Esercizi sulla complessità computazionale

Andrea Ciprietti

27 novembre 2021

Gli esercizi indicati con una $\star$ sono da considerarsi più difficili degli altri.

Esercizio 1. Farsi un'idea di cosa s'intende per *complessità computazionale* (non serve una definizione precisa) e del significato delle notazioni $\mathcal{O}$ ("O grande"), Ω e Θ.

Esercizio 2. Dimostrare che l'algoritmo implementato nel seguente codice `C++` ha complessità computazionale $\Theta(n^5)$ (dove n è un intero positivo fissato):

```
for (int a = 0; a < n; a++) {
    for (int b = 0; b < a; b++) {
        for (int c = 0; c < b; c++) {
            for (int d = 0; d < c; d++) {
                for (int e = 0; e < d; e++) {
                    cout << "Un'altra␣iterazione" << endl;
                }
            }
        }
    }
}
```

(La spiegazione deve essere *formale*; non basta dire "Sono cinque cicli `for` annidati".)

Esercizio 3. Nel seguito $\lfloor x \rfloor$ e $\lceil x \rceil$ denotano, rispettivalemte, le funzioni *parte intera* (il più grande intero $\leq x$) e *parte intera superiore* (il più piccolo intero $\geq x$).

(a) Sia k un intero positivo, e sia $d(k)$ il numero di divisori positivi di k. Dimostrare che $d(k) \leq 2\lfloor\sqrt{k}\rfloor$. Provare inoltre che non esiste nessuna costante $c > 0$ tale che $d(k) \leq c\log_2(k)$ per ogni k.

(b) Dimostrare che, dato un intero positivo n,

$$\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \leq \lceil \log_2(n) \rceil.$$

(*Hint:* dividere gli addendi in blocchi lunghi 1, 2, 4, ...)

(c) Dedurne che

$$d(1) + d(2) + d(3) + \cdots + d(n) \leq n\lceil \log_2(n) \rceil.$$

Notare in particolare, dal punto (a), che non sarebbe stato possibile ottenere questa stima analizzando i termini individualmente.

Esercizio 4. ($\star\star$) Camilla ha una barretta di cioccolato lunga n quadratini (e larga 1 quadratino). I suoi amici si mettono in fila e, uno alla volta, le chiedono un pezzo della barretta. Camilla, che è molto generosa (ma non troppo), ogni volta spezza la barretta in due parti e dà al suo amico il pezzo più corto, tenendo quello più lungo per sé. Se, in un certo momento, la barretta è lunga $k \geq 2$ quadratini, quando Camilla la spezza tutte le possibili suddivisioni hanno la stessa probabilità, pari a $\frac{1}{k-1}$.

Dimostrare che esiste una costante $c > 0$ tale che la probabilità che Camilla si ritrovi con un solo quadratino dopo aver spezzato la barretta più di $100 \log_2(n)$ volte è al massimo $\frac{c}{n}$.

(*Hint:* scegliere un intero positivo $t \geq 2$ e osservare che la probabilità che, dopo uno spezzamento, la barretta si riduca di un fattore $\frac{t-1}{t}$ è circa $1 - \frac{2}{t}$; scegliere poi $m = 2 \log_{t/(t-1)}(n)$ e stimare la probabilità che la barretta si riduca a un quadratino dopo non meno di m mosse.)

Esercizio 5. ($\star$) Analizzare la complessità di una chiamata alla seguente funzione ricorsiva:

```
void strange_function(int n) {
    cout << "Un'altra chiamata" << endl;
    for (int i = 0; i < n; i++) {
        strange_function(i);
    }
}
```

Confrontare poi con l'esercizio 2.

Esercizio 6. ($\star$) Supponiamo data in input una permutazione A degli interi 1, 2, ..., n, ovvero un array di dimensione n che contiene ciascun numero tra 1 ed n una e una sola volta.

(a) Dimostrare che l'algoritmo implementato nel seguente codice C++ è $\mathcal{O}(n \log(n))$:

```
int k = n + 1;
int i = 0;

while (k > 1) {
    if (k/2 <= A[i] and A[i] < k) {
        k = A[i];
    }
    i = (i + 1) % n;
}
```

(b) Dimostrare che nel caso pessimo l'algoritmo di cui sopra è $\Omega(n \log(n))$.