

*senza nome*

1) Siano  $f, g, h$  arbitrarie funzioni asintoticamente crescenti e positive. Dimostrare la verità o falsità della seguente affermazione:  
se  $f(n) = \Theta(g(n))$ , allora  $h(f(n)) = \Theta(h(g(n)))$   
nota: si ricordi che  $h(f(n))$  denota la composizione delle funzioni nn la moltiplicazione di  $f$  e  $g$

*compiuto 1*

2) Sia data la seguente equazione di ricorrenza:  
 $T(n) = 1$  se  $n=1$ ,  $T(n) = 2T(1/2*n) + n(\log_2(n^2))$  se  $n > 1$   
Calcolare la stima asintotica più vicina possibile a  $T(n)$

3) sia dato un generico albero binario  $T$ . Si definisca un algoritmo ricorsivo che costruisca un nuovo albero  $T'$  strutturalmente identico a  $T$ , tale, cioè, che ogni nodo di  $T$  abbia una controparte situata nella stessa posizione in  $T'$ . Inoltre, ciascun nodo  $u$  di  $T'$  deve contenere come chiave il numero di nodi pari contenuti nel sottoalbero di  $T$  radicato nel suo nodo controparte  $u$  in  $T$ .

*esercizio 2*  
~~esercizio 2~~

purtroppo non ho lo scanner...mo te la scrivo qui  
esercizio 1 [5 punti]

Siano  $f$  e  $g$  due arbitrarie funzioni asintoticamente crescenti e positive.  
Si dimostri la verità o falsità della seguente affermazione:  
se  $\log(\log f(n)) = \Theta(\log(\log g(n)))$  allora  $\log f(n) = \Theta(\log g(n))$   
ps il log è in base due

esercizio 2 [7 punti]

$T(n) =$   
1 se  $n=1$   
 $\sqrt{n} * T(\sqrt{n}) + n$  se  $n > 1$

esercizio 3 [7 punti]

Si scriva un algoritmo ricorsivo efficiente che, dato un albero binario  $T$ , verifichi (in una singola visita dell'albero) se per ogni nodo dell'albero i suoi sottoalberi sinistro e destro hanno lo stesso numero di nodi con chiave pari.

Non è ammesso l'uso di variabili globali né di parametri per riferimento

esercizio 4 [11 punti]

Un percorso semplice (quindi non ciclico) in un grafo orientato  $G$  si dice [A] massimale [B] se non vi si può aggiungere "alla fine" nessun altro nodo senza renderlo un percorso ciclico o fargli perdere la proprietà di essere un percorso.

Si scriva un algoritmo efficiente che, dato un grafo orientato  $G$  e un nodo  $s$  di  $G$ , stampi tutti i percorsi massimali di  $G$  che si dipartono da  $s$ .

[B] Suggerimento: [B] è possibile risolvere il problema tramite un'opportuna variante della visita in profondità e l'impiego di una coda o stack esplicito

# Tema d'esame di Algoritmi e Strutture Dati

## Modulo A

### 22/01/2001

**Tempo a disposizione: 3 ore.**

1. Ordinare in modo crescente secondo il tasso di crescita asintotico le seguenti funzioni, esplicitando e dimostrando per esteso le relazioni asintotiche ( $o(\cdot)$ ,  $\omega(\cdot)$ , oppure  $\Theta(\cdot)$ ) esistenti tra le funzioni adiacenti nell'ordinamento risultante:

$$\begin{array}{ll} n^{1/a} & \log(\log n^a) \\ 2^{\log(2e^{\ln n})} & n^a \\ \log(\log n) & n + 2 \end{array}$$

**Nota:**  $a$  è da considerarsi una costante arbitraria che soddisfa come **unico vincolo** quello di essere **maggiori di 0**.

2. Sia data la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 5 & \text{se } n = 2 \\ 2\sqrt{n} T(\sqrt{n}) + n & \text{se } n > 2 \end{cases}$$

- (a) Trovare la stima asintotica più vicina possibile a  $T(n)$ , utilizzando il **metodo iterativo**.
- (b) Risolvere l'esercizio utilizzando il **metodo di sostituzione**, utilizzando come ipotesi di soluzione i risultati calcolati per il punto (a)

3. Considerate l'algoritmo di ordinamento Quicksort.

- (a) Illustrare dettagliatamente i passi eseguiti dall'algoritmo sulla seguente sequenza di numeri interi in input:

$$\langle 95, 90, 76, 45, 20, 25, 34, 38 \rangle$$

- (b) Mostrare una permutazione degli 8 numeri della sequenza al punto (a) che costituisce il **caso migliore** per tale algoritmo di ordinamento per sequenze di 8 elementi, e motivare la scelta della sequenza.

4. Sia dato un albero binario di ricerca  $T$  qualsiasi con  $N$  nodi. Ciascun nodo  $n$  contiene un campo `colore[n]` che può assumere valori `rosso` o `nero`. Si assuma che  $T$  sia stato colorato assegnando a ciascun nodo  $n$  un valore per il campo `colore[n]`. Scrivere un algoritmo che decide se  $T$  è un albero Red–Black oppure no. In caso affermativo, l'algoritmo deve ritornare l'altezza nera; in caso negativo, deve ritornare il primo sottoalbero che viola la condizione e segnalare il motivo per cui l'albero non è un albero Red–Black.

# Tema d'esame di Algoritmi e Strutture Dati

## Modulo A

### 01/03/2001

**Tempo a disposizione: 2 ore e 30 minuti.**

1. Date le seguenti coppie di funzioni, dimostrare per esteso le relazioni asintotiche più restrittive possibili esistenti tra le funzioni di ciascuna coppia:

(i)	$(3/2)^n$	$(7/5)^n$
(ii)	$\frac{n^2}{\sqrt{\log n}}$	$n\sqrt{\log n}$
(iii)	$n \log^2 n$	$\log(4^n) \log(n^4)$
(iv)	$n^{(\log n)}$	$2^{\log^4 n}$

2. Sia data la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} k & \text{se } n = 1 \\ 2 T(\sqrt{n}) + \log n & \text{se } n > 1 \end{cases}$$

dove  $k$  è una costante positiva. Trovare la stima asintotica più vicina possibile di  $T(n)$ , utilizzando il **metodo iterativo**. È ammesso l'uso di un albero di ricorrenza.

3. Sia dato un array di  $n$  elementi le cui chiavi possono assumere solo valore `vero`, `forse` o `falso`. Scrivere un algoritmo che in tempo  $O(n)$  riarrangi gli elementi dell'array in modo che tutte le chiavi con valore `falso` precedano tutte le chiavi con valore `forse`, e queste ultime precedano tutte le chiavi con valore `vero`.
4. Partendo dalla definizione di **albero AVL minimo**, dimostrare la seguente affermazione per induzione o argomentarene la falsità, esibendo un controesempio:

in un **albero AVL minimo** di altezza  $h$ , l'**altezza minima** di un nodo foglia è esattamente  $\lceil h/2 \rceil$ .

