Problem Set 1

docente: Luciano Gualà

Esercizio 1 (*notazione asintotica*)
Siano $f(n), g(n), h(n)$ tre funzioni asintoticamente positive. Inoltre, sia $c > 1$ una costante reale positiva. Si dimostrino o confutino le seguenti affermazioni:

1. $2^{f(n)+2^c} = \Theta(2^{f(n)})$.
2. $g(n) = \Theta(1)$ implica $2^{f(n)+g(n)} = O(2^{f(n)})$.
3. $g(n) = o(f(n))$ implica $2^{f(n)+g(n)} = O(2^{f(n)})$.
4. $f(n) + g(n) + h(n) = \Theta(\max\{f(n), g(n), h(n)\})$.
5. $f(n) = \Theta(\log n)$ implica $\log n^{f(n)} = O(\log^c n^{g(n)})$.
6. $f(n) = \Theta(f(c \cdot n))$.
7. $f(n) = \Theta(f(c + n))$.

Soluzione Esercizio 1

1. *Vera.* Infatti $2^{f(n)+2^c} = 2^{f(n)}2^{2^c} = \Theta(2^{f(n)})$, perché 2^{2^c} è una costante.
2. *Vera.* Infatti, $g(n) = \Theta(1)$ implica che esistono due costanti $c > 1, n_0$ tale che $g(n) \leq c$ per ogni $n \geq n_0$. Quindi, quando $n \geq n_0$, abbiamo $2^{f(n)+g(n)} = 2^{f(n)}2^{g(n)} \leq 2^{f(n)}2^c = O(2^{f(n)})$, perché 2^c è una costante.
3. *Falsa.* Un controesempio è: $g(n) = \sqrt{n}$, $f(n) = n$. Chiaramente $2^{n+\sqrt{n}} = 2^n 2^{\sqrt{n}} = \omega(2^n)$.
4. *Vera.* Infatti, poiché le funzioni sono asintoticamente positive, sicuramente abbiamo che per n sufficientemente grande:

   $$\max\{f(n), g(n), h(n)\} \leq f(n) + g(n) + h(n) \leq 3\max\{f(n), g(n), h(n)\},$$

   da cui segue la tesi (le costanti della definizione di $\Theta(\cdot)$ sono $c_1 = 1$ e $c_2 = 3$ e n_0 il valore dopo il quale tutte le funzioni sono sempre positive).
5. *Falsa.* Poiché si ha che $\log n^{f(n)} = f(n)\log n$ e $\log^c n^{g(n)} = g(n)^c \log^c n$, un controesempio è: $f(n) = \log n$, $g(n) = 1$ e $c = 1.5$.
6. *Falsa.* Un controesempio è $f(n) = 2^n$ e $c = 2$. Infatti $2^n = o(2^{2 \cdot n})$.
7. *Falsa.* Un controesempio è $f(n) = 2^{2^n}$ e $c = 2$. Infatti $2^{2^n} = o(2^{2^{n+2}})$, perché $2^{2^{n+2}} = 2^{2^n 4} = (2^{2^n})^4$.

Esercizio 2 (*trovare l'intero mancante*)
Sia $A[1 : n]$ un vettore ordinato di n interi distinti compresi fra 1 e $n + 1$. Chiaramente A contiene tutti gli elementi dell'insieme $\{1, 2, \ldots, n+1\}$ tranne uno. Progettare un algoritmo con complessità temporale $o(n)$ che trova l'elemento mancante.

Soluzione Esercizio 2 Descriviamo un algoritmo di complessità temporale $O(\log n)$ che restituisce l'intero mancante. L'idea è quella di usare l'approccio della ricerca binaria. Infatti, guardando una coppia adiacente di elementi, per esempio in posizione $i-1$ e i possiamo capire se i è l'elemento mancante o se l'elemento mancante è più grande di i (e quindi dobbiamo cercarlo alla destra di questi elementi) o se è più piccolo (e quindi dobbiamo cercarlo alla sinistra). Lo pseudocodice riportato di seguito si spiega da solo. L'unico dettaglio tecnico del codice è il seguente: poiché ad ogni passo stiamo guardando due elementi vicini, dobbiamo stare attenti che gli indici dell'array non escano dal range. Questo è ottenuto, nel codice, controllando a parte se l'elemento mancante è 1. Un'altra cosa che per comodità tecnica è fatta a parte è contollare se l'elemento mancante non sia $n+1$. La complessità temporale dell'algoritmo è uguale a quella della ricerca binaria e quindi è $O(\log n)$.

