**Introduzione agli algoritmi**
**Prova di esame del 7/6/2016**
**Prof.sse E. Fachini - R. Petreschi**

# A

1. (Max punti 10) Dimostrare che, nel modello basato su confronti, il numero minimo di confronti che un algoritmo di ordinamento deve effettuare è $\Omega(n\log n)$ nel caso peggiore.

   **Sol.** si veda libro e lucidi

2. (Max punti 8) Si considerino le seguenti affermazioni:

   A. se $g(n) = O(f(n))$ e $f(n) = O(h(n))$ allora $g(n) = O(h(n))$. E' vera o falsa? Se ne dimostri la verità con una dimostrazione generale oppure se ne dimostri la falsità producendo tre funzioni che la contraddicono.

      **Sol.** L'affermazione è vera. Infatti, in base alla definizione di O, possiamo dire che esistono $c_1$ e $n_1$ tali che $0 \leq g(n) \leq c_1 f(n)$ per ogni $n \geq n_1$ ed esistono $c_2$ e $n_2$ tali che $0 \leq f(n) \leq c_2 h(n)$ per ogni $n \geq n_2$. Possiamo concludere che $0 \leq g(n) \leq c_1 c_2 h(n)$, per ogni $n \geq \max\{n_1, n_2\}$ e quindi che $g(n) = O(h(n))$.

   B. se si dimostra che un algoritmo ha tempo di esecuzione $\Theta(n)$ nel caso migliore, posso dedurne che nel caso peggiore terminerà in $O(n)$ passi?

      **Sol.** La risposta è no, il caso migliore fornisce un limite inferiore al tempo di esecuzione in ogni caso, non un limite superiore. Per
      esempio l'insertionSort ha tempo di esecuzione $\Theta(n)$ nel caso migliore ma nel caso peggiore è in $\Theta(n^2)$.

3. (Max punti 12) Si supponga che un maxheap H di altezza h sia stato modificato in modo tale che gli elementi di un livello i, con $0 < i < h$, siano stati sostituiti con elementi più piccoli di quelli originali del maxheap. Si definisca un'algo-

**Introduzione agli algoritmi**
**Prova di esame del 7/6/2016**
**Prof.sse E. Fachini - R. Petreschi**

# A

ritmo Ripristina(H,i) che ristabilisce la proprietà di essere Max-Heap, eventualmente violata dalla modifica sopra specificata. Si valuti il tempo di esecuzione asintotico dell'algoritmo presentato.

**Sol.** La funzione ripristina deve solo chiamare la Maxheapify su tutti i nodi del livello, perché la proprietà di essere maxheap è violata nel nodo del livello i al più verso i figli e non verso il padre visto che gli elementi che hanno sostituito gli originali sono minori di questi ultimi. La Maxheapify, che riceve in input un array e un indice di un elemento che in cui la condizione di essere maxheap può essere violata rispetto ai figli, ripristina la proprietà su quell'indice, sotto l'ipotesi, la precondizione, che i figli del nodo siano radici di maxheap. Nel nostro caso dunque è applicabile perché i nodi del livello i sono gli unici ad aver subito un cambiamento. Poiché i nodi del livello i hanno indice da $2^i$ a $2^{i+1}$ -1, poiché 0 < i < h, il tempo di esecuzione dipende da i per il numero di chiamate della maxheapify e dal tempo di esecuzione di ciascuna chiamata. Poiché l'altezza del sotto albero radicato in un nodo di livello i è h - i, supponendo che l'albero sia completo, in generale potremmo dire che il tempo di esecuzione è in $O(2^i *(h-i))$. Volendo esprimerlo in termini del numero degli elementi nel maxheap, n, potremmo dire che nel caso peggiore i è il penultimo livello e quindi si eseguono chiamate su alberi di altezza 1, concludendo che nel caso peggiore si ha un tempo di esecuzione O(n), mentre nel caso migliore i =1 e il tempo di esecuzione è in O(lg n), perché in questo caso le due chiamate sono su sotto alberi di altezza al più h-1.