La complessità – Notazione *O*

- Useremo la notazione *O* (“o grande”, “big-o”)
  - Solitamente interessa dare un limite superiore (più comodo nelle analisi),
  - Molto informativa nel caso peggiore.
- Complessità asintotica: analisi basata sulla dimensionalità dell’input (caso peggiore: infinito)
- Ci sono casi in cui è possibile analizzare la variazione della complessità sulla base dei dati in input (non dimensionalità)
  - Caso migliore
  - Caso medio (spesso difficile da vedere! Servono nozioni statistiche sulla distribuzione/probabilità di dati)
  - Caso peggiore
- Esempio:

```
bool foo(int c, int p) {
    for (int i = 0; i < p; i++)
        cout << c << endl;
}
```

Complessità ? *O(p)*

```
bool bar(int c, int n) {
    for (int i = 0; i < 100; i++)
        cout << c*n << endl;
}
```

Complessità ? *O(1)*

Esercizio 1

Studiare la complessità asintotica di f

```
1  void f(int n)
2  {
3      int i=n;
4      while (i>=1)
5          {
6           for (int j=1; j<=n; j++)
7                   a++;
8           i--;
```

$O(N^2)$

Esercizio 2

Studiare la complessità asintotica di f

```
1  void f(int n)
2  {
3      int i=n;
4      while (i>=1)
5          {
6           for (int j=1; j<=n; j++)
7                  a++;
8           i=i-n;
9  }
```

O(N)

Esercizio 3

Studiare la complessità asintotica di f

```
1   int f (int n)
2   {
3       int a=n;
4       if (n<=500)
5           for (int i=1; i<=n; i++)
6               for (int j=1; j<=n;j++)
7                   a+=n;
8       else
9           for (int i=1; i<=n; i++)
10              a+=n;
11  return a; }
```

per n<=500 le istruzioni sono ripetute n^2 volte ma la complessità asintotica è ***O(n)***

Esercizio 4

Studiare la complessità nel caso migliore e nel caso peggiore della funzione elabora.

Caso migliore: $O(N)$

Caso peggiore: $O(N^3)$

```
1  bool elabora(int M[][N], int V[N]) {
2      bool b = false;
3      int i = 1;
4      while (i <= N && !b) {
5          if (somma(M) || V[i-1] == 0)
6              b = true;
7          i = i+2;
8      }
9      return b;
10 }
11
12 bool somma(int M[][N]) {
13     int s=0;
14     for (int i=0; i<N; i++)
15         if (M[i][i]!=0)
16             for (int j=0; j<N; j++)
17                 s += M[i][j];
18     if (s > 0)
19         return true;
20     else
21         return false;
22 }
```

Esercizio 5

Studiare la complessità nel caso migliore e nel caso peggiore della funzione f.

Caso migliore: *O(1)*

Caso peggiore: *O(N*log(N))*

```
1  int f(int V1[N], int V2[N]) {
2      bool b = false;
3      int i = 1;
4      while (!b && i < N) {
5          b = g(V1[i], V2) && g(V2[i], V1);
6          i = i * 2;
7      }
8      return i;
9  }
10
11 bool g(int val, int &V[N]) {
12     bool b = false;
13     for (int i=0; i < N && !b; i++)
14         if (val == V[i])
15             b = true;
16     return b;
17 }
```

Esercizio 6

Determinare la complessità del seguente programma. Studiare sia il caso peggiore che il ca migliore.

Caso migliore: $O(N^2)$

Caso peggiore: $O(N^3)$

```
1   Const int n
2
3   int costruisciD (int C[n][n], int V[n], int& D[n][n+1]);
4   {   for (int i=0; i<n; i++)
5           D[i][0]=V[i];
6       int nextCol=1;
7       for (int j=1; j<n; j++)
8           if (ContieneMultiplo(C,V,j))
9           {   for (int x=0; x<n; x++)
10                  D[x][nextCol]=C[x][j];
11              nextCol++;
12          }
13      return nextCol;
14  }
15
16  bool ContieneMultiplo(int C[n][n], int V[n], int col)
17  {   bool trovato=false;
18      for (int x=0; (x<n) && !trovato; x++)
19          for (int y=0; (y<n) && !trovato; y++)
20              if (C[x][col]%V[y]==0)
21                  trovato=true;
22      return trovato;
23  }
```