) \leq 2^n - 1$ è solo apparente ed è dovuta al fatto che abbiamo lavorato prescindendo dalla base dell'induzione.

Per quanto riguarda l'occupazione di memoria dobbiamo prestare attenzione al fatto che ad ogni chiamata di procedura viene allocata una certa quantità di memoria. Nel caso della nostra semplice procedura `long long int T(int n)`, tale quantità di memoria risulta essere $\Theta(1)$. Quando la procedura termina, tale quantità di memoria viene però rilasciata. E tuttavia singole quantità di memoria allocate a fronte di chiamate diverse vengono a sommarsi se la seconda chiamata avviene prima che la prima chiamata sia già stata esaurita. Siamo cioè chiamati a valutare l'altezza dell'albero di ricorsione, che corrispende al massimo numero di ambienti che dovranno essere mantenuti aperti contemporaneamente. Nel nostro caso, l'altezza dell'albero di ricorsione risulta essere n. Pertanto, l'occupazione di memoria sarà $n\Theta(1) = \Theta(n)$.

Complemento 6+ *Per questa particolare ricorrenza ($C_n = C_{n-1} + C_{n-1}$), è possibile pensare ad un accorgimento che consenta di ottenere un buon tempo di calcolo anche per la procedura ricorsiva. Sapresti indicare quale?*

Svolgimento Complemento 6+

Si vede che la procedura può facilmente evitare di fare due chiamate ricorsive. Basta infatti sostituire, internamente alla procedura ricorsiva `long long int T(int n)`, la riga di codice

```
return T(n-1) + T(n-1);
```

con la riga di codice

```
return 2*T(n-1);
```

così che la ricorrenza per il tempo di calcolo diviene

$$T(n) = T(n-1) + \Theta(1)$$

portando quindi ad un tempo di calcolo che sarà $\Theta(n)$ come è facile evidenziare srotolando la ricorrenza.

Purtroppo, tale modifica non migliora la situazione per quanto concerne l'occupazione di memoria.

Sapresti, come ulteriore esercizio, modificare la versione iterativa di cui all'Esercizio 5, ottenendo una soluzione algoritmica che richieda solamente $\Theta(1)$ come occupazione di memoria, senza per altro peggiorare il running time (anzi, di fatto migliorando anche esso, anche se non dal punto di vista asintotico)?