Algoritmi e Principi dell'Informatica
Appello del 7 febbraio 2019
Chi deve sostenere l'esame integrato (API) deve svolgere tutti gli esercizi in 2 ore.
Chi deve sostenere solo il modulo di *Informatica teorica* deve svolgere gli Esercizi 1 e 2 in 1 ora.
Chi deve sostenere solo il modulo di *Informatica 3* deve svolgere gli Esercizi 3 e 4 in 1 ora.
NB: i punti attribuiti ai singoli esercizi hanno senso solo con riferimento all'esame integrato e hanno valore puramente indicativo.

Esercizio 1 (7 punti)
Utilizzare un formalismo a potenza minima (tra tutti quelli visti a lezione) che caratterizzi il linguaggio costituito da tutte e sole le stringhe binarie tali per cui il numero di 0 nella stringa sia divisibile per 5.

Esercizio 2 (9 punti)
Si consideri la funzione $f(x,y) = f_z(x)$, dove $z = f_y(y)$. f è computabile? Motivare brevemente la risposta.

Esercizio 3 (8 punti)
Si consideri il seguente codice:

```
FUN(A) :
    if A.length < 5 then return FUN1(A)
    a = A.length-1
    FUN(A[0..a-1])
    FUN(A[1..a])
    FUN2(A)
```

Sapendo che FUN1 ha complessità $\Theta(n)$ e FUN2 ha complessità $\Theta(n^2)$, dove n è la lunghezza dell'array passato a entrambi come argomento, si valuti la complessità temporale di FUN fornendo un estremo superiore asintotico.

Esercizio 4 (8 punti)
Siano dati k array di n elementi ciascuno, già ordinati. Si descriva, preferibilmente mediante pseudocodice, un algoritmo per ottenere un array ordinato contenente tutti e soli gli elementi degli array dati, valutandone la complessità in funzione di k e n.

Algoritmi e Principi dell'Informatica, *English version*
February 14, 2019
The total available time is 2h.
If you are registered for the first module only (*Informatica Teorica*), you must solve Exercises 1 and 2 in 1h.
If you are registered for the second module only (*Informatica 3*), you must solve Exercises 3 and 4 in 1h.
NB: points assigned to individual exercises are intended w.r.t. the entire exam and are only indicative.

Exercise 1 (7 points)
Build a formal model with the minimum computational power among those considered in this course that defines the language over the alphabet {0, 1} consisting of all and only the strings containing a number of '0' that is a multiple of 5.

Exercise 2 (9 points)

Consider the function $f(x,y) = f_z(x)$, where $z = f_y(y)$. Is f computable? Briefly explain you answer.

Exercise 3 (8 points)

Consider the following code:

```
FUN(A) :
    if A.length < 5 then return FUN1(A)
    a = A.length-1
    FUN(A[0..a-1])
    FUN(A[1..a])
    FUN2(A)
```

Knowing that FUN1's complexity is $\Theta(n)$ and FUN2's one is $\Theta(n^2)$, where n is the length of the array that they receive as parameter, evaluate FUN's complexity and provide an asymptotic upper bound thereof.

Exercise 4 (8 points)

Given k sorted arrays of n elements each, describe, preferably through pseudocode, an algorithm that delivers a single sorted array containing all and only the elements of the original k arrays. Evaluate the asymptotic complexity of your algorithm with respect to both n and k.

Tracce delle soluzioni

Esercizio 1
L'espressione regolare (1*01*01*01*01*)*, facilmente traducibile in un automa a stati finiti, definisce il linguaggio richiesto.

Esercizio 2
f(x,y) può essere calcolata mediante il seguente algoritmo:

1. Calcolo $f_y(y)$. Se il calcolo non termina (proprietà indecidibile) f(x,y) non è definita per alcun x.
2. Se il calcolo di $f_y(y)$ termina producendo il risultato z, inizio il calcolo di $f_z(x)$. Se il calcolo non termina f(x,y) non è definita per la coppia y, x. Se termina ottengo il risultato desiderato.

f(x,y) è quindi calcolabile anche se non totale.

Esercizio 3
Si ottiene una ricorrenza $T(n) = \Theta(1)$, per $n < 5$, altrimenti $T(n) = 2T(n-1) + \Theta(n^2)$. Il risultato presentato dopo il teorema dell'esperto ci fornisce: $T(n) = O(2^n n^2)$.

In aggiunta, è possibile dimostrare un limite asintotico più stretto, ossia $T(n) = O(2^n)$. Si può facilmente verificare che dimostrare $T(n) \le c\,2^n$ risulta complicato, a causa del termine $\Theta(n^2)$. Proviamo quindi a dimostrare che $T(n) \le c\,2^n - bn^2$, che ovviamente implica che $T(n) \le c\,2^n$.

