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

**ANNO ACCADEMICO 2009/2010-11 Febbraio 2011**

**Esercizio1**

Sia dato il seguente vettore A di interi

[66 4 16 94 43 14 35]

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

| *A* | *Inf* | *Sup* | *Perno* |
|---|---|---|---|
| [66 4 16 94 43 14 35 ] | 0 | 6 | 94 |
| [66 4 16 35 43 14 94 ] | 0 | 5 | 16 |
| [14 4 16 35 43 66 94 ] | 0 | 1 | 14 |
| [4 14 16 35 43 66 94 ] | 3 | 5 | 43 |
| [4 14 16 35 43 66 94 ] | | | |

**Esercizio 2**

Si consideri la memorizzazione di un albero generico (senza etichette) mediante un array come nell'algoritmo di Kruskal, in cui cioè ogni elemento dell'array è un nodo e contiene l'indice del padre. I nodi sono numerati da 0 a n-1 e t[0]=-1. Supponendo di avere un array `tree` con n elementi, scrivere dei frammenti di codice per le seguenti richieste.

1. Stampare il padre del nodo x
2. Stampare tutti figli del nodo x
3. Stampare tutti gli antenati del nodo x
4. Stampare tutti i fratelli del nodo x
5. Stampare il primo figlio del primo figlio del nodo x

**SOLUZIONE**

```
1. if (t[x] !=-1) cout << t[x];

2. for (int i=0; i<n; i++) if (t[i] ==x) cout << i;

3. for (int i=x; t[i]!=-1; i=t[i]) cout << i;

4. for (int i=0; i<n; i++) if (t[i] ==t[x] && i!=x) cout << t[i];

5.
int figlio = -1;
for (int i=0; i<n; i++) if (t[i]==x) { figlio=i; break; }
if (figlio >=0)for (i=0; i<n; i++) if (t[i]==figlio) { cout << i; break; }
```

**Esercizio 3**

Calcolare la complessità del blocco in funzione del numero di nodi dell'albero binario t:

```
{
  int a = 0;
  for (int i=0; i <= Nodes(t); i++)
     {a += f(t);
      es5(t);
     }
}
```

con le funzioni **`f`** e **`es5`** 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.

```
   void es5 (Node* t ) {
   if (!t) return;
   t->label*= f (t);
   es5 (t->left);
   es5 (t->left);
   es5 (t->right);
   es5 (t->right);
      }
```

```
int f(Node* t) {
if (!t) return 1;
for (int i=0; i <= Nodes(t); i++)
     cout << Nodes(t);
return 1 + f(t->left);
}
```

Funzione f

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

Funzione es5

$T_{es5}(0)$=d
$T_{es5}(n)$= cn^2+ 4 $T_{es5}(n/2)$ $T_{es5}(n)$ è O(n^2logn)

Calcolo for del blocco:
numero iterazioni: n
Complessità della singola iterazione: $T_{nodes}(n) + T_f(n) + T_{es5}(n)$ = O(n) + O(n^2) + O(n^2logn) = O(n^2logn)
Complessità del for: O(n^3logn)

**Esercizio 4**
Scrivere una funzione **`bool sotto-albero(Node* t1, Node* t2)`** che, dati due alberi binari, restituisce true se t2 coincide con un sottoalbero di t1.

```
bool sotto_albero (Node* t1, Node* t2) {
if (!t2) return true;
if (!t1) return false;
if (uguali (t1,t2)) return true;
return sotto_albero (t1->left, t2) || sotto_albero (t1->right, t2)
}


bool uguali (Node* t1, Node* t2) {
if (!t1 && !t2) return true;
if (!t1 && t2) || (t1 && !t2) return false;
if (t1->label!=t2->label) return false;
return uguali (t1->left, t2->left) && uguali (t1->right, t2->right);


}
```

**Esercizio 5**
Scrivere una funzione **`void esau(Node* t)`** che, dato un albero generico memorizzato con la memorizzazione figlio-fratello, scambia il primo con il secondo sottoalbero di ogni nodo che ha almeno due sottoalberi.

```
void esau (Node* t) {
if (!t) return;
if (t->left && t->left->right) {
  Node* temp= t->left->right;
  t->left->right= temp->right;
  temp->right=t->left;
  t->left=temp;
  }
esau (t->left);
esau (t->right);
}
```