UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI MATEMATICA E INFORMATICA

(Introduzione) AND (Complessità)

Esercitazione del 11/03/2021

Francesco Cauteruccio, Ph.D.
cauteruccio@mat.unical.it
francescocauteruccio.info

(still) Inside the Vault EDITION

Who am I?

- Francesco Cauteruccio
  - Ph.D. in Computer Science
  - Ricercatore @ DEMACS

- Contatti
  - cauteruccio@mat.unical.it
  - https://francescocauteruccio.info

- Ricevimento?
  - No orario fisso
  - Inviatemi una mail e/o scrivetemi su Teams

Link utilissimi

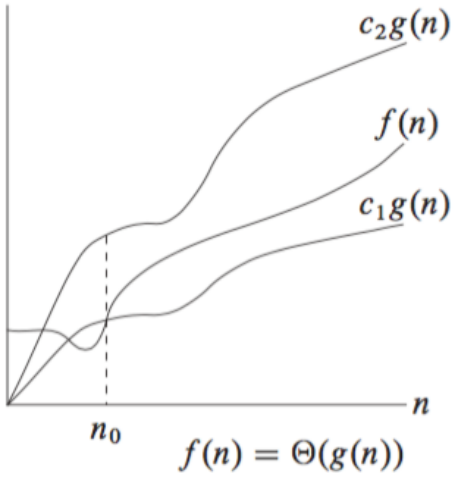
- Pagina del corso
  - https://www.mat.unical.it/terracina/asd/
  - Tutto il materiale usato a lezione.

- Prototypes ASD
  - http://prototypes.mat.unical.it/asd/
  - Diversi esercizi di programmazione (registratevi!)
  - Conterrà tutto il materiale che useremo ad esercitazione,
  - Contiene **tutte le tracce (sia modalità laboratorio che online) degli appelli precedenti** (dal 2017).

- Gruppo Ufficiale Facebook: Algoritmi e Strutture Dati - CdL Informatica, UNICAL
  - https://www.facebook.com/groups/asd.demacs.unical/
  - Avvisi vari, notizie, link utili.

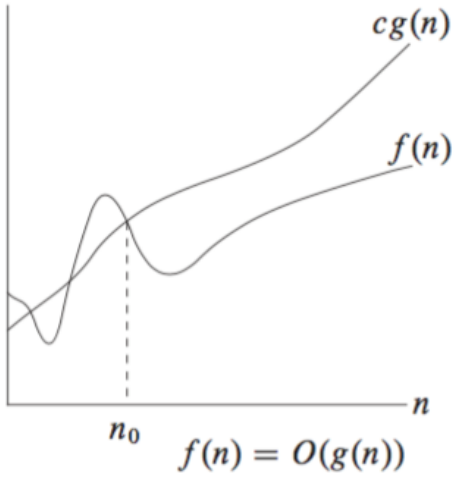
La complessità – Notazioni asintotiche

- Usiamo le notazioni asintotiche per caratterizzare il *running time* di un algoritmo
  - Ci interessiamo solo del comportamento asintotico
  - Per il running time, generalmente ci interessa il *caso migliore* e il *caso peggiore*.
- Esistono tre diverse notazioni asintotiche: Θ, O, Ω



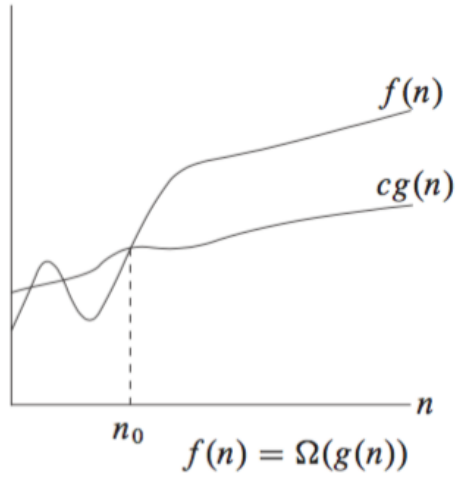
Delimitazione stretta

"f(n) cresce esattamente come g(n)"



Delimitazione superiore

"f(n) cresce al più come g(n)"