La nostra ipotesi induttiva è dunque $T(n-1) \le c\,2^{n-1} - b(n-1)^2$. Sostituendola nella ricorrenza, otteniamo:

$$T(n) = 2T(n-1) + dn^2 \le 2\left(c\,2^{n-1} - b(n-1)^2\right) + dn^2 = c\,2^n - 2bn^2 - 2b + 4bn + dn^2 \le$$
$$\le c\,2^n - 2bn^2 + 4bn + dn^2 \le c\,2^n - 2bn^2 + \frac{b}{2}n^2 + dn^2 = c\,2^n - \frac{3}{2}bn^2 + dn^2 \le c\,2^n - bn^2$$

L'ultima maggiorazione è valida se $\frac{-3}{2}b + d \le -b \Leftrightarrow d \le \frac{b}{2} \Leftrightarrow b \ge 2d$

La penultima maggiorazione è valida per un valore di n sufficiente grande. Infatti:

$4bn \le \frac{b}{2}n^2 \Leftrightarrow 4 \le \frac{n}{2} \Leftrightarrow n \ge 8$

Di conseguenza, consideriamo come caso base n=8. Notiamo che per la validità della dimostrazione non è necessario imporre vincoli sulla costante c, quindi possiamo sceglierla grande a sufficienza per verificare la tesi per n=8, ovvero che $T(8) \le c\,2^8 - b\,8^2 = 256c - b\,8^2$

Si poteva anche direttamente notare, senza effettuare ulteriori maggiorazioni, che

$c\,2^n - 2bn^2 + 4bn + dn^2 \le c\,2^n - bn^2$

se e solo se

$-bn + 4b + dn \le 0$

e quest'ultima disuguaglianza è verificata se b>d e $n \ge \frac{4b}{b-d}$.

Esercizio 4

L’algoritmo più semplice è il *merge* dei k array. Si dovranno fare k confronti per ciascuno dei kn elementi, ottenendo una complessità temporale $\Theta(k^2n)$.

In alternativa, è possibile ottenere un algoritmo più complesso che ha però una complessità temporale minore, ovvero $O(nk \log(k))$.
L'idea dell'algoritmo è simile a selection sort, ossia trovare il minimo tra gli elementi che non sono ancora stati copiati nell'array ordinato e copiare questo minimo in coda all'array ordinato.
Possiamo ottenere una versione più efficiente del classico selection sort sfruttando il fatto che il nostro array è diviso in k array ordinati. Grazie a questo fatto, il minimo elemento da inserire in coda all'array ordinato da nk elementi può sempre essere ottenuto confrontando al più k elementi, che sono i minimi degli elementi che devono ancora essere copiati nell'array in output per ognuno dei k array. Ad esempio, nella prima iterazione, dobbiamo trovare il minimo tra tutti gli nk elementi: questo sarà necessariamente il minimo tra i primi elementi dei k array. Nella seconda iterazione, otteniamo l'elemento da copiare considerando gli stessi elementi dell'iterazione precedente, eccetto per l'elemento precedentemente copiato nell'array ordinato, che viene sostituito dal secondo elemento dell'array contenente l'elemento precedentemente copiato. Per ognuna delle nk iterazioni, dobbiamo calcolare il minimo di k elementi, ottenendo una complessità totale di $\Theta((nk)k) = \Theta(nk^2)$.
Tuttavia, possiamo migliorare la complessità asintotica utilizzando un BST bilanciato, ad esempio un red and black tree. Costruiamo questo albero con i primi elementi di ognuno dei k array. Un nodo dell'albero contiene come chiave l'elemento e un dato che identifica l'array di appartenenza dell'elemento tra i k array in input. Ogni iterazione cancella il minimo di questo albero, rimpiazzandolo con l'elemento successivo dell'array a cui apparteneva l'elemento cancellato dall'albero, informazione che può essere ricavata dal dato associato al nodo eliminato dall'albero. L'elemento cancellato dall'albero è inserito in coda all'array ordinato che si sta costruendo.
La costruzione dell’albero bilanciato ha un costo $O(k\log(k))$. Una singola iterazione dell'algoritmo ha costo $O(\log(k))$, siccome sia la cancellazione del minimo elemento che l'inserimento dell'elemento successivo hanno entrambe costo $O(\log(k))$ in un albero bilanciato. Dato che dobbiamo effettuare nk iterazioni, otteniamo una complessità totale di $O(nk \log(k))$.