



# PROGRAMARE PROCEDURALĂ

Bogdan Alexe

[bogdan.alexe@fmi.unibuc.ro](mailto:bogdan.alexe@fmi.unibuc.ro)

Secția Informatică, anul I,

2016-2017

Cursul 2

# Recapitulare – cursul trecut

1. Algoritmi.
2. Limbaje de programare.
3. Introducere în limbajul C.

# Programa cursului

## □ Introducere

- Algoritmi.
  - Limbaje de programare.
- Introducere în limbajul C. Structura unui program C.  
**Complexitatea algoritmilor.**

## □ Fundamentele limbajului C

- Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
- Instrucțiuni de control
- Directive de preprocesare. Macrodefiniții.
- Funcții de citire/scriere.
- Etapele realizării unui program C.

## □ Tipuri deriveate de date

- Tablouri. Siruri de caractere.
- Structuri, uniuni, câmpuri de biți, enumerări.
- Pointeri.

## □ Funcții (1)

- Declarație și definire. Apel. Metode de transmisie a parametrilor.
- Pointeri la funcții.

## □ Tablouri și pointeri

- Legătura dintre tablouri și pointeri
- Aritmetică a pointerilor
- Alocarea dinamică a memoriei
- Clase de memorare

## □ Siruri de caractere

- Funcții specifice de manipulare.

## □ Fișiere text și fișiere binare

- Funcții specifice de manipulare.

## □ Structuri de date complexe și autoreferite

- Definire și utilizare

## □ Funcții (2)

- Funcții cu număr variabil de argumente.
- Preluarea argumentelor funcției main din linia de comandă.
- Programare generică.

## □ Recursivitate

# Cuprinsul cursului de azi

1. Introducere în limbajul C. Structura unui program C.
2. Complexitatea algoritmilor.
3. Tipuri de date fundamentale.

# Caracteristici ale limbajului C

- ❑ limbaj procedural, structurat, compilat, de nivel de mijloc, scurt
- ❑ limbaj procedural, structurat
  - ❑ instrucțiuni specificate sub forma unor comenzi grupate într-o ierarhie de subprograme (denumite funcții) și care pot forma module
- ❑ limbaj compilat
  - ❑ compilatorul transformă instrucțiunile limbajului C în limbaj mașină
- ❑ limbaj de nivel de mijloc
  - ❑ permite accesul la date și comenzi aflate aproape de nivelul fizic folosind o sintaxă specifică limbajelor de nivel înalt
- ❑ limbaj scurt
  - ❑ număr redus de cuvinte cheie
  - ❑ multe funcționalități nu sunt incluse în limbajul de bază ci necesită includerea unor biblioteci standard

# Cuvinte cheie

## C89 = ANSI C : 32 de cuvinte cheie

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

## C99: ANSI C + alte 5 cuvinte cheie

\_Bool \_Complex \_Imaginary inline restrict

# Structura generală a unui program C

- modul principal (funcția main)
- zero, unul sau mai multe module (funcții/proceduri) care comunică între ele și/sau cu modulul principal prin intermediul parametrilor și/sau a unor variabile globale
- unitatea de program cea mai mică și care conține cod este funcția/procedura și conține:
  - partea de declarații/definiții;
  - partea imperativă (comenzile care se vor executa);

# Primul program C

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Primul program scris in C\n");
6     return 0;
7 }
```

# Primul program C explicat

```
1 #include <stdio.h>
```

Directivă de preprocesare pentru includerea bibliotecii standard de i/o

Antetul funcției principale

```
3 int main()
4 {
5     printf("Primul program scris in C\n");
6     return 0;
7 }
```

Funcția principală

Corpul funcției

## Observații:

- `main` nu este cuvânt cheie în limbajul C, îl utilizăm pentru numirea funcției principale;
- `printf` nu este cuvânt cheie, este funcție de bibliotecă (`print` (afișare) + `f` (format));
- C este case sensitive, se face diferență între litere mici și mari;
- toate cuvintele cheie se scriu cu litere mici;
- instrucțiunile se termină cu `caracterul ;` (punct și virgulă);
- mai multe instrucțiuni pot fi scrise pe aceeași linie;
- spațiile ajută la organizarea codului.

# Structura unui program C simplu

*directive de procesare*

```
int main()
{
    instrucțiuni
}
```

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Primul program scris in C\n");
6 }
7 }
```

- Directive de procesare
  - directive de definiție: #define N 10
  - directive de includere a bibliotecilor: #include <stdio.h>
  - directive de compilare condiționată: #if, #ifdef, ...
  - alte directive (vorbim în cursurile următoare)
- Funcții
  - grupări de instrucțiuni sub un nume;
  - returnează o valoare sau se rezumă la efectul produs;
  - funcții scrise de programator vs. funcții furnizate de biblioteci;
  - programul poate conține mai multe funcții;
    - **main** este obligatoriu;
  - antetul și corpul funcției.

# Structura unui program C simplu

*directive de procesare*

```
int main()
{
    instrucțiuni
}
```

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Primul program scris in C\n");
6 }
7 }
```

## □ Instrucțiuni

- formează corpul funcțiilor
  - exprimate sub formă de comenzi
- 5 tipuri de instrucțiuni:
  - instrucțiunea declarație;
  - instrucțiunea atribuire;
  - instrucțiunea apel de funcție;
  - instrucțiuni de control;
  - instrucțiunea vidă;
- toate instrucțiunile (cu excepția celor compuse) se termină cu caracterul ";"
  - caracterul ; nu are doar rol de separator de instrucțiuni ci instrucțiunile încorporează caracterul ; ca ultim caracter
  - omiterea caracterului ; reprezintă eroare de sintaxă

# Structura unui program C complex

*comentarii*

*directive de preprocessare*

*declarații și definiții globale*