# Tema d'esame di Algoritmi e Strutture Dati

## Modulo B

### 18/06/2001

**Tempo a disposizione: 3 ore.**

Considerate il problema di memorizzare una collezione di elementi tramite un albero di ricerca binario in modo da minimizzare il numero di confronti necessari per trovare un dato elemento. Se tutti gli elementi avessero uguale probabilità di essere ricercati, allora la soluzione ottima per memorizzare gli elementi sarebbe tramite un albero perfettamente bilanciato. Se però alcuni elementi vengono ricercati con maggiore frequenza di altri, un albero non bilanciato può essere preferibile.

Sopponete di avere  $N$  elementi distinti con chiavi  $k_1 < k_2 < \dots < k_N$ , e che l'elemento  $i$ -esimo abbia probabilità  $p_i \geq 0$  di essere ricercato. Supponete, inoltre, che ogni ricerca acceda effettivamente ad un elemento nell'albero, in altre parole che valga  $\sum_{i=1}^N p_i = 1$ . Se l'elemento  $i$ -esimo è memorizzato a livello  $l_i$  dell'albero (la radice è a livello 1, i suoi figli a livello 2, ecc.), allora per trovare l'elemento  $i$  saranno necessari  $l_i$  confronti.

Quindi, per un dato albero, il numero medio di confronti necessari per effettuare una ricerca è  $\sum_{i=1}^N p_i l_i$ . L'obiettivo è determinare l'albero binario che minimizza tale quantità.

1. Considerate l'algoritmo “greedy” che inserisce le chiavi in ordine decrescente di frequenza. Fornire un esempio minimo (con il più piccolo numero di elementi) che mostri che questa strategia greedy non sempre fornisce l'albero ottimo di ricerca.
2. Fornire un algoritmo che risolve correttamente il problema, e se ne analizzi la complessità.

# Tema d'esame di Algoritmi e Strutture Dati

## Modulo B

### 18/06/2001

**Tempo a disposizione: 2 ore e 30 minuti.**

Considerare il problema di disporre  $N$  libri su degli scaffali in una libreria.

Sia  $b_i$  (con  $1 \leq i \leq N$ ) il libro  $i$ -esimo nell'ordinamento, e siano  $s_i$  la misura del suo spessore e  $a_i$  la misura della sua altezza. Ogni scaffale ha la stessa lunghezza  $L$ .

**L'ordine in cui i libri devono essere disposti è fissato secondo un sistema di catalogazione (ad esempio in ordine alfabetico per autore, e se dello stesso autore per ordine alfabetico del titolo) e non può essere cambiato.**

Le altezze dei libri possono essere differenti da libro a libro, e le distanze in cui fissare gli scaffali nella libreria possono essere aggiustati secondo l'altezza del libro più alto che verrà riposto sullo scaffale stesso.

Il **costo** di una particolare disposizione dei libri sugli scaffali sia definita come la somma su tutti gli scaffali delle altezze del libro più alto su ciascuno scaffale.

Il problema è quello di trovare la disposizione dei libri sugli scaffali in modo da minimizzare il costo.

1. Fornire un esempio che mostri che l'algoritmo “greedy” per disporre i libri sugli scaffali, utilizzando la strategia di riempire il più possibile gli scaffali, non sempre fornisce la soluzione ottima.
2. Fornire un algoritmo che risolve correttamente il problema e ne si analizzi la complessità.

# Tema d'esame di Algoritmi e Strutture Dati

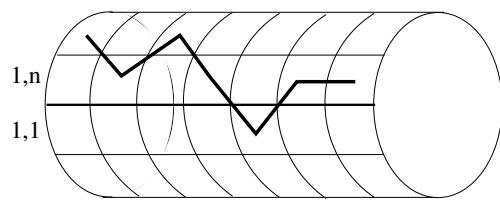
## Modulo B

### 21/11/2001

**Tempo a disposizione: 2 ore e 30 minuti.**

Considerare una matrice  $A : n \times n$  di interi positivi ( $a_{i,j}$ ), con  $1 \leq i, j \leq n$ . Si supponga di unire i due bordi (o estremi) orizzontali della matrice in modo da formare un cilindro, cioè in modo che la prima e l'ultima riga siano tra loro contigue (vedi figura).

j \ i	1	2	3	4	...
1					
2					
3					
4					
:					
n					



Si consideri ora il problema di tracciare un *percorso* a partire da una cella della prima colonna fino ad una cella nell'ultima colonna, sotto le seguenti restrizioni: da ogni cella  $(i, j)$ , è possibile muoversi solo nelle celle  $(i + 1, j)$ ,  $(i + 1, j - 1)$  o  $(i + 1, j + 1)$ . Il percorso può partire da una qualsiasi cella della prima colonna, e terminare in una qualsiasi cella dell'ultima colonna (vedi figura). Il *costo* di un percorso è costituito dalla somma dei valori delle celle attraversate dal percorso stesso. Il problema è di calcolare il percorso di peso minimo nella matrice  $A$  in input.

1. Esibire la *sottostruttura ottima* per il problema in questione e dimostrarne l'ottimalità.
2. Esibire l'*equazione di ricorrenza* che caratterizza il costo della soluzione ottima.
3. Fornire un algoritmo per calcolare la *soluzione ottima* (e non solamente per il costo della soluzione ottima) e studiarne la complessità.

# Tema d'esame di Algoritmi e Strutture Dati

## Modulo A

### 28/02/2002

**Tempo a disposizione: 3 ore.**

1. Studiare la relazione asintotica tra le seguenti funzioni, esplicitando e dimostrando per esteso la relazione asintotica ( $o(\cdot)$ ,  $\omega(\cdot)$ , oppure  $\Theta(\cdot)$ ) esistente tra di esse:

$$n^a \quad \log |\log n^a|$$

**Nota:**  $a$  è da considerarsi una **costante arbitraria**, mentre  $|f(n)|$  indica il valore assoluto della funzione  $f(n)$ .

2. Sia data la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} k & \text{se } n = 2 \\ 4T(n/4) + \sqrt{n} & \text{se } n > 2 \end{cases}$$

Trovare la stima asintotica più vicina possibile a  $T(n)$ , utilizzando il **metodo iterativo** (è ammesso l'impiego degli alberi di ricorrenza).

3. Dimostrare per induzione la validità della seguente uguaglianza:

$$\sum_{i=1}^{n-1} \frac{1}{i(i+1)} = 1 - \frac{1}{n}$$

4. Sia dato un albero binario di ricerca  $T$  nei cui nodi **non** sia presente il puntatore al padre. Si descriva prima e si sviluppi poi un algoritmo **ricorsivo** che realizzi la cancellazione di un nodo dell'albero (se esiste). L'algoritmo dovrà ricevere in input:

- un puntatore  $T$  alla radice del (sotto)-albero su cui eseguire l'operazione di cancellazione;
- il valore della chiave  $k$  (e **non** il nodo) da eliminare (se presente).

