Esercizio 1- 2

a. Si confronti $n^2 \lg n$ con $n \lg^2 n$:

è vero che $n \lg^2 n = \Omega(n^2 \lg n)$ oppure $n \lg^2 n = O(n^2 \lg n)$?

Risposta: $n \lg^2 n = O(n^2 \lg n)$ è vero perché $n \lg^2 n$ cresce meno in fretta di $n^2 \lg n$. Formalmente
esistono due costanti positive n_0 e c tali che $n \lg^2 n \leq c n^2 \lg n$ per ogni $n \geq n_0$. Infatti se $\lg n \geq 1$ questa disuguaglianza si riduce a $\lg n \leq c\, n$, che risulta vera per ogni $n \geq 2$, anche con $c = 1$.

Mentre $n \lg^2 n = \Omega(n^2 \lg n)$ è falso, infatti non esistono due costanti positive n_0 e c tali che $n \lg^2 n \geq c n^2 \lg n$, che per $n \geq 2$ si riduce a $\lg n \geq c\, n$, che è falsa. Infatti per ogni scelta di c troviamo un valore di n per cui è falsa. Per esempio si può prendere $n = 2^c$ e allora dovrebbe essere $c \geq c\, 2^c$ ma questo è falso per ogni scelta di $c \geq 1$.

b. Si risponda alla seguente domanda, motivando brevemente la risposta.
Se si dimostra che un algoritmo ha tempo di esecuzione $\Theta(n)$ nel caso peggiore, è possibile che in qualche caso l'algoritmo termini in $\Theta(1)$ passi?

Sì è possibile, si può pensar alla ricerca sequenziale che ha esattamente questo tempo di esecuzione: $\Theta(n)$ nel caso peggiore e $\Theta(1)$ in quello migliore.
D'altro canto non è sempre vero, per esempio il calcolo del massimo tra n elementi è sempre in $\Theta(n)$.

Esercizio 2- 2

2. Si risponda alle seguenti domande, motivando brevemente la risposta.

a. Si confronti nlg n con n:

è vero che n = Θ(n lg n) oppure n = O(nlgn) ?
n = O(nlgn) è vero perché n lg n è una funzione che cresce più in fretta di n, e infatti è vero che 1 ≤ c lg n per c = 1 e per valori di n≥2.

mentre non è vero che n è in Θ(n lg n), perché non è vero che n = Ω(n lg n), cioè che n ≥ c n lg n per un certa costante c e per valori di n a partire da un certo n_0. Se si prende n≥1 la disuguaglianza di riduce a lg n ≤ c, ma per ogni scelta di c troviamo un valore di n per cui è falsa, per esempio 2^{c+1}.

b. Se si dimostra che un algoritmo ha tempo di esecuzione Θ(n) nel caso peggiore, posso dedurne che nel caso migliore terminerà in O(n) passi?

Sì, il caso peggiore fornisce un limite superiore al tempo di esecuzione di ogni altro caso, compreso il migliore.

Esercizio 1 - 3

Si definisce punto di sella di una matrice quell'elemento che gode della proprietà di essere minimo di riga e massimo di colonna.
Scrivere un algoritmo che riceve in input una matrice A e dia in output gli indici di un punto di sella in A, (0,0) se non ce ne sono.
Si valuti asintoticamente il tempo di esecuzione dell'algoritmo proposto.

L'approccio più semplice consiste nel calcolare il minimo di ogni riga e il massimo di ogni colonna per poi vedere se c'è un elemento in comune.
Se n sono le righe e m le colonne, questo algoritmo ha tempo di esecuzione Θ(n*m), perché Θ(m) è il tempo necessario per il calcolo del minimo di una riga, per n righe ho Θ(n*m) e analogamente Θ(n) per il calcolo dei massimi di colonna, per ogni colonna ho Θ(n*m).
Inoltre servono due array di appoggio per i minimi e i massimi per una occupazione di spazio Θ(n+m).
Si può evitare di salvare i minimi o i massimi semplicemente confrontando ogni minimo trovato con gli elementi della sua colonna e fermando la ricerca in caso si sia trovato un punto di sella, altrimenti si passa alla riga successiva, se non ci sono duplicati.
Questo secondo approccio comporta un tempo di esecuzione Θ(n(m+n)), perché per ogni riga si calcola il minimo della riga, in Θ(m), e si confronta con gli elementi della colonna, in Θ(n). Lo spazio aggiuntivo però è costante.