```
int main()
```

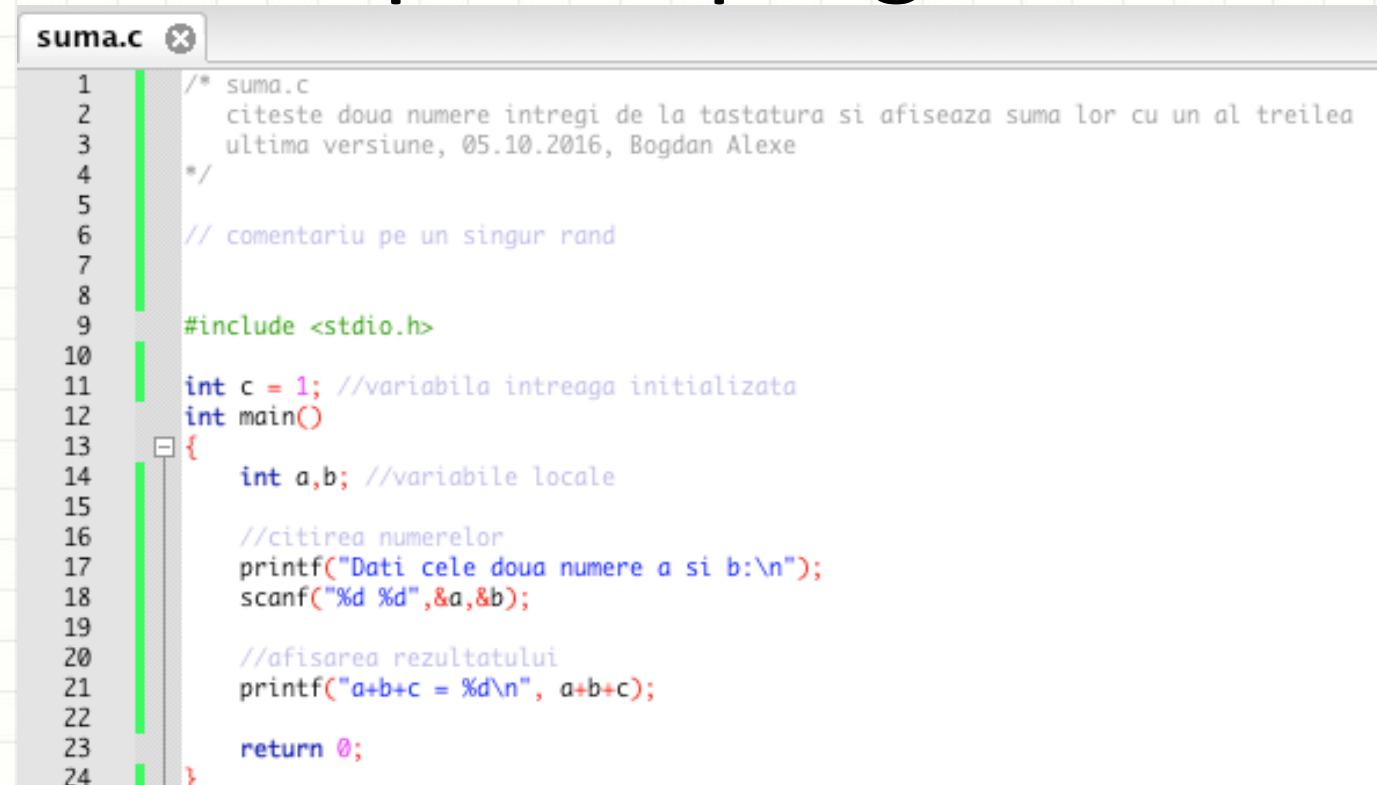
```
{
```

*declarații și definiții locale*

*instrucțiuni*

```
}
```

# Exemplu de program C complex



```
suma.c
1  /* suma.c
2   citeste doua numere intregi de la tastatura si afiseaza suma lor cu un al treilea
3   ultima versiune, 05.10.2016, Bogdan Alexe
4   */
5
6  // comentariu pe un singur rand
7
8
9  #include <stdio.h>
10
11 int c = 1; //variabila intreaga initializata
12 int main()
13 {
14     int a,b; //variabile locale
15
16     //citirea numerelor
17     printf("Dati cele doua numere a si b:\n");
18     scanf("%d %d",&a,&b);
19
20     //afisarea rezultatului
21     printf("a+b+c = %d\n", a+b+c);
22
23     return 0;
24 }
```

## □ Comentariile

- formă de documentare a codului sursă, sunt ignore de compilator
- 2 tipuri de comentariu:
  - începe cu /\* și se termină cu \*/: se pot extinde pe mai multe linii, nu se pot imbrica, sunt utile pentru inserarea unor explicații mai lungi
  - începe cu // și se termină la sfârșitul liniei: utile pentru comentariile inserate pe marginea codului (apare în C99, nu este în C89)

# Cuprinsul cursului de azi

1. Introducere în limbajul C. Structura unui program C.
2. Complexitatea algoritmilor.
3. Tipuri de date fundamentale.

# Complexitatea algoritmilor

- definită de resursele (timp de execuție, spațiu de memorie, număr de CPU utilizate pentru calcul paralel) de care are nevoie algoritmul pentru a produce datele de ieșire;
- ne va interesa numai studiul complexității timp (timp de execuție)
- observație: considerăm că algoritmii se execută secvențial (nu tratăm cazul în care operațiile se pot realiza paralel).

# Complexitatea algoritmilor

A1: algoritmul lui Euclid cu resturi

```
4
5     int euclidResturi(int a, int b)
6     {
7         int rest;
8         while(b>0)
9         {
10            rest = a%b;
11            a = b;
12            b = rest;
13        }
14    return a;
15 }
```

A2: algoritmul lui Euclid cu scăderi