**Nota:** ogni ulteriore algoritmo di supporto sviluppato dovrà essere di tipo ricorsivo.

# Tema d'esame di Algoritmi e Strutture Dati Mod. B

## 31/01/2003

Tempo a disposizione: **3 ore**

Il conduttore di un programma radiofonico deve ogni giorno compilare il palinsesto del suo programma musicale, pevedendo la programmazione di alcune delle canzoni dalla lista  $I$  di  $m$  canzoni della stazione, indicizzate con i numeri da 1 ad  $m$ , in modo da massimizzare il gradimento dei suoi ascoltatori. La radio per cui il conduttore lavora lancia ogni giorno un sondaggio col quale gli ascoltatori possono votare le canzoni da loro preferite. Il conduttore ha quindi ogni giorno a disposizione i risultati del sondaggio di gradimento del giorno prima, in forma di valori da 0 ad 1 ( $g_i$  indica il gradimento della canzone  $i$ ). Il programma radiofonico ha la durata di  $N$  minuti e ciascuna canzone ha la sua durata ( $d_i$  è la durata in secondi della canzone  $i$ ).

Definire un algoritmo che, dati in ingresso la durata  $N$  in minuti del programma, la lista  $I$  delle canzoni dalla prima alla  $m$ -esima, la lista associata  $D[1..m]$  delle durate in secondi delle canzoni e la lista  $G[1..m]$  dei valori di gradimento del giorno prima, fornisca in uscita il palinsesto per il programma che massimizza il gradimento del pubblico (nota che il palinsesto non può contenere più occorrenze della stessa canzone).

# Tema d'esame di Algoritmi e Strutture Dati

## Modulo B

### 024/02/2003

**Tempo a disposizione: 3 ore.**

Un turista in viaggio deve visitare una serie di località di interesse artistico sitate nei pressi della città  $X$ . Prima di partire per il viaggio ha già definito l'ordine di visita delle località (dato dalla sequenza  $l[1], \dots, l[n]$ ), una per ogni giornata di viaggio. Una volta giunto nella città  $X$ , il turista deve iniziare il suo viaggio tra le località. Viene informato che in ogni località  $i$  (con  $1 \leq i \leq n$ ) di suo interesse sono a disposizione differenti mezzi di trasporto tra una località e la successiva (l'insieme  $M[i]$  elenca i mezzi di trasporto che il turista ha a disposizione nella località  $i$  per raggiungere la località  $i + 1$ , mentre  $M[0]$  sono i mezzi a disposizione per raggiungere la località 1 dalla città  $X$  di partenza). Ogni mezzo di trasporto disponibile in ciascuna località ha un suo costo di utilizzo (sia  $price[i, j]$  il costo per l'utilizzo del mezzo di trasporto  $j$  nella località  $i$ ). Inoltre, il turista ha tra le sue preferenze di viaggio il non voler mai utilizzare lo stesso mezzo di trasporto per due giorni di seguito (anche se è disposto a riutilizzare uno stesso mezzo di trasporto in giornate non consecutive).

Si definisca un algoritmo di Programmazione Dinamica che risolva il problema di pianificare quali mezzi di trasporto convenga utilizzare al turista per spostarsi tra le varie località, in modo da ridurre al minimo il costo complessivo del viaggio, ma anche da rispettare le preferenze di viaggio del turista.

# Tema d'esame di Algoritmi e Strutture Dati

## Modulo B

### 27/06/2003

**Tempo a disposizione: 3 ore.**

Sia dato un **grafo orientato aciclico**  $G = \langle V, E \rangle$  rappresentato tramite liste di adiacenza. Ad ogni arco  $(i, j)$  sia associato in costo  $\text{peso}(i, j)$  come valore intero. Il problema è quello di calcolare il percorso di peso minimo tra un vertice sorgente  $s_1$  ed un vertice destinazione  $s_z$ . Il costo di un percorso sia definito come la somma dei pesi assicati agli archi che lo costituiscono. Definire un algoritmo per risolvere il problema dato. Più specificatamente:

1. scomporre il problema in sottoproblemi e dimostrare la proprietà di sottostruttura ottima;
2. esibire l'equazione di ricorrenza per il calcolo del costo del percorso ottimo;
3. fornire un algoritmo di programmazione dinamica che calcoli il costo della soluzione ottima, e se ne analizzi la complessità;  
[**Suggerimento:** al fine di poter riempire correttamente la struttura tabellare che rappresenta i (costi dei) sottoproblemi, può risultare utile calcolare un qualche ordinamento tra i vertici, utilizzando, ad esempio, una variante opportuna di uno degli algoritmi su grafi visti a lezione.]
4. infine, fornire un algoritmo che stampi il percorso ottimo tra  $s_1$  e  $s_z$ .

```
algo(T, P, T')
  if (T != NIL)
    algo(sx[T], T, T')
    algo(dx[T], T, T')
    if (cerca(T', key[T]) != NIL)
      cancella(T, P)
```

```
cerca(T, k)
  if (T != NIL)
    if (k < key[T])
      return cerca(sx[T])
    else if (k > key[T])
      return cerca(dx[T])
  return T
```

# Tema d'esame di Algoritmi e Strutture Dati

## Modulo A

### 19/06/2006



**Tempo a disposizione: 3 ore.**

1. [5 punti] Si dimostri la verità o la falsità della seguente affermazione:

$$\text{se } 2^{f(n)} = \Theta(2^{g(n)}), \text{ allora } f(n) = \Theta(g(n))$$

2. [5 punti] Sia data la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 3T(n/4) + \sqrt{n} & \text{se } n > 1 \end{cases}$$

Risolvere l'equazione di ricorrenza utilizzando il metodo iterativo (alberi di ricorrenza).

3. [10 punti] Si scriva un *algoritmo ricorsivo* efficiente che cancelli da un albero binario di ricerca  $T$  tutte le chiavi contenute in un altro albero binario di ricerca  $T'$ .
4. [10 punti] Si scriva un algoritmo dettagliato che, dato un grafo orientato  $G$ , ne calcoli le *componenti fortemente connesse* e stampi i vertici contenuti in ciascuna componente e il numero delle componenti del grafo.

