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

ANNO ACCADEMICO 2009/2010- 25 Febbraio 2011

Esercizio1

Sia dato lo heap : [70 65 60 50 60 53 50 45 51 54]
Indicarne il contenuto dopo 2 estrazioni e il successivo inserimento del valore 44. Indicare tutte le chiamate a down e up effettuate durante le operazioni

Contenuto heap	Chiamate a down/up	
65 60 60 50 54 53 50 45 49	down(0) down(1) down(4)	dopo una estrazione
60 54 60 50 49 53 50 45	down(0) down(1) down(4)	dopo la seconda estrazione
60 54 60 50 49 53 50 45 44	up(8)	dopo l'inserimento del valore

Esercizio 2

Dire per ogni coppia di funzioni sotto indicate, se una è O dell'altra. In caso affermativo fornire una coppia (n0,c), in caso negativo giustificare la risposta.

$$f(n)= \begin{cases} 4n^2 & \text{se n è multiplo di 9} \\ 60\ n^3 & \text{altrimenti} \end{cases}$$

$$g(n)= \begin{cases} n^3 & \text{se n è multiplo di 3} \\ 10\ n^3 & \text{altrimenti} \end{cases}$$

$h(n)= 100\ n^2$

	SI/NO	dimostrazione
f è O(g) ?	Si	n0 = 1 c=60
f è O(h) ?	no	Ci sono infiniti numeri che non sono multipli di 9
g è O(f) ?	no	Ci sono infiniti numeri che sono multipli sia di 3 che di 9
g è O(h) ?	no	Ci sono infiniti numeri che sono multipli di 9
h è O(f) ?	si	n0 = 1 c=25
h è O(g) ?	si	n0 = 1 c=100

Esercizio 3

Calcolare la complessità del for in funzione di n.

```
for (int i=0; i <= f(g(n)); i++) cout << i;
```

con le funzioni **f** e **g** definite come segue. Indicare per esteso le relazioni di ricorrenza e, per ogni comando ripetitivo, il numero di iterazioni e la complessità della singola iterazione.

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

```
int f(int x) {
if (x<=0) return 1;
for (int i=0; i <= g(100); i++)
    cout << g(x);
return g(x) + f(x-1);
}
```

Funzione g

Numero di iterazioni del for: n^2
Complessità della singola iterazione: O(1)
Complessità del for: =O(n^2)
Il contenuto di a all'uscita dal for è O(n^4): somma dei primi n^2 numeri

$T_g(0)= d$
$T_g(n)= cn^2+ T_g(n/2)$ $T_g(n)$ è $O(n^2)$

$R_g(0)= d$
$R_g(n)= cn^4+ R_g(n/2)$ $R_g(n)$ è $O(n^4)$

Funzione f

Numero di iterazioni del for: =O(1)
Complessità della singola iterazione: O(n^2)
Complessità del for: =O(n^2)

$T_f(0)= d$
$T_f(n)= cn^2+ T_f(n-2)$ $T_f(n)$ è $O(n^3)$

$R_f(0)= 1$
$R_f(n)= cn^4+ R_f(n-1)$ $R_f(n)$ è $O(n^5)$

For esterno

Numero di iterazioni del for: = risultato di f(g(n))= f(n^4)= O((n^4)^5)= O(n^20)
Complessità della singola iterazione: tempo per il calcolo di g(n) + tempo per il calcolo di f(n^4)= O(n^2) + O((n^4)^3)= O(n^12)
Complessità del for: O(n^12) * O(n^20) = O(n^32)

Esercizio 4
Scrivere una funzione che, dato un albero binario con etichette intere >0, restituisce l'etichetta della foglia di livello maggiore. Se ci sono più foglie all'ultimo livello, restituisce quella più a destra fra queste.

```
int level(Node* t, int l, int &foglia) {
if (!t) { foglia=0; return -1; }
if (! t->left && ! t->left) { foglia=t->label; return 0; }
int foglia_left, foglia_right;
int level_left= level (t->left, foglia_left);
int level_right= level (t->left, foglia_left);
if (level_left > level_right )
        { foglia=foglia_left; return level_left +1; }
   else { foglia=foglia_right; return level_right +1; }
}
```

Esercizio 5
Scrivere una funzione **`void stampa_nipoti(Node* t)`** che, dato un albero generico memorizzato con la memorizzazione figlio-fratello, stampa il numero di "nipoti" (figli di figli) di ogni nodo.

```
int figli (Node* t) {
if (!t) return 0;
return 1 + figli(t->right);
}
int nipoti (Node* t) {
if (!t) return 0;
return figli(t->left) + nipoti(t->right);
}
void stampa-nipoti (Node* t) {
if (!t) return;
cout << nipoti(t->left);
stampa-nipoti (t->left);
stampa-nipoti (t->right);
}
```