```
17
18     int euclidScaderi(int a, int b)
19     {
20         while (a!=b)
21         {
22             if (a>b)
23                 a = a-b;
24             else
25                 b = b-a;
26         }
27     return a;
-- }
```

A3: algoritm brut

```
27
28     int cmmdcBrut(int a, int b)
29     {
30         int min = (a>b)? b : a;
31         int i, cmmdc = 1;
32         for (i = 2; i<= min; i++)
33         {
34             if((a%i==0) && (b%i == 0))
35                 cmmdc = i;
36         }
37     return cmmdc;
38 }
```

Câte operații elementare  
(atribuiri, comparații, operații  
aritmetice, indexări, etc)  
realizează fiecare algoritm  
dacă  $a = 99$  și  $b = 97$ ?

# Complexitatea algoritmilor

- **T(n)** - timp de rulare al unui algoritm măsurat în număr de operații elementare = instrucțiuni ale procesorului (comparații, atribuiri, operații aritmetice, indexări, etc)
- **n** reprezintă dimensiunea datelor de intrare
- analiza lui **T(n)** se realizează de obicei în 3 cazuri:
  - cazul cel mai defavorabil
  - cazul cel mai favorabil
  - cazul mediu

# Cazul cel mai defavorabil

- caracterizăm complexitatea unui algoritm = complexitatea algoritmului în cazul cel mai defavorabil
- timpul mediu de execuție este de multe ori apropiat de timpul de executare în cazul cel mai defavorabil
- oferă o limită superioară a timpului de execuție (avem certitudinea că execuția algoritmului nu va dura mai mult)