l'algoritmo semplice

```
sella(A)
input: una matrice A
prec: gli elementi sono tutti distinti
output: le coordinate di un punto di sella, se presente, (0,0) altrimenti
sia n il numero delle righe di A
sia m il numero delle colonne di A
i = 1
 while i ≤ n do
       calcola il minimo della riga i-sima, sia A[i,k]                    Θ(m)
       confrontiamo A[i,k] con gli elementi della colonna k-sime
       j = 1
        while j ≤ n and A[i,k] ≥ A[j,k] do  j= j+1                        Θ(n)
        if j = n+1 then A[i,k] è il massimo della colonna k-sima
             return A[i,k] else i=i+1
return (0,0)
```

Il tempo di esecuzione nel caso peggiore è in Θ(n(n+m)), infatti il calcolo del minimo in una riga prende tempo Θ(m), il ciclo while interno ha tempo di esecuzione Θ(n), ma il ciclo esterno può essere eseguito n volte.

i duplicati

In presenza di duplicati, se la funzione che calcola il minimo di riga restituisce gli indici del primo che occorre, poi bisogna confrontare gli altri elementi della riga per vedere se ci sono altre occorrenze del minimo e se sì, controllare anche le colonne delle altre occorrenze.

Il tempo di esecuzione nel caso peggiore è in Θ(n(n+m)), infatti il calcolo del minimo in una riga prende tempo Θ(m), il controllo sulle colonne prende tempo di esecuzione Θ(n), se il numero dei duplicati è piccolo rispetto a n, infine il ciclo esterno può essere eseguito n volte.
Altrimenti si avrà un tempo $O(n^2(n+m))$

Analisi del problema

$A =$

	1	2	3	4	5	6	7	8	9
1		a_1	$\le b_3$			b_6			
2	b_1		$\le b_3$		a_2				
3	$\ge a_3$	$\ge a_3$	$a_3 = b_3$	$\ge a_3$	$b_5 \ge a_3$	$\ge a_3$	$\ge a_3$	$\ge a_3$	$b_9 \ge a_3$
4		b_2	$\le b_3$				a_4		
5			$\le b_3$	b_4				b_8	a_5
6	a_6		$\le b_3$				b_7		

Siano $a_1,\ldots,a_6$ i minimi sulle righe, e $b_1,\ldots b_9$ i massimi sulle colonne e supponiamo che ci sia un punto di sella nella riga 3, per cui $s = a_3 = b_3$. Valutiamo come devono essere gli altri minimi e massimi rispetto al punto di sella.

Osserviamo che ogni elemento nella riga 3 è maggiore o uguale al minimo $a_3 = s$, cioè $A[3,i] \ge s$, per $i=1,\ldots,9$, e quindi $b_1,\ldots, b_9 \ge s$, cioè s è minore o uguale ad ogni massimo di colonna.
Quindi $s=b_3$ è il minimo tra i massimi di colonna.
Analogamente gli elementi nella colonna 3 sono tutti minori di $b_3 = s$ e quindi tutti maggiori del minimo su ogni riga, quindi $s=a_3 \ge a_1,\ldots, a_9$, cioè $a_3 = s$ è il massimo tra i minimi di riga.
D'altra parte se $s = A[i,j]$ è il minimo tra massimi di colonna, ma anche il massimo tra i minimi di riga allora è un punto di sella. Quindi concludiamo che A[i,j] è un punto di sella se e solo se è il minimo tra massimi di colonna e il massimo tra i minimi di riga.

Ulteriore semplificazione

Quindi A[i,j] è un punto di sella se e solo se è il minimo tra massimi di colonna e il massimo tra i minimi di riga. Bisogna calcolare il minimo tra massimi di colonna e il massimo tra i minimi di riga e vedere se sono uguali.
Notiamo però che il massimo dei minimi di riga è un minimo di riga, quindi se è il massimo della sua colonna è un punto di sella.
Ma possiamo concludere che non ce sono in caso contrario?
No, se ci sono ripetizioni:

A_3 =

	1	2	3	4
1	8	16	8	10
2	9	20	6	24

A_4 =

	1	2	3	4
1	8	16	9	10
2	9	20	8	24

Se il massimo dei minimi di riga si ripete, potrebbe capitare che solo uno dei due sia un punto di sella, come in A_3, o nessuno come in A_4.
Se non ci sono ripetizioni invece se il massimo dei minimi di riga è il massimo della sua colonna allora è l'unico punto di sella.