# Tema d'esame di Algoritmi e Strutture Dati I

## 11/09/2006

**Tempo a disposizione: 3 ore.**

1. **(4 punti)** Dimostrare o falsificare la seguente affermazione:

$$\text{Se } f(n) = O(g(n)) \text{ allora } \sqrt{g(n)} = \Omega(\sqrt{f(n)})$$

2. **(6 punti)** Risolvere la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ 4T(n/9) + \sqrt{n} & \text{se } n > 1 \end{cases}$$

3. **(10 punti)** Si definisca un algoritmo ricorsivo efficiente che, ricevuti in ingresso un (riferimento ad un) Albero Binario di Ricerca  $T$  e tre valori  $k_{min}, k_{max}$  (con  $k_{min} \leq k_{max}$ ) e  $z \geq 0$  cancelli dall'albero  $T$  tutti i nodi che o hanno chiave con valore **esterno all'intervallo**  $[k_{min}, k_{max}]$  o stanno a **distanza maggiore uguale a**  $z$  dalla radice. Non è ammesso l'uso di variabili globali né del passaggio di parametri per riferimento.

4. **(10 punti)** Scrivere un algoritmo che, in tempo lineare sulla dimensione del grafo in input, ne calcoli un ordinamento topologico. **Non è ammesso l'impiego della visita in profondità (DFS).**

# Tema d'esame di Algoritmi e Strutture Dati I

## 25/06/2007

**Tempo a disposizione: 3 ore.**

1. [6 punti] Si supponga che  $h(n) = \Theta(t(n))$  e che  $f(n)/h(n) = \Theta(g(n))$ . Si dimostri la verità o la falsità della seguente affermazione:

$$\frac{f^2(n)}{h(n) t(n)} = \Theta(g^2(n))$$

2. [7 punti] Sia data la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 15 T(n/4) + n^2 \log n & \text{se } n > 1 \end{cases}$$

Trovare la stima asintotica più vicina possibile a  $T(n)$ .

3. [11 punti] Scrivere un algoritmo ricorsivo efficiente che, dato un albero binario di ricerca  $T$  e un due valori  $k_1$  e  $k_2$  (con  $k_1 \leq k_2$ ), cancelli da  $T$  tutti nodi con chiavi comprese tra  $k_1$  e  $k_2$  e restituisca il numero di nodi cancellati dall'albero.

**Non è ammesso l'uso di passaggio di parametri per riferimento né l'impiego di variabili globali.**

4. [6 punti] Un *tour di Eulero* in un grafo orientato  $G = \langle V, E \rangle$  è un percorso ciclico che attraversa ogni arco una e una sola volta. Un tour di Eulero in un grafo orientato  $G$  esiste se il grado entrante di ogni vertice è uguale al suo grado uscente. Si scriva un algoritmo che, dato un grafo orientato  $G$ , verifichi se esista o meno un tour di Eulero in tempo  $\Theta(|V| + |E|)$ .

# Tema d'esame di Algoritmi e Strutture Dati I

## 16/07/2007

**Tempo a disposizione: 3 ore.**

1. [7 punti] Usando la definizione di  $\Theta$ , si domostri la verità o la falsità della seguente affermazione:

se  $f(n) = \Theta(n)$  e  $g(n) = \Theta(2^{n^2})$ , allora  $2^{2 \cdot \log f(n)} = \Theta(\log(g(n)))$

2. [7 punti] Sia data la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 2 \cdot T(n/2) + T(n/4) + n & \text{se } n > 1 \end{cases}$$

Trovare la stima asintotica più vicina possibile a  $T(n)$ .

3. [10 punti] Scrivere un algoritmo ricorsivo efficiente che, dato un albero binario di ricerca  $T$ , due valori  $k_1$  e  $k_2$  (con  $k_1 \leq k_2$ ) e un valore  $c$ , restituisca (se esiste), effettuando una sola visita dell'albero, il puntatore al nodo di  $T$  che ha chiave compresa tra  $k_1$  e  $k_2$  e al tempo stesso che sia la più vicina possibile al (ma diversa dal) valore  $c$ .

**Non è ammesso l'uso di passaggio di parametri per riferimento né l'impiego di variabili globali.**

4. [6 punti] Si scriva un algoritmo che, dato un grafo orientato  $G$  e un vertice  $v$  di  $G$ , verifichi in tempo lineare sulla dimensione del grafo se  $G$  è un albero radicato in  $v$ .

**Non è ammesso l'uso di passaggio di parametri per riferimento né l'impiego di variabili globali.**

# Tema d'esame di Algoritmi e Strutture Dati I

## (12/02/2008)

**Tempo a disposizione: 3 ore.**

1. [7 punti] Siano  $f(n)$  e  $g(n)$  due funzioni asintoticamente positive e crescenti. Dimostrare la verità o la falsità della seguente affermazione:

$$\log(f(n) \cdot g(n)) = O(\max\{\log f(n), \log g(n)\})$$

2. [6 punti] Si consideri la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 17T(n/2) + n^4 & \text{se } n > 1 \end{cases}$$

Trovare la stima più vicina possibile a  $T(n)$ .

3. [10 punti] Si consideri un albero binario  $T$  che soddisfa la seguente proprietà: *il valore della chiave di ogni nodo è non minore del valore delle chiavi dei suoi due figli*.

Si definisca un algoritmo ricorsivo che dato (il riferimento a) l'albero  $T$  e due valori di chiave  $k_{min}$  e  $k_{max}$  (con  $k_{min} < k_{max}$ ), cancelli dall'albero  $T$  tutti i nodi con chiave compresa tra  $k_{min}$  e  $k_{max}$ , preservando la proprietà sopra riportata.

**Suggerimento:** Può risultare utile sviluppare un algoritmo ricorsivo di appoggio che esegua la cancellazione della radice di un (sotto) albero del tipo descritto sopra e ritorni l'albero risultante.

4. [7 punti] Si definisca un algoritmo che, dato in ingresso un grafo  $G$  arbitrario e un vertice qualsiasi  $v$ , trasformi il grafo in ingresso nell'albero radicato in  $v$  dei percorsi minimi che si dipartono da  $v$ .

# Tema d'esame di Algoritmi e Strutture Dati I

## 27/03/2008

Tempo a disposizione: 3 ore.

1. (7 punti) Siano  $f$  e  $g$  due funzioni positive e asintoticamente crescenti. A seconda del caso, dimostrare la verità o la falsità delle seguenti affermazioni:

(a) se  $h^2(n) = \Theta(\min\{f(n), g(n)\})$ , allora  $\sqrt{f(n)} = O(h(n))$ .

(b) se  $\sqrt{h(n)} = O(\min\{f(n), g(n)\})$ , allora  $h(n) = O(g^2(n))$ .

2. (6 punti) Risolvere la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ T(3n/4) + T(n/2) + n & \text{altrimenti} \end{cases}$$

3. (9 punti) Si definisca un algoritmo ricorsivo efficiente che, ricevuti in ingresso un (riferimento ad un) Albero Binario di Ricerca  $T$  e tre valori  $k_{\min}, k_{\max}$  (con  $k_{\min} \leq k_{\max}$ ) e  $k$ , restituisca, se esiste, quel nodo di  $T$  che ha chiave con valore interno all'intervallo  $[k_{\min}, k_{\max}]$  e che, contemporaneamente, sia il più lontano possibile da  $k$ . Non è ammesso l'uso di variabili globali né del passaggio di parametri per riferimento.

