Problem Set 2

docente: Luciano Gualà

Esercizio 1 (*equazioni di ricorrenza*)
Si risolvano le seguenti equazioni di ricorrenza. Si assuma sempre $T(1) = 1$.

(*a*) $T(n) = T(n-10) + 10$.

(*b*) $T(n) = T(n/2) + 2^n$.

(*c*) $T(n) = T(n/3) + T(n/6) + n^{\sqrt{\log n}}$.

(*d*) $T(n) = T(\sqrt{n}) + \Theta(\log \log n)$.

(*e*) $T(n) = T(n/2 + \sqrt{n}) + \Theta(1)$.

(*f*) $T(n) = \sqrt{n}T(\sqrt{n}) + n$.

Esercizio 2 Un array A di n elementi è detto *unimodale* se consiste di una sequenza crescente seguita da una sequenza decrescente o più precisamente se esiste un indice $m \in \{1, 2, \ldots, n\}$ tale che:

- $A[i] < A[i+1]$, per ogni $1 \leq i < m$, e
- $A[i] > A[i+1]$, per ogni $m \leq i < n$.

In particolare $A[m]$ è il massimo elemento ed è l'unico che è circondato da due elementi più piccoli ($A[m-1]$ e $A[m+1]$).

(*a*) Si progetti un algorimo con complessità temporale $o(n)$ che, dato un array unimodale A, restituisce l'indice dell'elemento massimo.

(*b*) Si progetti un algorimo con complessità temporale $o(n \log n)$ che ordina (in ordine crescente) un array unimodale A.

Esercizio 3 L'elemento *mediano* di un insieme S di m elementi (distinti) è quell'elemento che ha esattamente $\lfloor m/2 \rfloor$ elementi minori in S. Per esempio, l'insieme $S = \{1, 4, 6, 8, 9, 12\}$ ha mediano 8. Siano dati due insiemi A e B di n elementi ciascuno, rappresentati come sequenze ordinate memorizzate negli array $S_A[1:n]$ e $S_B[1:n]$. Progettare un algoritmo che trovi il mediano di $A \cup B$ in tempo $O(n)$ e che usi memoria ausiliaria $O(1)$. Si faccia attenzione al fatto che i due insiemi possono avere intersezione non vuota.

Esercizio 4 (*sui modelli di calcolo e i numeri di Fibonacci*)
Nell'analizzare la complessità temporale degli algoritmi per calcolare l'n-esimo numero di Fibonacci, abbiamo stimato il numero (asintotico) di linee di codice eseguite dai vari algoritmi in funzione del valore n preso in ingresso dall'algoritmo stesso. Ora che conosciamo cosa si intende per modello di calcolo, ci rendiamo conto che le complessità temporali sono state derivate assumendo un modello di calcolo RAM a *costi uniformi*. Sfortunatamente,

tale modello non è particolarmente adatto quando si ha a che fare con algoritmi che, come nel caso del calcolo dei numeri di Fibonacci, devono gestire interi il cui valore cresce esponenzialmente all'aumentare del valore di input.

Questo esercizio vi chiede di porre rimedio a questa imprecisione. In particolare, si chiede di derivare nuovamente un upper bound asintotico al tempo di esecuzione degli algoritmi `Fibonacci2`, `Fibonacci3`, e `Fibonacci6`, nei due seguenti modelli di calcolo (di fatto più ragionevoli per il problema in esame):

- Il primo modello è quello RAM a *costi logaritmici*. In tale modello, ogni operazione (di lettura, scrittura, somma e prodotto) che coinvolge un operando di valore x ha un costo (temporale) pari a $\Theta(\log x)$, o equivalentemente ha un costo proporzionale al numero di bit necessari per rappresentare x.

- In alcuni casi il modello precedente risulta essere ancora impreciso, perché non distingue fra i costi delle varie operazioni. Di fatto, l'operazione di prodotto fra due interi è un'operazione più costosa rispetto all'operazione di somma. Mentre è possibile sommare due numeri di k bit in tempo $O(k)$[1], non è noto un algoritmo con la stessa complessità per moltiplicare due numeri di k bit. Si assuma che il prodotto fra due interi di k bit possa essere calcolato in tempo $O(k^{\log_2 3})$. Come cambiano le complessità temporali dei vari algoritmi?

In entrambi i modelli considerati, si può dire ancora che `Fibonacci3` è esponenzialmente più veloce di `Fibonacci2`? E `Fibonacci6` di `Fibonacci3`?

[1] Abbiamo imparato come si fa alle scuole elementari: si allineano i due numeri e si sommano cifra-cifra dalla meno significativa alla più significativa.