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
Siete interessati ad analizzare alcuni dati che sono difficili da ottenere e che sono distribuiti su due database differenti. Ogni database contiene n valori numerici – così i valori in totale sono $2n$ – e potete assumere che sono valori tutti distinti. Voi volete determinare il *mediano* di questi valori, ovvero l'n-esimo valore più piccolo. L'unico modo che avete per accedere ai database è attraverso delle *query*. In ogni singola query potete specificare uno dei due database e un valore k, e ottenere come risposta alla query il k-esimo valore più piccolo del database che state interrogando. Poiché le query sono costose, volete farne il meno possibile. Progettare un algoritmo che è in grado di trovare il mediano facendo nel caso peggiore $O(\log n)$ query.

Esercizio 3 (*battaglia navale*)
Un array $A[1:n]$ di n elementi è riempito interamente di zeri, tranne che in una porzione di ℓ celle contigue che contengono degli uni. Progettare un algoritmo che in tempo $O(\frac{n}{\ell})$ trovi almeno un indice i tale che $A[i] = 1$. Si fornisca una soluzione prima assumendo di conoscere la lunghezza ℓ della porzione contigua di uni, poi si rimuova questa assunzione e si progetti un algoritmo che sempre in tempo $O(\frac{n}{\ell})$ trovi almeno la posizione di un 1 senza conoscere ℓ.

Esercizio 4 (*test di infrangibilità di bicchieri*)
Dovete valutare l'infrangibilità di un certo tipo di bicchiere di vetro e organizzate un test per determinare l'altezza massima da cui potete far cadere il bicchiere senza che esso si rompa. Per il test, avete a disposizione una scala con n pioli e voi dovete capire quale è il più alto piolo da cui è possibile far cadere il bicchiere senza conseguenze. Chiameremo questo piolo il *più alto piolo sicuro*.

Ora, essendo dei bravi algoritmisti, se aveste a disposizione tante copie dello stesso tipo di bicchiere, al fine di fare pochi lanci, simulereste una ricerca binaria. Quindi lascereste cadere un bicchiere dal piolo di altezza $n/2$ e, in base all'esito, passereste a provare il piolo ad altezza $n/4$ o $3n/4$, e così via. Questo vi consentirebbe di trovare il più alto piolo sicuro facendo $O(\log n)$ lanci ma potreste rompere molti bicchieri.

Se invece il vostro obiettivo primario fosse quello di conservare quanti più bicchieri potete, allora la strategia migliore sarebbe quella di provare in sequenza il piolo 1, poi il

piolo 2, e così via, fino a trovare il più alto piolo sicuro. Questo sacrificherebbe un solo bicchiere ma vi costerebbe in termini di tempo, perché nel caso peggiore potreste dover eseguire un numero lineare di lanci.

In questo esercizio vi si chiede di trovare una soluzione che bilancia il numero di lanci con il numero di bicchieri che potete rompere. Per essere più precisi, considerate il caso in cui avete a disposizione un numero massimo $k \geq 1$ di bicchieri che potete rompere, e voi volete capire quale è il più alto piolo sicuro effettuando meno lanci possibile. In particolare:

(a) Supponete di avere a diposizione $k = 2$ bicchieri. Trovate una strategia che risolva il problema eseguendo al pià $f(n)$ lanci, per una qualche funzione $f(n) = o(n)$.

(b) Ora assumete di avere a disposizione $k > 2$ bicchieri. Descrivete una strategia per trovare velocemente il più alto piolo sicuro utilizzando al più k bicchieri. Se $f_k(n)$ denota il numero di lanci che occorrono alla vostra strategia per un certo k, allora le funzioni $f_1(n), f_2(n), \ldots$ dovrebbero avere la caratteristica di possedere un tasso di crescita via via più basso al crescere di k, ovvero, dovrebbe valere che $f_k(n) = o(f_{k-1}(n))$, per ogni k.