4. (8 punti) Scrivere un algoritmo che dato in ingresso un vertice  $s$  e un grafo orientato  $G$ , rappresentato con liste di adiacenza, stampi tutti percorsi ciclici di  $G$  che si dipartono da  $s$ .

# Tema d'esame di Algoritmi e Strutture Dati I

## 20/06/2008

Tempo a disposizione: 3 ore.

1. [6 punti] Si domostri la verità o la falsità della seguente affermazione:

se  $2^{f(n)} = \Theta(g(n))$  e  $g(n) = \Theta(h(n)^k)$  per una qualche costante  $k > 0$ , allora  
 $f(n) = \Theta(\log h(n))$

2. [5 punti] Sia data la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 4 T(n/4) + \sqrt{n} & \text{se } n > 2 \end{cases}$$

Trovare la stima asintotica più vicina possibile a  $T(n)$ .

3. [10 punti] Sia dato un albero binario di ricerca  $T$  in cui ogni nodo contiene esclusivamente un campo per la chiave, uno per il puntatore al figlio destro e uno per il figlio sinistro. Siano inoltre dati in ingresso un possibile valore di chiave  $k$  e due valori  $l_1$  e  $l_2$  (con  $l_1 \leq l_2$ ). Scrivere un algoritmo ricorsivo efficiente che cerchi e stacchi, se esiste, quel "nodo dell'albero  $T$  che, tra i nodi che si trovano ad un livello di profondità esterno all'intervallo  $[l_1, l_2]$  e che sono diversi dalla radice di  $T$ , contiene la chiave più vicina possibile a  $k$  ma maggiore di  $k$ ".

L'algoritmo dovrà ritornare il riferimento al nodo cercato, se esso esiste. Non è ammesso l'uso di passaggio di parametri per riferimento né l'impiego di variabili globali.

4. [9 punti] Sia dato un grafo orientato  $G = \langle V, E \rangle$ , rappresentato tramite liste di adiacenza. Si definisca un algoritmo che verifichi in tempo lineare sulla dimensione del grafo se è vero che esiste un vertice  $v \in V$  tale che:

- se  $u \in V$  allora  $u \rightsquigarrow v$  per qualche percorso  $\pi$ ;
- per ogni  $z \in V$ ,  $v \rightsquigarrow z$  per qualche percorso  $\lambda$ ;

# Tema d'esame di Algoritmi e Strutture Dati I

(12/01/2009)

Tempo a disposizione: 3 ore.

V

- [6 punti] Assumendo che  $f$  e  $g$  siano funzioni asintoticamente positive e crescenti, si dimostri la verità o la falsità della seguente affermazione:

$$\text{se } f(n) = \Theta(n) \text{ e } g(n) = \Theta(2^n), \text{ allora } 2^{f(n)} = \Theta(g(n))$$

✓

- [6 punti] Risolvere la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ 6 T(n/8) + \sqrt{n} & \text{se } n > 1 \end{cases}$$

- [10 punti] Scrivere un algoritmo ricorsivo efficiente che, dato un albero binario di ricerca  $T$ , un possibile valore di chiave  $k$ , e due interi  $l_1$  e  $l_2$  (con  $1 \leq l_1 \leq l_2$ ), cerchi un nodo dell'albero che soddisfa la seguente proprietà:

"contiene la più grande chiave pari minore di  $k$  tra i nodi che si trovano a profondità compresa tra  $l_1$  e  $l_2$ "

Se il nodo cercato esiste, l'algoritmo dovrà staccare dall'albero  $T$  il nodo trovato e rimuovere il riferimento al nodo stesso. Non è ammesso l'uso di passaggio di parametri per riferimento né l'impiego di variabili globali.

- [8 punti] Si definisca un algoritmo che, dato in ingresso un grafo  $G$  arbitrario e un vertice qualsiasi  $v$ , trasformi il grafo in ingresso nell'albero radicato in  $v$  dei percorsi minimi che si dipartono da  $v$ . L'algoritmo deve risolvere il problema in tempo lineare sulla dimensione del grafo.

C

# Tema d'esame di Algoritmi e Strutture Dati I

15/07/2009

Tempo a disposizione: 3 ore.

1. Siano  $f$  e  $g$  due funzioni asintoticamente positive e crescenti. A seconda del caso, dimostrare la verità o la falsità delle seguenti affermazioni:

(i) se  $\sqrt{h(n)} = O(2^{f(n)^{p(n)}})$ , allora  $\log \log h(n) = \Theta(g(n) \cdot \log f(n))$ . (1)

(ii) se  $h(n) = \Theta(\max\{\log \log f(n), \log \log g(n)\})$ , allora  $g(n) = O(2^{(2^{h(n)})})$ .

2. Risolvete la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 1 & \text{se } n \leq 2 \\ T(n/3) + T(n/4) + n & \text{altrimenti} \end{cases}$$

✓

3. Si definisca un algoritmo ricorsivo efficiente che, noti un Albero Binario di Ricerca  $T$ , i valori  $k_{min}, k_{max}$  (con  $k_{min} \leq k_{max}$ ) e un intero  $x \geq 0$ , cancelli tutti i nodi con chiave pari compresa tra  $k_{min}$  e  $k_{max}$  che sono radici di sottoalberi di altezza non inferiore ad  $x$ . Notare che il rimbalzo sulle altezze dei sottosalberi va misurato rispetto all'albero originario, non ha quello eventualmente modificato. Non è ammesso l'uso di variabili globali, né del passaggio di parametri per riferimento.

4. Scrivere un algoritmo che dato in ingresso un vertice  $s$  e un grafo orientato  $G$ , rappresentato con liste di adiacenza, stampi tutti percorsi di  $G$  che raggiungono il vertice  $s$ .



✓

# Tema d'esame di Algoritmi e Strutture Dati I

## (Gruppo I)

22/01/2010

**Tempo a disposizione: 3 ore.**

1. Siano  $f(n)$  e  $g(n)$  due funzioni asintoticamente positive e crescenti. Dimostrare la verità o la falsità della seguente affermazione:

$$\log(f(n) \cdot g(n)) = O(\max\{\log f(n), \log g(n)\})$$

2. Risolvere la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ 7T(n/2) + n^3 \log n & \text{se } n > 1 \end{cases}$$

3. Scrivere un algoritmo ricorsivo efficiente che cancelli da un Albero Binario di Ricerca  $T$  (i cui nodi contengono solo il campo chiave, figlio destro e figlio sinistro) ogni nodo, diverso dalla radice dell'albero, che soddisfa la seguente proprietà:

“contiene una chiave pari minore di  $k$  ed è radice di un sottoalbero di altezza minore di  $H$ ”

dove  $H$  è un valore fornito in ingresso. Si noti che la proprietà dei nodi da cancellare è da intendersi rispetto all'albero originario  $T$  in ingresso. Non è ammesso l'uso di variabili globali né di passaggio di parametri per riferimento.