Elementi distinti

$A =$

	1	2	3	4	5	6	7	8	9
1		a_1				b_6			
2	b_1				a_2				
3	$>a_3$	$>a_3$	$a_3 < b_3$	$>a_3$	b_5	$>a_3$	$>a_3$	$>a_3$	$>a_3$	b_9	$>a_3$
4		b_2					a_4		
5				b_4				$b_8 = a_5$	
6	a_6						b_7		

Sia a_3 il massimo dei minimi di riga e non sia una punto di sella. Vogliamo dimostrare che non è possibile che ci sia un altro punto di sella.
Supponiamo che ci sia e sia $a_5 = b_8$.
Se $a_5 = b_8$ allora $b_8 > a_3$ ma allora anche $a_5 > a_3$ contro l'ipotesi che a_3 fosse il massimo dei minimi di riga.

l'algoritmo

sella2(A)
input: una matrice A
prec: gli elementi sono tutti distinti
output: le coordinate di un punto di sella, se presente, (0,0) altrimenti
sia n il numero delle righe di A
sia m il numero delle colonne di A
metti (1,1) in (maxminR,maxminC)
for i = 1 to n do
calcola in [minR,minC] le coordinate del minimo della riga i-sima **Θ(n*m)**
if A[minR,minC] > A[maxminR,maxminC] then **Θ(m)**
maxminR = minR ; maxminC = minC
in (maxminR,maxminC) le coordinate del massimo tra i minimi
if A[maxminR,maxminC] è il massimo della colonna maxminC
then return (maxminR,maxminC)
else return (0,0) **Θ(n)**

Il tempo di esecuzione è in Θ(n*m), infatti
c'è un ciclo che ha tempo di esecuzione Θ(n*m) e l'altro Θ(n).

Se ci sono duplicati, si devono calcolare tutti i minimi di riga, determinare le coordinate del massimo tra questi e le sue ripetizioni e per ciascuna controllare se è il massimo nella sua colonna. Se nessuno lo è si può concludere che non c'è. Tempo in Θ(n*m), se il numero dei duplicati è piccolo rispetto a n, altrimenti tempo $O(n^2*m)$.

Esercizio 2 - 3

Scrivere un algoritmo che verifichi se esiste almeno una coppia di conoscenti coetanei, dati in input:

- **un vettore di interi C di dimensione *n*. Tali interi sono le età di *n* persone differenti;**
- **una matrice quadrata e simmetrica *A* di dimensione *n*x*n*, a valori *0* ed *1*. *A[i,j]* é uguale ad *1* se e solo se la persona *i* conosce la persona *j*. L'algoritmo dovrebbe dare in output una coppia con le coordinate dei due conoscenti coetanei se ce ne sono oppure la coppia (0,0). Valutare asintoticamente il tempo di esecuzione dell'algoritmo presentato.**

La prima soluzione che viene in mente prevede l'esame della matrice e il controllo sull'età in C per ogni coppia di indici (i,j) per cui A[i,j]=1. La complessità asintotica è $\Theta(n^2/2)$ nel caso peggiore e $\Theta(1)$ nel migliore.

Verso una soluzione alternativa

A =

	1	2	3	4	5	6
1		1				1
2	1				1	
3			1		1	
4		1				
5				1		
6	1					

C =

18	40	30	22	51	18
1	2	3	4	5	6

C' =

(18,1)	(40,2)	(30,3)	(22,4)	(51,5)	(18,6)
1	2	3	4	5	6

Si potrebbe partire da C e andare a vedere nella matrice se due coetanei si conoscono. Però la ricerca dei coetanei in C non ordinato dà un tempo quadratico. L'indice in C individua la persona, quindi è necessario non separarlo dall'età. Ma se ordinassimo C, potremmo individuare i duplicati e controllare nella matrice se si conoscono.
Creiamo allora un altro array di coppie il cui primo elemento è l'età e il secondo la persona di quell'età.
Ora l'ordinamento sulla prima chiave non ci fa perdere informazione.

La soluzione alternativa

A =

	1	2	3	4	5	6
1		1				1
2	1				1	
3			1		1	
4		1				
5				1		
6	1					

C’ =

(18,1)	(18,6)	(19,2)	(19,4)	(30,3)	(51,5)
1	2	3	4	5	6

Per ogni coppia di coetanei si deve verificare se si conoscono.

Quindi se il massimo numero di coetanei è piccolo rispetto a n, si può considerare costante questo tempo di verifica, altrimenti, se il numero dei coetanei è vicino a n, questo comporta un tempo quadratico in n (per esempio se sono tutti coetanei e non si conoscono).

Quale ordinamento

C' =

(18,1)	(18,6)	(19,2)	(19,4)	(30,3)	(51,5)
1	2	3	4	5	6

Usando l'heapsort si ha un tempo di esecuzione in Θ(n lg n) per l'ordinamento.

Se calcolassimo età minima, m, e massima, M, in C', potremmo considerare le età diminuite del minimo e quindi considerare le prime componenti di C' in [0,M-m] e applicare il countingsort, diciamo se le persone sono più di M.
L'ordinamento ora è lineare nel numero delle persone, n, e anche il calcolo del minimo e del massimo sono lineari in n.

Perché tutto l'algoritmo sia lineare, o in Θ(n lg n), bisogna che il numero massimo dei coetanei sia piccolo rispetto a n.