Algoritmi e basi di dati – modulo Algoritmi e Strutture dati a.a. 2010/2011
8 luglio 2011

Esercizio1

Dimostrare che, se f(n) è O(g(n)) e h(n) è O(s(n)), allora f(n)+g(n) è O(g(n)+s(n)).
Soluzione. Siano
n1 e c1 tali che per ogni n>n1 f(n) <= c1g(n) e
n2 e c2 tali che per ogni n>n2 h(n) <= c2s(n)
abbiamo che
per n >= max(n1,n2) f(n)+g(n) <= c1g(n) + c2s(n) da cui
per n >= max(n1,n2) f(n)+g(n) <= max(c1,c2) (g(n) + s(n))
quindi n0= max(n1,n2) e c= max(c1,c2)

Esercizio2

Dato lo heap = [50 28 15 22 9 10 8], indicare lo heap dopo una estrazione e il successivo inserimento del valore 16. Indicare per ogni operazione le chiamate a up e down con relativo argomento

28 22 15 8 9 10	down(0), down(1), down(3)	Dopo l'estrazione
28 22 16 8 9 10 15	up(6), up(2)	Dopo l'inserimento di 16

Esercizio3

Calcolare la complessità del for in funzione di n >0.

```
for (int i=0; i <= g(g(n))+f(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; b=0;
  for (int i=0; i <= x; i++){
    a+=i;
    b+=1; }
  for (int j=a; j >= 0; j--)
    cout << b;
  return a/b + g (x-1);
}
```

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

Funzione g
Primo for :
Complessità della singola iterazione: O(1)
Numero di iterazioni : O(n)
Complessità del for: =O(n)

Secondo for :
Complessità della singola iterazione: O(1)
Numero di iterazioni : O(n^2) (a è O(n^2), b=n)

Complessità del for: =O(n^2)

$T_g(0)$= d
$T_g(n)$= cn^2+ $T_g(n-1)$ $T_g(n)$ è O(n^3)

$R_g(0)$= d
$R_g(n)$= cn+ $R_f(n-1)$ $R_g(n)$ è O(n^2) (a/b=O(n))

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

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

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

Calcolo for del blocco:
numero iterazioni: g(g(n)) + f(n)= g(n^2) + O(n^2logn)= O(n^4)+O(n^2logn)= O(n^4)
Complessità della singola iterazione: $T_g(n)$ + $T_g(n^2)$ + $T_f(n)$ = O(n^3) + O(n^6) + O(n^3) = O(n^6)
Complessità del for: =O(n^10)

Esercizio 4

Scrivere una funzione che, dato un albero binario t con etichette intere e due etichette x e y, conta i nodi che hanno più nodi con etichetta x che nodi con etichetta y nel proprio sottoalbero.

```
int conta (Node* t, int x, int y, int & quantix, int & quantiy) {
    if (!t) {quantix=0; quantiy=0; return 0; }
    int quantix_l, quantix_r, quantiy_l,quantiy_r;
    int l = conta(t->left, quantix_l,quantiy_l);
    int r = conta(t->right, quantix_r,quantiy_r);
    quantix= quantix_l + quantix_r + (t->label==x);
    quantiy= quantiy_l + quantiy_r + (t->label==y);
    return l + r + (quantix > quantiy);
    }
```

Esercizio 5

Scrivere una funzione che, dato un albero generico e una etichetta x, elimina il primo figlio di x, se esiste, e inserisce i suoi eventuali figli come figli di x.

```
void elimina (Node* t, int x) {
    Node * a= find(x,t);
    if (!a || !a->left ) return;
    Node * b=a->left;
    if (!b->left) {
       a->left = b->right;
       b->right = 0;
       delete b;
       return;
    }
    a->left = b->left;
    for (c=b->left; c->right != 0; c=c->right);
    c->right = b->right;
    b->right=b->left=0;
    delete b;
}
```