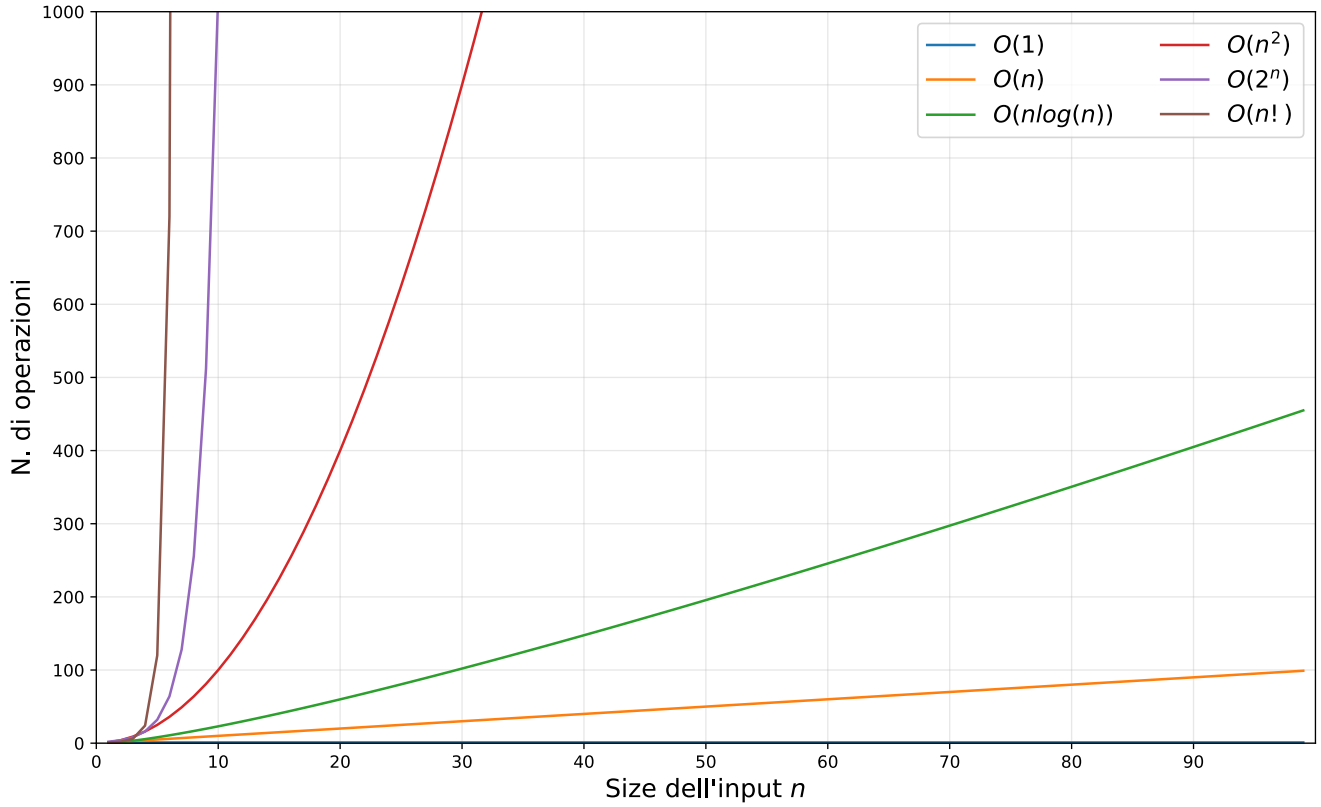
Delimitazione inferiore

"f(n) cresce almeno come g(n)"

Figura da Cormen, Thomas H., et al. *Introduction to algorithms*. MIT press, 2009.

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.



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.

- Esempio:

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

Complessità?

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

Complessità?

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.

- 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?

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.