Algorithm 1: `InteroMancante`(A)

```
n = lunghezza di A ;
if A[1] = 2 then
 ⌞ return 1
if A[n] = n then
 ⌞ return n + 1
return InteroMancanteRic(A, 2, n)
```

Dove, la procedura ricorsiva ausiliaria è:

Algorithm 2: `InteroMancanteRic`(A, i, j)

```
m = ⌊(i+j)/2⌋ ;
if A[m] = m + 1 e A[m − 1] = m − 1 then
 ⌞ return m
if  A[m − 1] = m then
 |  return InteroMancanteRic(A, i, m − 1)
else
 ⌞  return InteroMancanteRic(A, m + 1, j)
```

Esercizio 3 (*trovare l'intero mancante: un gioco di prestigio*)
Un vostro amico sa fare il seguente gioco di magia. Per un certo n, vi chiede di scegliere un valore nell'insieme $\{1, 2, \ldots, n+1\}$ senza dirlo, e di elencare, in un qualsiasi ordine, gli n elementi restanti. Quando voi avete finito di elencare gli elementi lui sa indovinare l'elemento che manca. La cosa che trovate sorprendente è che il vostro amico è in grado di eseguire questo gioco per valori di n piuttosto grandi (una volta ve l'ha fatto per $n = 200$), senza usare fogli su cui scrivere e facendo passare poco tempo fra un numero e l'altro. Ora, siete sicuri che il vostro amico non è in grado di ricordare tutti i numeri che gli snocciolate e quindi deve esserci un trucco. Ma quale? Forse ragionare in modo algoritmico può aiutarvi.

Per essere più precisi, considerate questo problema. Avete un vettore $A[1 : n]$ non ordinato di n interi distinti compresi fra 1 e $n+1$. Progettate un algoritmo che scorre A una sola volta da sinistra a destra e che alla fine calcola l'elemento mancante. L'algoritmo deve avere complessità temporale $O(n)$ e deve usare memoria ausiliaria costante.

Soluzione Esercizio 3 L'algoritmo è molto semplice: mentre si leggono i numeri si mantiene una variabile con la somma degi interi visti fino a quel momento. Sia x il valore della somma di tutti gli elementi del vettore. Ora, se non mancasse nessun intero, il valore x sarebbe uguale a $S = \sum_{i=1}^{n+1} i = \frac{(n+2)(n+1)}{2}$. Però sappiamo che manca un elemento e questo, chiaramente, può essere derivato per differenza, ovvero, l'intero mancante è $S - x$. Quindi l'algoritmo deve solo calcolare x mentre scandisce il vettore, e questo può essere fatto in tempo lineare e memoria costante (dobbiamo mantenere una sola variabile). Il vostro amico, nell'eseguire il trucco deve solo saper fare velocemente le somme e poi, alla fine, una sottrazione.

Esercizio 4 Si consideri una tavoletta di cioccolata rettangolare composta da n file di m quadratini di cioccolata. Si vuole spezzarla in modo da avere tutti i quadratini di cioccolata separati. Una strategia consiste in una serie di *spezzate*, dove ogni spezzata può essere vista come una procedura che prende un pezzo di cioccolata (di qualsiasi forma) e lo separa in due pezzi di cioccolata (di qualsiasi forma). Una semplice strategia è quella di separare prima le n file eseguendo $n-1$ spezzate orizzontali, e poi per ognuna delle n file eseguire $m-1$ spezzate verticali per separare i relativi quadratini. Questa strategia esegue complessivamente:

$$n - 1 + n(m - 1) = n - 1 + nm - n = nm - 1$$

spezzate. Esiste una strategia migliore, cioè una strategia che separa tutti i quadratini eseguendo un numero minore di spezzate? Si argomenti la risposta.

Solizione Esercizio 4 Dimostriamo che *ogni* strategia effettua esattamente $mn - 1$ spezzate, e che quindi la strategia proposta è ottima. Forniremo due argomentazioni.

Prima argomentazione. E' possibile vedere una strategia di spezzate come un albero binario (radicato) T con le seguenti caratteristiche. Ogni nodo rappresenta un pezzo di cioccolata dove: (i) la radice rappresenta la tavoletta intera, e (ii) se un nodo interno rappresenta un certo pezzo x, i suoi due figli rappresentano i due pezzi y e y' ottenuti una volta che la data strategia ha spezzato x. Ora è facile vedere che T ha esattamente mn foglie, una per ogni quadratino, e che ogni nodo interno ha esattamente due figli. Inoltre il numero di spezzate è proprio uguale al numero dei nodi interni. A questo punto la domanda è: quanti nodi interni ha l'albero corrispondente a una generica strategia? Essi sono sempre $mn - 1$, perché sono esattamente uno in meno del numero di foglie (Lemma 1.2 del libro di testo).

Seconda argomentazione. E' possibile ragionare nel seguente modo. Per come abbiamo definito le regole, ogni volta che si effettua una spezzata, il numero complessivo di pezzi aumenta di (esattamente) uno, perché prendiamo un pezzo e lo dividiamo in due. All'inizio il numero di pezzi di cioccolata è uguale a 1, perché la tavoletta è intera, e alla fine il numero di pezzi è esattamente mn. Quindi il numero di spezzate deve essere $mn - 1$.