# Numărarea operațiilor elementare realizate de un algoritm în cazul cel mai defavorabil

Aflarea maximului unui vector de dimensiune  $n$

maxim = v[0]; → 2 (o indexare + o atribuire)

for (i=1; i<n; i++) → 2n (o atribuire + n comparații + (n-1)incrementări)

if (maxim < v[i]) → 2(n-1) (n-1 indexări + n-1 comparații)

maxim = v[i]; → 2(n-1) (n-1 indexări + n-1 atribuirii)

return maxim → 1 (o întoarcere a rezultatului)

$$T(n) = 6n - 1 \text{ operații elementare}$$

# Estimarea timpului de rulare al unui algoritm în cazul cel mai defavorabil

- aflarea maximului unui vector de dimensiune  $n$  necesită executarea a  $6n-1$  operații elementare în cazul cel mai defavorabil
- definim  $a$  și  $b$  ca fiind:
  - $a$  = timpul necesar pentru executarea celei mai rapide operații elementare
  - $b$  = timpul necesar pentru executarea celei mai încete operații elementare
  - $a$  și  $b$  sunt CONSTANTE pe o mașină
- atunci avem:  $a(6n-1) \leq T(n) \leq b(6n-1)$
- timpul de rulare  $T(n)$  al algoritmului în cazul cel mai defavorabil este mărginit inferior și superior de funcții liniare

# Rata de creștere a timpului de rulare

- dacă modificăm hardware-ul (avem un procesor mai rapid/încet):
  - a și b se modifică (devin mai mici/mari)
  - $T(n)$  se modifică cu un factor CONSTANT
  - rata de creștere liniară a lui  $T(n)$  în raport cu n NU SE MODIFICĂ
- rata de creștere liniară a timpului de rulare  $T(n)$  este o proprietate intrinsecă a algoritmului de afilare a maximului unui vector.

