**FONDAMENTI DI INFORMATICA II –Algoritmi e Strutture Dati**

**a.a** 2015/16 – 5 luglio 2016

**Esercizio 1**

a) Descrivere il tipo di dato heap: definizione, operazioni e loro complessità, memorizzazione
b) Un array ordinato in ordine decrescente è uno heap? SI
c) Uno heap è un array ordinato in ordine decrescente? NO
d) Sia dato lo heap:

**[ 100 77 87 25 56 34 12 ]**

mostrare lo stato dello stesso e le chiamate ad up e down

1) Dopo l'inserimento del valore 33
2) Dopo l'estrazione di un elemento (a partire dallo heap ottenuto al passo a)

1)

[100 77 87 33 56 34 12 25 ]  up(7)  up(3)

2)

[87 77 34 33 56 25 12 ]  down(0) down(2) down(5)

**Esercizio 2**. Calcolare la complessità del comando (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) in funzione di n:

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

con le funzioni **f** e **g** definite come segue:

```
int f(int x) {
   if (x<=1)
     return 1;
   cout << 2*f(x/2)+f(X/2);
   return f(x/2)+f(x/2)+1;
}
```

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

Indicare per esteso le relazioni di ricorrenza e, per ogni comando ripetitivo, il numero di iterazioni e la complessità della singola iterazione.

$T_f(0,1)= k_1$
$T_f(n)= k_2 + 4\ T_f(n/2)$  $T_f(n)$ è $O(n^2)$
$R_f(0,1)= 1$
$R_f(n)= 1 + 2R_f(n/2)$  $R_f(n)$ è $O(n)$

Calcolo $T_g(n)$
numero iterazioni: $O(n)$
complessità singola iterazione: $O(n^2)$
complessità dei for: $O(n^3)$

il valore di alla fine e' $O(n^2)$

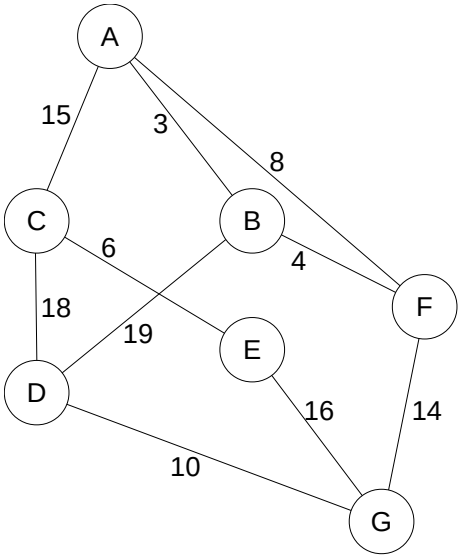
$T_g(0,1)= a$
$T_g(n)= bn^3 + T_g(n-1)$  $T_f(n)$ è $O(n^4)$
$R_g(0,1)= 1$
$R_g(n)= n^2 + R_g(n-1)$  $R_f(n)$ è $O(n^3)$

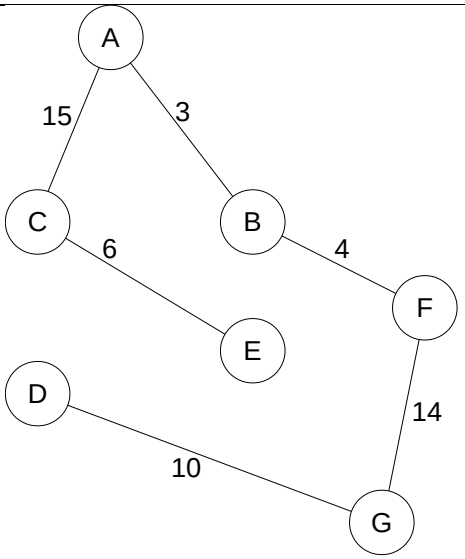
Calcolo for del blocco:
numero iterazioni: $O(n^3)$
Complessità singola iterazione: $T_g(n)+ T_f(n^3)+ T_f(n)= O(n^3) + O(n^6) + O(n^2)= O(n^6)$
Complessità totale blocco = $O(n^3)* O(n^6) = O(n^9)$

**Esercizio 3**
Scrivere una funzione che, dato un albero generico con etichette intere, aggiunge ad ogni nodo la somma delle etichette dei suoi figli. Si supponga che l'albero generico sia memorizzato figlio-fratello. Indicare la complessità.

```
void somma (Node* t) {
if (!t) return;
t->label= sommafigli(t->left);
somma (t->left);
somma (t->right);
}
int sommafigli (Node* t) {
if (!t) return 0;
return (t->label +sommafigli(t->right));
}
```

**Esercizio 4**
a) Descrivere l'algoritmo di Kruskal: a cosa serve, come è implementato e qual è la sua complessità.
b) Trovare l'albero di copertura di costo minimo del grafo in figura mediante l'algoritmo di Kruskal. Indicare l'ordine in cui vengono presi gli archi durante l'esecuzione dell'algoritmo.




b)

**Esercizio 5**

Indicare l'output del seguente programma c++. Per ogni linea di output, indicare quali costruttori vengono chiamati.

```
#include<iostream.h>
class uno {
protected:
        int a;
public:
        uno() { a=0; cout << "uno, a= " << a << endl; }
        uno(int z) { a=z; cout << "uno, a= " << a << endl; }
};
```

```
class due : public uno{
protected:
        int a;
public:
        due() { a=1; cout << "due, a= " << a << endl; }
         due(int y): uno(y){ a=1; cout << "due, a= " << a << endl; }
};
class tre: public due{
public:
        tre() {cout << "tre, a= " << a << endl;}
        tre(int x, int y  ): due(y) { a=x; cout << "tre, a= " << a << endl; }
          tre(int x): due(x) { a=x; cout << "tre, a= " << a << endl; }
};
void main(){
        tre obj1;
        tre obj2(4,6);
        tre obj3(5);
}
================
uno, a= 0                                   //   uno::uno()
due, a= 1                                    //   due:: due()
tre, a= 1                                    //   tre::tre()
uno, a= 6                                    //  uno::uno(6)
due, a= 1                                    //  due::due(6)
tre, a= 4                                    // tre::tre(4,6)
uno, a= 5                                     //  uno::uno(5)
due, a= 1                                     //  due::due(5)
tre, a= 5                                     //   tre::tre(5)
```