4. Dato un grafo orientato  $G$  e un vertice  $s$  di  $G$ , scrivere un algoritmo che in tempo lineare sulla dimensione del grafo, stampi, senza ripetizioni, tutti i vertici di  $G$  che o raggiungono  $s$  o sono da  $s$  raggiungibili.

```

Algo(G, v, u)
    init(G)
    arr = vuoto
    dfsVisit(G, v, u, false, arr)
    while(arr!=vuoto)
        x = pop(arr)
        xVisit(x)

```

```

dfsVisit(G, v, u, b, arr)
    if (b = true)
        print(v)
        colore[v] = ROSSO
    else
        colore[v] = GRIGIO

foreach a in adiac[v]
    if (a = u)
        b = true
    if (colore[a] = BIANCO)
        dfsVisit(G, a, u, b, arr)
    else if (colore[a] = GRIGIO)
        push(arr, a)
    colore[v] = nero

```

```
Init(G)
    for each v in V[G]
        colore[v] = bianco
```

---

```
DFSVisit(G, G', s)
    colore[s] = grigio
    for each v in adiacenti[s]
        if (colore[v] = bianco)
            aggiungi(G', s, v)
            DFSVisit(G, G', v)
    colore[s] = nero
```

---

```
Algoritmo(G, s)
    Init(G)
    G' = CreaGrafo()
    DFSVisit(trasposta(G), G', s)
    return G'
```

```

algo(T, F, k1, k2, h)
    max = -1
    if (T != NIL)
        p = ALGO(sx[T], T, k1, k2, h+1)
        q = ALGO(dx[T], T, k1, k2, h+1)
        if (p > max)
            max = p
        else if (q > max)
            max = q
        if (key[T] >= k1) AND (key[T] <= k2) AND (key[T] % 2 = 0)
            if (h > max)
                max = h
            cancella(T, F)
    return max

```

-----

```

staccaMin(T, F)
    if (T != NIL)
        if (sx[T] != NIL)
            return staccaMin(sx[T], T)
        else //trovato
            if (T = sx[F])
                sx[F] = dx[T]
            else
                dx[F] = dx[T]
    return T

```

-----

```

cancella(T, F)
    nodo = T
    if (sx[T] = NIL)
        if (T = sx[F])
            sx[F] = dx[T]
        else
            dx[F] = dx[T]
    else if (dx[T] = NIL)
        if (T = sx[F])
            sx[F] = sx[T]
        else
            dx[F] = sx[T]
    else
        nodo = staccaMin(dx[T], T)
        key[T] = key[nodo]
        dealloca[nodo]

```

```

algo(T, h1, h2, n1, n2, k1, k2, F, h)
    n = 0
    current = 0

    if (T != NIL)
        nsx = ALGO(sx[T], h1, h2, n1, n2, k1, k2, T, h+1)
        ndx = ALGO(dx[T], h1, h2, n1, n2, k1, k2, T, h+1)
        n = nsx + ndx
        if ((key[T] >= k1) && (key[T] <= k2)) //compreso tra k1 e k2
            current = 1
            if ((key[T] % 2) = 0) //è pari
                if ((h >= h1) && (h <= h2))
                    if ((n >= n1) && (n <= n2))
                        cancella(T, F);

    return n + current

```

-----

```

staccaMin(T, F)
    if (T != NIL)
        if (sx[T] != NIL)
            return staccaMin(sx[T], T)
        else //trovato
            if (T = sx[F])
                sx[F] = dx[T]
            else
                dx[F] = dx[T]
    return T

```

-----

```

cancella(T, F)
    nodo = T
    if (sx[T] = NIL)
        if (T = sx[F])
            sx[F] = dx[T]
        else
            dx[F] = dx[T]
    else if (dx[T] = NIL)
        if (T = sx[F])
            sx[F] = sx[T]
        else
            dx[F] = sx[T]
    else
        nodo = staccaMin(dx[T], T)
        key[T] = key[nodo]
        dealloca[nodo]

```

```
Algo(G, x, y)
  if (G != NIL) AND (x != NIL) AND (y != NIL)
    G' = trasposta(G)
    BFS(G', x, y)
```

---

```
BFS(G, x, y)
  init(G)
  queue = vuota

  colore[x] = GRIGIO
  push(queue, x)
  print(key[x])

  while(queue != vuota)
    h = pop(queue)
    foreach (v in adiac[h])
      if (colore[v] = BIANCO)
        colore[v] = GRIGIO
        if (v != y)
          push(queue, v)
          print(key[v])
    colore[h] = NERO
```

---

```
init(G)
  foreach v in V[G]
    colore[v] = BIANCO
```

```

algo(T,F,A,h,p,r)
if (T != NIL)
    q = ricBin(key[T], A, h, p, r)
    if (q < 0)
        algo(sx[T], T, A, h+1, p, r)
        algo(dx[T], T, A, h+1, p, r)
    else
        algo(sx[T], T, A, h+1, p, q)
        algo(dx[T], T, A, h+1, q, r)
        if (h >= 1)
            cancella(T, F)

```

-----

```

ricBin(k, A, p, r)
if (r > p)
    q = (r - p) / 2
    if (k < A[q])
        return ricBin(k, A, p, q)
    else if (k > A[q])
        return ricBin(k, A, q, r)
    else
        return q
else
    return -1

```

-----

```

staccaMin(T, F)
if (T != NIL)
    if (sx[T] != NIL)
        return staccaMin(sx[T], T)
    else //trovato
        if (T = sx[F])
            sx[F] = dx[T]
        else
            dx[F] = dx[T]
return T

```

-----

```

cancella(T, F)
nodo = T
if (sx[T] = NIL)
    if (T = sx[F])
        sx[F] = dx[T]
    else
        dx[F] = dx[T]
else if (dx[T] = NIL)
    if (T = sx[F])
        sx[F] = sx[T]
    else
        dx[F] = sx[T]
else
    nodo = staccaMin(dx[T], T)
    key[T] = key[nodo]
    dealloca[nodo]

```

```
algo(G, v, A)
    G' = trasposta di G
    return distanzaK(G', v, A)
```

---

```
init(G)
    if (G != NIL)
        for each v in G
            colore[v] = bianco
            dist[v] = -1
            ina[v] = false
```

---

```
aizefy(A)
    if (A != NIL)
        for each v in A
            ina[v] = true
```

---

```
distanzaK(G, s, A)
    init(G')
    aizefy(A)

    queue = vuoto

    colore[s] = grigio
    dist[s] = 0
    if (ina[s] = true)
        return 0
    push(queue, s)

    while (size(queue) > 0)
        x = pop(queue)
        for each v in adiac[x]
            if (colore[v] = bianco)
                colore[v] = grigio
                dist[v] = dist[x] + 1
                if (ina[v] = true)
                    return dist[v]
                push(queue, v)
            colore[x] = nero

    return -1
```

definiamo H le iterazioni del ciclo interno e K quelle del ciclo esterno  
e  $y^l$  come il valore iniziale di y prima del ciclo interno