# Comportamentul asimptotic al timpului de rulare

- avem 2 algoritmi A și B care rezolvă aceeași problemă;
- pentru date de intrare de dimensiune  $n$  avem:
  - complexitatea timp a lui A este  $T_A(n) = 5000n$
  - complexitatea timp a lui B este  $T_B(n) = \lceil 1.1^n \rceil$
- care este comportamentul asimptotic ( $n \rightarrow \infty$ ) al algoritmilor?

$n$	$T_A(n)$	$T_B(n)$
10	50,000	3
100	500,000	13781
1,000	5,000,000	$2.5 \cdot 10^{41}$
1,000,000	$5 \cdot 10^9$	$4.8 \cdot 10^{41392}$

# Comportamentul asimptotic al timpului de rulare

- algoritmul B nu poate fi folosit pentru date de intrare de dimensiune mare. Totuși putem folosi algoritmul A.
- ceea ce este important este creșterea asimptotică a timpului de rulare  $T(n)$  a unui algoritm.
- creșterea asimptotică este o măsură bună pentru a compara algoritmii și a decide care algoritm este mai performant

n	$T_A(n)$	$T_B(n)$
10	50,000	3
100	500,000	13781
1,000	5,000,000	$2.5 \cdot 10^{41}$
1,000,000	$5 \cdot 10^9$	$4.8 \cdot 10^{41392}$

# Creșterea asimptotică pentru câteva funcții

$n$	$\log n$	$n$	$n \log n$	$n^2$	$n^3$	$2^n$
4	2	4	8	16	64	16
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536
32	5	32	160	1,024	32,768	4,294,967,296
64	6	64	384	4,094	262,144	$1.84 * 10^{19}$
128	7	128	896	16,384	2,097,152	$3.40 * 10^{38}$
256	8	256	2,048	65,536	16,777,216	$1.15 * 10^{77}$
512	9	512	4,608	262,144	134,217,728	$1.34 * 10^{154}$
