Algoritmi e Strutture Dati
Anno accademico 20016/2017- 22 febbraio 2018

1	2	3	4	5	6
5	6	5	6	6	5

Esercizio1

a) Definire quando una funzione è O di un'altra.

b) Confrontare le due funzioni F e G dal punto di vista della complessità: dire se una è O dell'altra e viceversa. In caso affermativo, indicare una coppia (n0,c), in caso negativo, giustificare la risposta.

F(x)= 3x^2 x pari; 50x^3 x dispari

G(x)= 9x^2 x divisore di 255; x^3 altrimenti

F è O(G) n0=256, c=51

G non è O(F) perché ci sono infiniti numeri pari dove G vale x^3 e F 3x^2.

c) Fare un esempio di funzioni incommensurabili.

Esercizio2

a) Descrivere il tipo di dato heap: definizione, operazioni e loro complessità, memorizzazione.

b) Un array ordinato in ordina decrescente è sempre uno heap? Se sì, dimostrarlo, se no, mostrare un contro esempio. **SI, perché ogni elemento è maggiore o uguale dei suoi due figli.**

c) Uno heap è sempre un array ordinato in ordine decrescente? Se sì, dimostrarlo, se no, mostrare un contro esempio. **NO, vedi ad esempio lo heap [9 3 4]**

d) Sia 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

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(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;
  for (int j=a; j >= 0;j--)b +=j;
     return b/a + g (x-1);
            }
```

```
int f(int x) {
if (x<=0) return 1;
int a=g(x);
int b= 5 + 2*f(x/2);
for (int i=0; i <= a; i++)
          cout << i;
return a*a + 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)
Complessità del for: =O(n^2)

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

Rg(0)= d
Rg(n)= cn^2+ Rf(n-1) Rg(n) è O(n^3)

Funzione f
for :
Complessità della singola iterazione: O(1)
Numero di iterazioni : O(n^3)
Complessità del for: =O(n^3)

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

Rf(0)= d
Rf(n)= cn^6+ 2Rf(n/2) Rf(n) è O(n^6)

Calcolo for del blocco:
numero iterazioni: g(n) + f(n)= O(n^3) + O(n^6)= O(n^6)
Complessità della singola iterazione: Tg(n) + Tf(n) = O(n^3) + O(n^3) = O(n^3)
Complessità del for: =O(n^9)

Esercizio 4

a) Definire il tipo di dato “albero binario”.
b) 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

a) Definire il tipo di dato “albero generico”.
b) Scrivere una funzione che, dato un albero generico (memorizzato figlio-fratello) 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; c=c->right);
    c->right = b->right;
    b->right=b->left=0;
    delete b;
    }
```

Esercizio 6

a) Spiegare il funzionamento delle funzioni virtuali in c++;
b) Indicare l'output del seguente programma
    1) Così come è scritto
    2) Togliendo tutte le occorrenze dalla parola `virtual`

```
#include <iostream>
using namespace std;

class A {
public:
    int i;
    A(int i) : i(i) {
      cout << "A(" << i <<")" <<
endl;
    }
    virtual int f() { return i; }
    virtual ~A() {
      cout << "delA(" << i << ")"
<< endl;
    }
};

class B : public A {
public:
    int j;
    B(int f) : A(f+2) { j=90;
      cout << "B(" << j <<")" <<
endl;
    }
    int f() { return 50; }
    virtual ~B() { cout << "delB()"
<< endl; }
};

class C : public B {
public:
    int i;
    C(int k) : B(k-1), i(k) {
      cout << "C(" << k <<")" <<
endl;
    }
    int f() { return i+1000; }
    virtual ~C() { cout << "delC()"
<< endl; }
};

int main() {
    C* c = new C(10);
    A* ref = c;
    cout << ref->i << endl;
    cout << c->i << endl;
    cout << c->f() << endl;
    cout << ref->f() << endl;
    delete ref;
}
```

1)

```
A(11)
B(90)
C(10)
11
10
1010
1010
delC()
delB()
delA(11)
```

2)

```
A(11)
B(90)
C(10)
11
10
1010
11
delA(11)
```