Algoritmi e Basi di dati – Modulo di Algoritmi e Strutture dati

2 luglio 2010 - ANNO ACCADEMICO 2009/10

NOME: **COGNOME:** **MATRICOLA:**

Esercizio 1

Sia dato il seguente vettore A di interi:

[58, 47, 4, 3, 98, 33, 91]

Scrivere tutte le chiamate a quicksort generate dalla chiamata quicksort(A, 0, 6). Per ogni chiamata indicare lo stato dell'array e il valore dei parametri attuali e del perno.

Array A	*inf*	*sup*	*perno*
[58, 47, 4, 3, 98, 33, 91]	0	6	3
[3, 47, 4, 58, 98, 33, 91]	1	6	58
[3, 47, 4, 33, 98, 58, 91]	1	3	4
[3, 4, 47, 33, 98, 58, 91]	2	3	47
[3, 4, 33, 47, 98, 58, 91]	4	6	58
[3, 4, 33, 47, 58, 98, 91]	5	6	98
[3, 4, 33, 47, 58, 91, 98]			

Esercizio 2

Siano date le seguenti funzioni

Dire se ognuna è O grande dell'altra. In caso affermativo, mostrare una coppia (n_0, c), altrimenti fornire una motivazione.

$f(x) = \begin{cases} 4x^4 + 5 & \text{se } x <= 10 \\ 3x^2 + 1 & \text{se } x>10 \text{ e quadrato perfetto} \\ 5x & \text{altrimenti} \end{cases}$	$g(x) = \begin{cases} x^2 \log_2 x + 10 & \text{per } x \text{ primo} \\ 2x^2+x & \text{altrimenti} \end{cases}$

f è O(g) (basta scegliere ad esempio n_0=11 e c = 2)

g non è O(f) in quanto esistono infiniti numeri primi (e non primi che non siano quadrati perfetti)

1	*2*	*3*	*4*	*5*

Esercizio 3
Calcolare la complessità della chiamata di funzione g(n) e del suo risultato in funzione di n (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) con le funzioni f e g definite come segue:
Le funzioni **f** e **g** sono definite come segue:

```
int f(int x) {
  if (x<=1) return 1;
  int a=0;
  for (int i=0; i <=1; i++)
     a+=f(x/2);
  return a+a+a+a+1;
}
```

```
int g(int x) {
  if x <= 1 return 1;
  int a=0;
  for (int i=0; i <= f(f(x)); i++)
     a++;
  return a + g(x-1);
}
```

$T_f(1)= k_1$
$T_f(n)= k_2 + 2T_f(n/2)$ $T_f(n)$ è $O(n)$
$R_f(1)= 1$
$R_f(n)= 1 + 8R_f(n/2)$ $R_f(n)$ è $O(n^3)$

Calcolo $T_g(n)$
for:
numero iterazioni: $O(n^9)$ Complessità singola iterazione: $O(n^3)$
complessità del for: $O(n^{12})$

$T_g(1)= k_1$
$T_g(n)= k_2\, n^{12} + T_g(n-1)$ $T_g(n)$ è $O(n^{13})$
$R_g(0,1)= 1$
$R_g(n)= n^9 + R_g(n-1)$ $R_g(n)$ è $O(n^{10})$

Esercizio 4
Sia dato un albero binario, non vuoto, contenente etichette intere comprese nell'intervallo (h,k). Scrivere una funzione controlla che ritorna true se la massima etichetta è maggiore del doppio della minima etichetta presente nell'albero, e false altrimenti. Calcolare la complessità della soluzione proposta nel numero di nodi dell'albero.

```
void min_max(Node* t, int& m, int& M)  {
  if (!t) {
    m = k;
    M = h;
    return;
  }
  int ml, mr, Ml, Mr;
  min_max(t->left,ml,Ml);
  min_max(t->right,mr,Mr);
  m = min(t->label, min(ml,mr));
  M = max(t->label, max(Ml,Mr));
  return;
}

bool controlla(Node* t) {
  int m, M;
  min_max(t,m,M);
  return (M>2*m);
}
```

Complessità: O(n)

Esercizio 5
Sia dato un albero generico ad etichette intere memorizzato figlio-fratello. Si scriva una funzione modifica che somma ad ogni nodo il valore massimo fra il livello del nodo e il numero dei suoi figli.

```
void modifica(Node* t, int
level=0) {
  if (t) {
    // conta i figli
    Node* f = t->left;
    int num_figli = 0;
    while (f) {
      ++num_figli;
      f = f->right;
    }
    t->label += max(level,
num_figli);
    modifica(t->left,level+1);
    modifica(t->right,level);

  }
}
```

```
int modifica2(Node* t, int level=0)
{
  if (!t) return 0;
  t->label += max(level,
     modifica2(t->left, level+1));
  return modifica2(t->right,level)
+1;
}
```