1024	10	1,024	10,240	1,048,576	1,073,741,824	$1.79 * 10^{308}$

# Notația $\mathcal{O}$

- fie funcțiile  $f, g : \mathbb{N} \rightarrow \mathbb{R}_+$ . Spunem că funcția  $f(n)$  este  $O(g(n))$  (citim “ $f$  este de clasă  $\mathcal{O}$  de  $g$ ” sau “ $f$  este dominată de  $g$ ”) dacă există constantele pozitive  $c$  și  $n_0$  astfel încât:

$$f(n) \leq cg(n) \text{ pentru orice } n \geq n_0$$

**Exemplu:**  $2n + 10$  este  $O(n)$

$$2n + 10 \leq cn, \text{ pentru } n \geq n_0$$

$$(c - 2)n \geq 10$$

$$n \geq 10/(c - 2)$$

$$\text{alegem } c = 3 \text{ și } n_0 = 10$$

**Exemplu:**  $n^2$  nu este  $O(n)$

$$n^2 \leq cn, \text{ pentru } n \geq n_0$$

$$n \leq c$$

nu putem alege constanta  $c$

$$n^2 \text{ este } O(n^2)$$

# Notația $\mathcal{O}$ - alte exemple

**Exemplu:**  $7n - 2$  este  $O(n)$

$$7n - 2 \leq cn, \text{ pentru } n \geq n_0$$

$$(c - 7)n \geq -2$$

$$\text{alegem } c = 7 \text{ și } n_0 = 1$$

**Exemplu:**  $3n^3 + 20n + 5$  este  $O(n^3)$

$$3n^3 + 20n^2 + 5 \leq cn^3, \text{ pentru } n \geq n_0$$

$$(c - 3)n^3 \geq 20n^2 + 5$$

$$\text{alegem } c = 4 \text{ și } n_0 = 21$$

**Exemplu:**  $3\log n + 5$  este  $O(\log n)$

$$3\log n + 5 \leq c \log n, \text{ pentru } n \geq n_0$$

$$(c - 3)\log n \geq 5$$

$$\text{alegem } c = 8 \text{ și } n_0 = 2$$

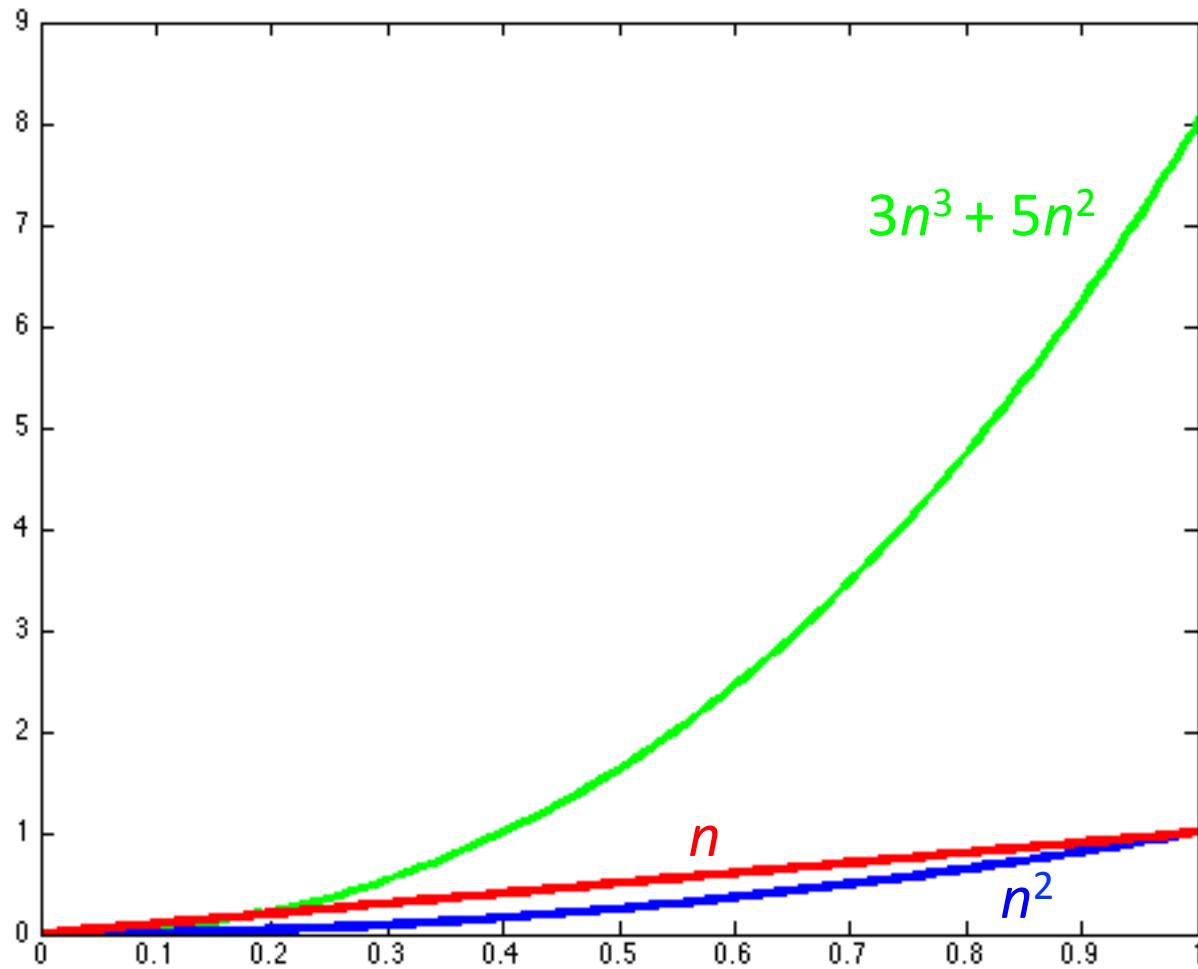
# Legătura dintre notația O și creșterea asimptotică a funcțiilor

- notația O oferă o margine superioară a creșterii asimptotice a unei funcții;