- 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 nel caso migliore e nel caso peggiore della funzione `elabora`.

```
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 1

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

Caso migliore: *O(N)*

```
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 1

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 2

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

```
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 2

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

Caso migliore: *O(1)*

```
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 2

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 3

Studiare la complessità nel caso migliore e nel caso peggiore della funzione `elabora` sapendo che:

- la funzione `f1(M)` ha complessità *O(N)* e può terminare restituendo indifferentemente *true* o *false*
- la funzione `f2(V)` ha complessità $O(N^2)$ e può terminare restituendo indifferentemente *true* o *false*

```
1  void elabora(int M[][N], int V[N]) {
2      bool b = false;
3      int j, i = 0;
4      char car;
5      while (i<N && !b) {
6          cin >> car;
7          switch (car) {
8              case 'a':
9                  if (f2(V))
10                     for (j=0; j<N; j++)
11                         cout << M[i][j];
12                 break;
13             case 'b':
14                 if (f1(M) && f2(V))
15                     b = true;
16                 else
17                     cout << V[i];
18                 break;
19             default:
20                 cout << V[i];
21         } // fine switch
22
23         for (int x=0; x<N; x++)
24            if (M[x][i] != V[x])
25                b = true;
26         i++;
27     } // fine while
28 }
```

Esercizio 3

Studiare la complessità nel caso migliore e nel caso peggiore della funzione `elabora` sapendo che:

- la funzione `f1(M)` ha complessità *O(N)* e può terminare restituendo indifferentemente *true* o *false*
- la funzione `f2(V)` ha complessità $O(N^2)$ e può terminare restituendo indifferentemente *true* o *false*

Caso migliore: *O(N)*

```
1  void elabora(int M[][N], int V[N]) {
2      bool b = false;
3      int j, i = 0;
4      char car;
5      while (i<N && !b) {
6          cin >> car;
7          switch (car) {
8              case 'a':
9                  if (f2(V))
10                     for (j=0; j<N; j++)
11                         cout << M[i][j];
12                 break;
13             case 'b':
14                 if (f1(M) && f2(V))
15                     b = true;
16                 else
17                     cout << V[i];
18                 break;
19             default:
20                 cout << V[i];
21         } // fine switch
22
23         for (int x=0; x<N; x++)
24            if (M[x][i] != V[x])
25                b = true;
26         i++;
27     } // fine while
28 }
```

Esercizio 3

Studiare la complessità nel caso migliore e nel caso peggiore della funzione `elabora` sapendo che:

- la funzione `f1(M)` ha complessità *O(N)* e può terminare restituendo indifferentemente *true* o *false*
- la funzione `f2(V)` ha complessità $O(N^2)$ e può terminare restituendo indifferentemente *true* o *false*

Caso migliore: *O(N)*

Caso peggiore: $O(N^3)$

```
1  void elabora(int M[][N], int V[N]) {
2      bool b = false;
3      int j, i = 0;
4      char car;
5      while (i<N && !b) {
6          cin >> car;
7          switch (car) {
8              case 'a':
9                  if (f2(V))
10                     for (j=0; j<N; j++)
11                         cout << M[i][j];
12                 break;
13             case 'b':
14                 if (f1(M) && f2(V))
15                     b = true;
16                 else
17                     cout << V[i];
18                 break;
19             default:
20                 cout << V[i];
21         } // fine switch
22
23         for (int x=0; x<N; x++)
24            if (M[x][i] != V[x])
25                b = true;
26         i++;
27     } // fine while
28 }
```

Esercizio 4 - bonus

Studiare la complessità nel caso migliore e nel caso peggiore della funzione `foo`.

```
1  void foo(int M[][N], int V[N]) {
2      bool b = false;
3      for (int i = 0; i < N && !b; i++) {
4          for (int h=0; h<N; h++)
5              cout << V[h];
6          if (bar(M,V)) {
7              int k = 0;
8              while (k < N) {
9                  for (int j = 0; j < N; j++)
10                     if (M[i,j] <= V[k])
11                         k = N;
12                 k++;
13             }
14         } else
15             for ( int j=0; j<N; j++ )
16                 if (M[i,j] > V[j])
17                     b = true;
18     }
19 }
```

```
20 bool bar(int M[][N], int V[N]) {
21       int k=0;
22       while (k < N) {
23         for (int j = 0; j < N; j++)
24             if (M[k][j] % 2 == 0)
25                 return false;
26         if (V[k] < 0)
27             return false;
28         else
29             if (V[k] <= M[k][k])
30                 return false;
31         k++;
32       }
33       return true;
34 }
```