Analizziamo il ciclo interno

$y_1 = y^l - 2$

$y_2 = y^l - 2 - 2$

$y_H = y^l - 2H$

Calcoliamo la H

Dal ciclo interno si uscirà quando  $y_H = x_K$

quindi  $y^l - 2H = x_K$

analizzando il codice si vede che  $y^l = y_K + x_K$

sostituendo e avremo  $y_K + x_K - 2H = x_K \implies H = (x_K - x_K + y_K) / 2 \implies H = y_K / 2$

calcoliamo il ciclo esterno:

$x_1 = x_0 / 2$

$x_2 = x_0 / 2 / 2 \Rightarrow x_0 / 4$

$x_K = x_0 / 2^K \implies x_K = n / 2^K + 1$  Calcoliamo la K  $n / (2 * 2^K) = 2 \implies K = \log(N/4)$

calcoliamo la Y del ciclo esterno

$Y_1 = Y_0 + X_0 - 2H$  (sostituiamo H)  $\implies Y_0 + X_0 - 2(Y_0 / 2) \implies Y_0 + X_0 - Y_0 \Rightarrow X_0$

$Y_2 = Y_1 + X_1 - Y_1 \Rightarrow X_1$

$Y_K = X_{K-1}$

(sostituiamo)  $Y_K = N / 2^K + 1 - 1 \Rightarrow Y_K = N / 2^K$

ora che abbiamo tutto quello che ci serve calcoliamo la H

valeva  $H = Y_K / 2 \implies H = N / 2 * 2^K$

ora abbiamo veramente tutto.. calcoliamo la sommatoria

Som di  $K=0$  a  $\log N / 4$  di  $1 + (\text{somm di } H=0 \text{ a } n / 2 * 2^K)$  1

calcoli tutto e ti trovi  $\log n / 4 + n$

$O(n)$

```
Algo(T,P,x,i)
if t!=nil then
    if (sx[t] != nil)
        i=algo(sx[t],t,x,i)
    D = i+1
    if (dx[t] != nil)
        D=algo(dx[t],t,x,i+1)
    if (i+1=x)
        cancella(T,P)
    return D
else
    return i
```

# Tema d'esame di Algoritmi e Strutture Dati I

## 13/01/2021

1. Sia dato un grafo orientato  $G = \langle V, E \rangle$ , rappresentato con liste di adiacenza, e un vertice  $s$  e due insiemi di vertici  $B \subseteq V$  e  $C \subseteq V$ , rappresentati come array. Si scriva un algoritmo che, dati in ingresso  $G$ ,  $s$ ,  $B$  e  $C$ , collezioni in **tempo lineare sulla dimensione di  $G$**  in una lista  $L$  tutti i vertici  $v$  che soddisfano entrambe le seguenti condizioni:

- $v$  appartiene a  $B$  e può raggiungere  $s$  tramite un percorso;
- esiste anche un percorso da  $s$  a  $v$  che non passa per alcun vertice di  $C$ .

# Tema d'esame di Algoritmi e Strutture Dati I

## 13/04/2021

Sia dato il seguente algoritmo ricorsivo, in cui sia assuma data la funzione  $\text{BEST}(a,b,k)$ :

**Algorithm 1**  $\text{ALG}(T,k)$

```
1   ret = NIL
2   if  $T \neq NIL$  then
3       if  $T \rightarrow key > k$  then
4           b = T
5           ret = BEST(ALG( $T \rightarrow sx, k$ ), b)
6       else if  $T \rightarrow key < k$  then
7           a = T
8           ret = BEST(a, ALG( $T \rightarrow dx, k$ ))
9       else
10          a = ALG( $T \rightarrow sx, k$ )
11          b = ALG( $T \rightarrow dx, k$ )
12          ret = BEST(a, b, k)
13 return ret
```

Scrivere un algoritmo **iterativo** che **simuli precisamente** il comportamento ricorsivo dell'algoritmo sopra riportato.

# Algoritmi e Strutture Dati I

## Traccia A 16/06/2021

Si definisca un **albero orientato** come un grafo orinetato con le seguenti proprietà: (i) esiste un unico vertice  $s$  (detto radice) che raggiunge ogni altro vertice tramite un qualche percorso orientato; (ii) se un qualsiasi vertice  $v$  è raggiungibile da un arbitrario vertice  $u$ , allora **esiste un solo percorso orientato da  $u$  a  $v$** . Una **foresta orientata** è, allora, un grafo orientato formato da uno o più alberi orientati tra loro non connessi.

Si scriva un algoritmo che, in **tempo lineare sulla dimensione del grafo** orientato  $G = (V, E)$  fornito in ingresso, verifichi se  $G$  è una **foresta orientata** oppure no.

# Tema d'esame di Algoritmi e Strutture Dati I

## 21/01/2022

**Tempo a disposizione: 1.30 ore.**

1. Si risolva la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 1 & \text{se } n \leq 2 \\ 3 \cdot T(\sqrt{n}) + \log(n) & \text{se } n > 2 \end{cases}$$

2. Sia dato il seguente algoritmo:

```
1 ALGORITMO( $T, i$ )
1    $a = G(i)$ 
2   if  $T = Nil$  then
3     return  $a$ 
4   else
5      $z = ALGORITMO(T \rightarrow sx, 2 \cdot i)$ 
6      $a = z + (T \rightarrow key) \cdot i + a$ 
7      $z = a + ALGORITMO(T \rightarrow dx, 2 \cdot i + 1)$ 
8     return  $3 \cdot z$ 
```

dove  $G(\cdot)$  è una funzione esterna. Scrivere un algoritmo **iterativo** che **simuli precisamente** il comportamento dell'algoritmo ricorsivo sopra riportato.