- când spunem “ $f(n)$  este  $O(g(n))$ ” înseamnă că rata de creștere a lui  $f(n)$  nu este mai mare decât rata de creștere a lui  $g(n)$
- putem folosi notația O pentru a ordona funcțiile pe baza ratei lor de creștere asimptotică

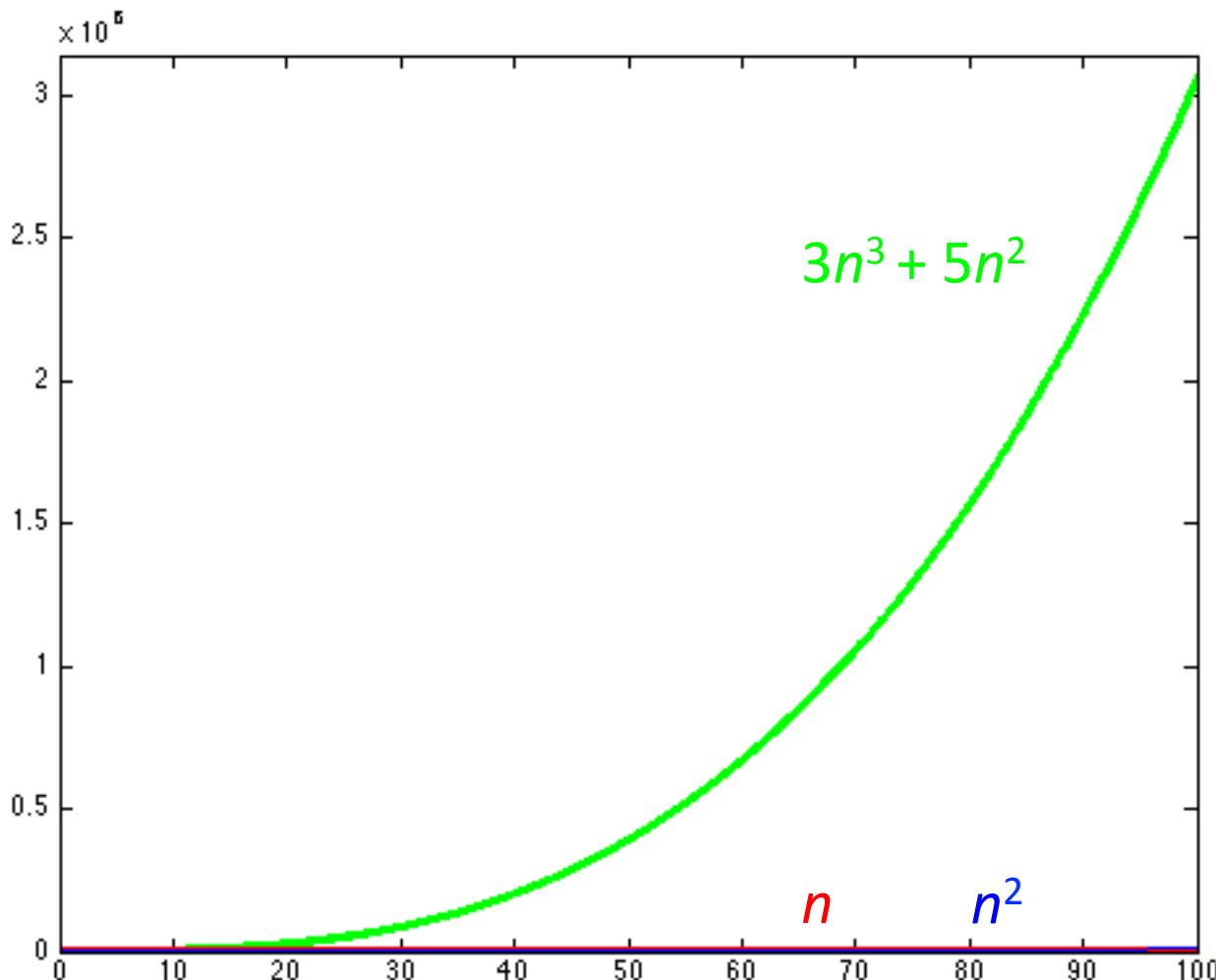
# Creșterea asimptotică – reprezentare grafică

- creșterea asimptotică caracterizează comportamentul funcțiilor pentru **valori mari ale lui  $n$** .
- exemplu: **termenul dominant** din  $3n^3 + 5n^2$  este  **$n^3$** .
- pe măsură ce  **$n$  devine** din ce în ce mai mare, **ceilalți termeni devin nesemnificativi**
- plotăm graficul funcției  $3n^3 + 5n^2$  și comparăm cu alte funcții ( $n^2$  și  $n$ )
- plotăm graficul funcției  $3n^3 + 5n^2$  și comparăm cu alte funcții ( $n^3$  și  $5n^3$ )

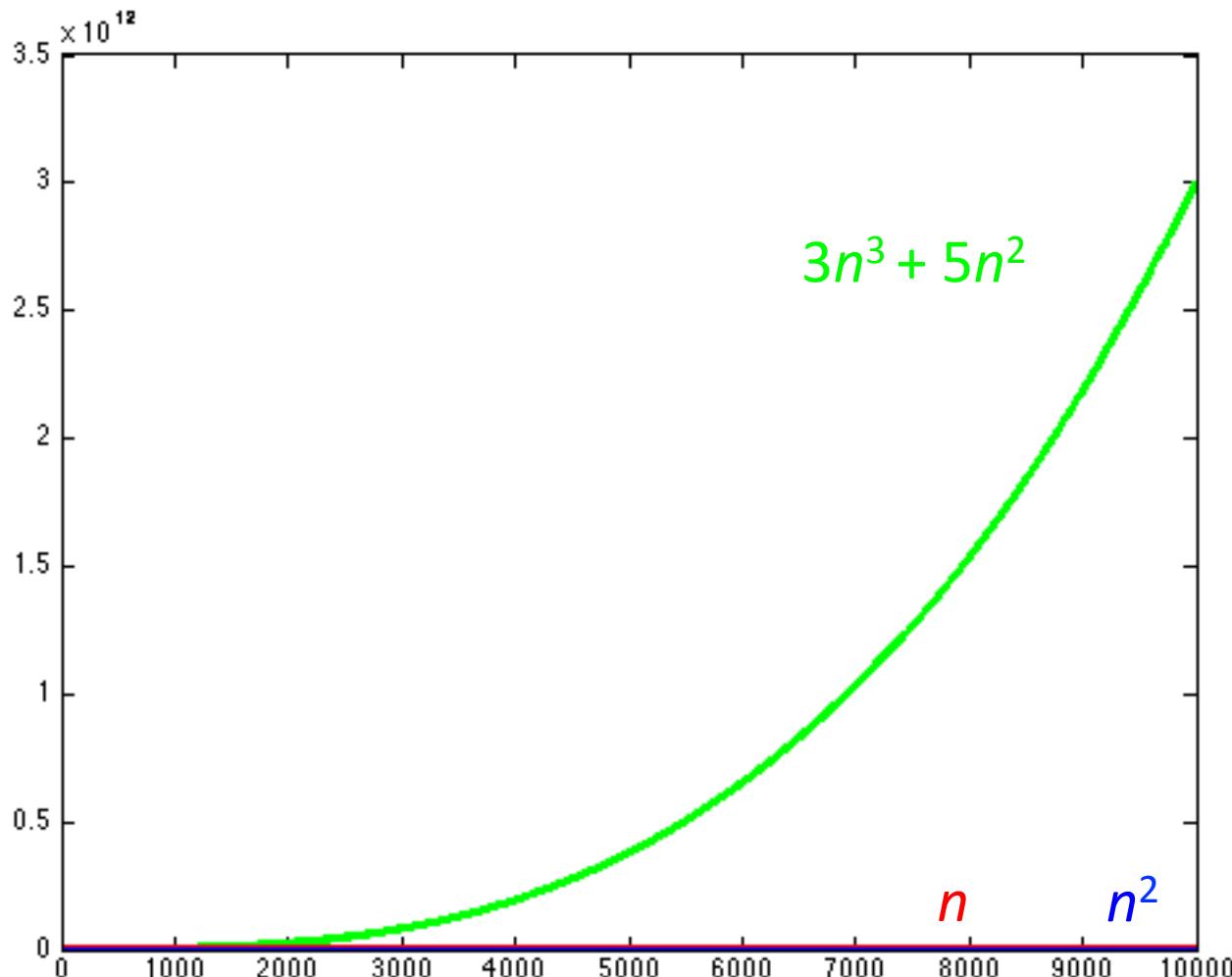
# Creșterea asimptotică – reprezentare grafică



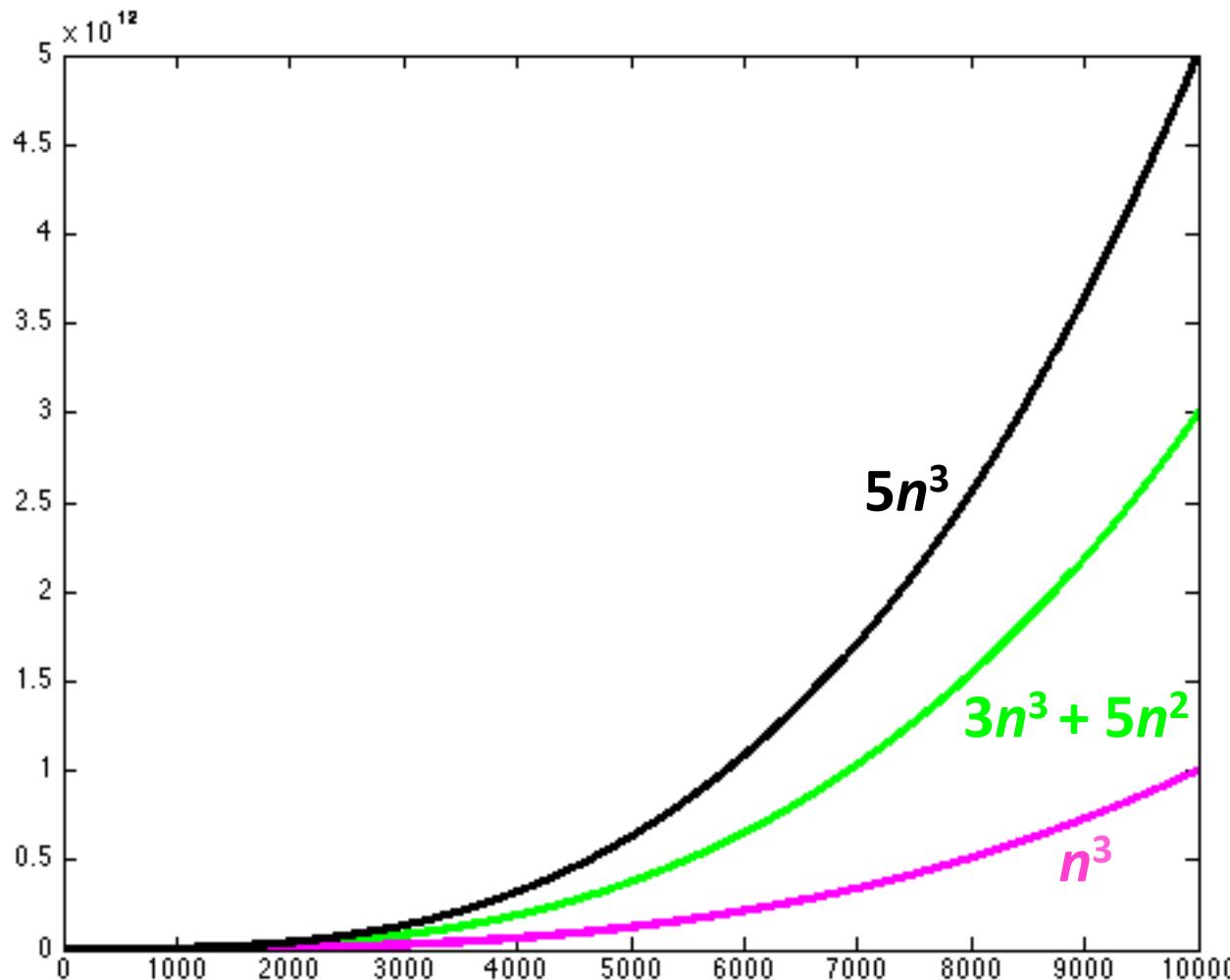
# Creșterea asimptotică – reprezentare grafică



# Creșterea asimptotică – reprezentare grafică



# Creșterea asimptotică – reprezentare grafică



$3n^3 + 5n^2$  este  $O(n^3)$

# Reguli pentru stabilirea clasei $O$ a unei funcții

- dacă  $f(n)$  este un polinom de grad  $d$ , atunci  $f(n)$  este  $O(n^d)$ , adică:
  1. eliminăm termenii de grad inferior ( $n^{d-1}, n^{d-2}, \dots$ )
  2. eliminăm factorii constanți
  3. păstrăm numai partea dominantă