# Equazioni di ricorrenza

## 1. [ Cormen ]

1.  $T(n) = 2T(n/2) + n^4$
2.  $T(n) = T(7n/10) + n$
3.  $T(n) = 16T(n/4) + n^2$
4.  $T(n) = 7T(n/3) + n^2$
5.  $T(n) = 7T(n/2) + n^2$
6.  $T(n) = 2T(n/4) + \sqrt{n}$
7.  $T(n) = T(n - 2) + n^2$
8.  $T(n) = 4T(n/3) + n \log_2 n$
9.  $T(n) = 3T(n/3) + n / \log_2 n$
10.  $T(n) = 4T(n/2) + n^2\sqrt{n}$
11.  $T(n) = 3T(n/3 - 2) + n/2$
12.  $T(n) = 2T(n/2) + n / \log_2 n$
13.  $T(n) = T(n/2) + T(n/4) + T(n/8) + n$
14.  $T(n) = T(n - 1) + 1/n$
15.  $T(n) = T(n - 1) + \log_2 n$
16.  $T(n) = T(n - 2) + 1 / \log_2 n$
17.  $T(n) = \sqrt{n}T(\sqrt{n}) + n$
18.  $T(n) = \sqrt[3]{n}T(\sqrt[3]{n}) + \sqrt{n}$

## 2. [ Jefferson ]

1.  $T(n) = 2T(n/4) + \sqrt{n}$
2.  $T(n) = 2T(n/4) + n$
3.  $T(n) = 2T(n/4) + n^2$
4.  $T(n) = 3T(n/3) + \sqrt{n}$
5.  $T(n) = 3T(n/3) + n$
6.  $T(n) = 3T(n/3) + n^2$
7.  $T(n) = 4T(n/2) + \sqrt{n}$
8.  $T(n) = 4T(n/2) + n$
9.  $T(n) = 4T(n/2) + n^2$
10.  $T(n) = T(n/2) + T(n/3) + T(n/6) + n$
11.  $T(n) = T(n/2) + 2T(n/3) + 3T(n/4) + n^2$
12.  $T(n) = T(n/15) + T(n/10) + 2T(n/6) + \sqrt{n}$
13.  $T(n) = 2T(n/2) + O(n \log n)$
14.  $T(n) = 2T(n/2) + O(n / \log n)$
15.  $T(n) = \sqrt{n}T(\sqrt{n}) + n$
16.  $T(n) = \sqrt{2n}T(\sqrt{2n}) + \sqrt{n}$

## 3. [ Esami ]

1.  $T(n) = 2\sqrt{n}T(\sqrt{n}) + n$
2.  $T(n) = 2T(\sqrt{n}) + \log n$

3.  $T(n) = 4T(n/4) + \sqrt{n}$
4.  $T(n) = 3T(n/4) + \sqrt{n}$
5.  $T(n) = 4T(n/9) + \sqrt{n}$
6.  $T(n) = 15T(n/4) + n^2 \log n$
7.  $T(n) = 2T(n/2) + T(n/4) + n$
8.  $T(n) = 17T(n/2) + n^4$
9.  $T(n) = T(3n/4) + T(n/2) + n$
10.  $T(n) = 6T(n/8) + \sqrt{n}$
11.  $T(n) = T(n/3) + T(n/4) + n$
12.  $T(n) = 7T(n/2) + n^3 \log n$
13.  $T(n) = T(2n/3) + T(n/3) + n^3 ?$
14.  $T(n) = 3T(\sqrt{n}) + \log n$

## Asintoticità

### 1. [ Esami ]

1. Si dimostri la verità o la falsità della seguente affermazione:

1. Se  $2^{f(n)} = \Theta(2^{g(n)})$ , allora  $f(n) = \Theta(g(n))$
2. Se  $f(n) = O(g(n))$ , allora  $\sqrt{g(n)} = \Omega(\sqrt{f(n)})$
3. Se  $h(n) = \Theta(t(n))$  e  $f(n)/h(n) = \Theta(g(n))$ , allora  $\frac{f^2(n)}{h(n) \cdot t(n)} = \Theta(g^2(n))$
4. Se  $f(n) = \Theta(n)$  e  $g(n) = \Theta(2^{n^2})$ , allora  $2^{2 \cdot \log f(n)} = \Theta(\log(g(n)))$
5. Se  $f, g$  sono due funzioni asintoticamente positive e crescenti, allora  $\log(f(n) \cdot g(n)) = O(\max\{\log(f(n)), \log(g(n))\})$
6. Se  $h^2(n) = \Theta(\min\{f(n), g(n)\})$ , allora  $\sqrt{f(n)} = O(h(n))$
7. Se  $\sqrt{h(n)} = O(\min\{f(n), g(n)\})$ , allora  $h(n) = O(g^2(n))$
8. Se  $2^{f(n)} = \Theta(g(n))$  e  $g(n) = \Theta(h(n)^k)$  per una costante  $k > 0$ , allora  $f(n) = \Theta(\log h(n))$
9. Se  $f(n) = \Theta(n)$  e  $g(n) = \Theta(2^n)$ , allora  $2^{f(n)} = \Theta(g(n))$
10. Se  $\sqrt{h(n)} = O(2^{f(n) \cdot g(n)})$ , allora  $\log(\log h(n)) = \Theta(g(n) \cdot \log f(n))$
11. Se  $h(n) = \Theta(\max\{\log \log f(n), \log \log g(n)\})$ , allora  $g(n) = O(2^{2^{h(n)}})$
12. Se  $z(n) = \Theta(2^{g(n)})$  e  $h(n) = \Theta(\log g(n))$ , allora  $\log z(n) = \Theta(2^{h(n)})$
13. Se  $f(n) = \Theta(\sqrt{g(n)})$  e  $2^{g(n)} = \Theta(2^n)$ , allora  $\log f(n) = \Theta(\log n)$
14. Se  $h(n) = \Theta(t(n))$  e  $f(n) = \Theta(g(n))$ , allora  $g(n) + h(n) = \Theta(t(n) + f(n))$
15. Se  $h(n) = \Theta(t(n))$  e  $f(n) = \Theta(g(n))$ , allora  $\log_2(g(n) \cdot h(n)) = \Theta(\log_2(t(n) \cdot f(n)))$
16. Se  $\log \frac{f(n)}{g(n)} = \Theta(\log(t(n) \cdot g(n)))$  e  $\log \frac{t(n)}{g(n)} = \Theta(\log \frac{h(n)}{f(n)})$ , allora  $\log h(n) = \Theta(\log t(n))$
17. Se  $f(n) = \Theta(\sqrt{g(n)})$  e  $f(n) = \Theta(k^2(n))$ , allora  $g(n) = \Theta(k(n)^4)$
2. Si trovino, se esistono, le costanti per soddisfare la seguente relazione asintotica:
  1.  $5n^2 - 8\sqrt{n} + 1 = \Theta(n^2)$
  2.  $\log \frac{n}{7} = \Theta(\log n^4)$
  3.  $4n^2 - 7\sqrt{n} + 2 = \Theta(n^2)$

- 
- 4.  $2n - \log \frac{n}{4} = \Theta(n)$
  - 5.  $2 \log_2(n) - 4/n = \Theta(\log_2 n)$
  - 6.  $n - \log_2(n) + 1 = \Theta(n)$
  - 7.  $n^2 \log(n^2) + 15n^2 = \Theta(n^2 \log(n))$
  - 8.  $7n\sqrt{n} + 3n - 10\sqrt{n} = \Theta(n^{3/2})$
  - 9.  $\log_2(2^n \cdot \frac{4^n}{n}) = \Theta(\log_2(3^{3n}))$
- 

68

- [Link Forum](#)
- [Link Github](#)