- folosim cea mai mică posibilă clasă de funcții  $O$ 
  - spunem “ $2n$  este  $O(n)$ ” în loc de “ $2n$  este  $O(n^2)$ ”

Ordonați funcțiile pe baza creșterii lor asimptotice

$$f_1(n) = 2^n + 10$$

$$f_6(n) = 10 \cdot \log_2(n)$$

$$f_2(n) = n^2 + 7$$

$$f_7(n) = n \cdot \sqrt{n}$$

$$f_3(n) = 2 \cdot n^2$$

$$f_8(n) = n^4$$

$$f_4(n) = 3 \cdot 2^n + 5$$

$$f_9(n) = 2 \cdot \ln(n)$$

$$f_5(n) = 4^n$$

$$f_{10}(n) = 4 \cdot n + 5$$

Ordonați funcțiile pe baza creșterii lor asimptotice

$$f_6(n) = 10 \cdot \log_2(n) \quad f_9(n) = 2 \cdot \ln(n)$$

$$f_{10}(n) = 4 \cdot n + 5$$

$$f_7(n) = n \cdot \sqrt{n}$$

$$f_2(n) = n^2 + 7 \quad f_3(n) = 2 \cdot n^2$$

$$f_8(n) = n^4$$

$$f_1(n) = 2^n + 10 \quad f_4(n) = 3 \cdot 2^n + 5$$

$$f_5(n) = 4^n$$

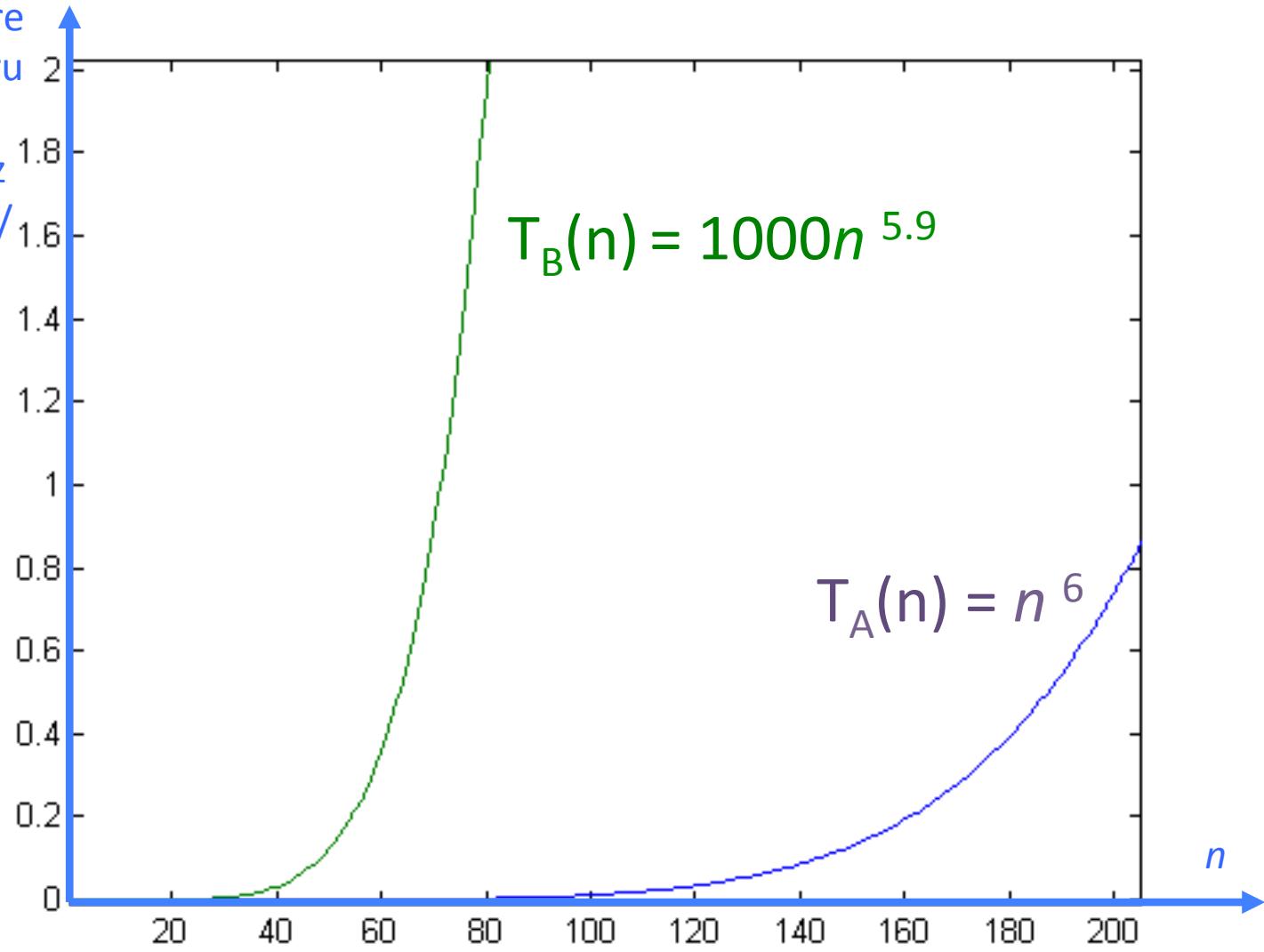


# Ce ascunde notația $O$ ?

- notația  $O$  furnizează informație despre creșterea asimptotică a algoritmilor. Pe baza acestei informații se poate decide care algoritm e mai rapid. În practică această decizie nu e întotdeauna corectă.
- considerăm doi algoritmi A și B:
  - timpi de rulare:  $T_A(n) = n^6$  și  $T_B(n) = 1000n^{5.9}$
  - asimptotic B e mai rapid ca A (cel puțin în teorie)
  - ce se întâmplă în practică?

# Ce ascunde notația $O$ ?

Timp de rulare  
(în zile) pentru  
un computer  
care realizează  
 $[10^9$  operații/  
secundă



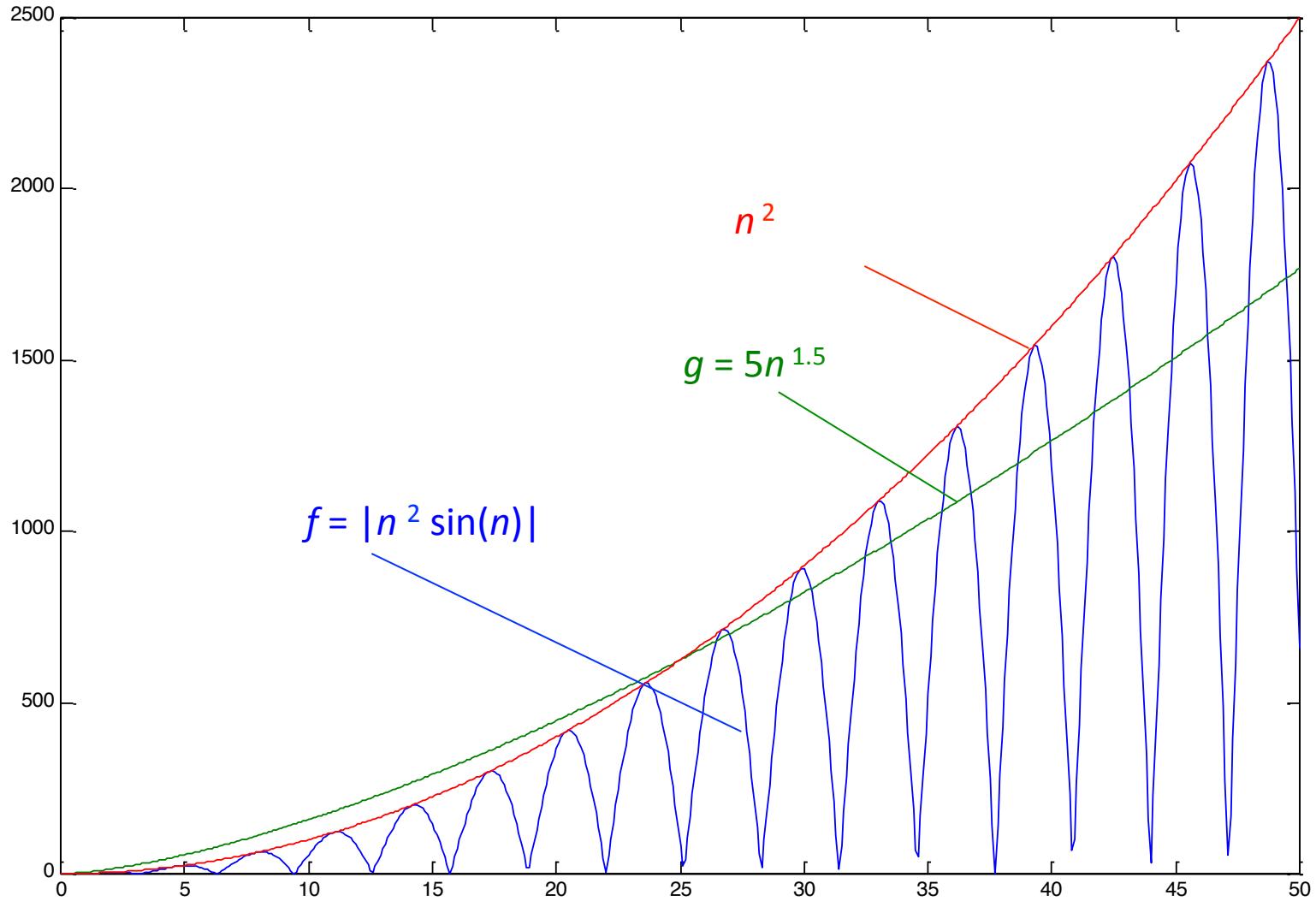
# Ce ascunde notația $O$ ?

- În practică,  $n^6$  îl “depășește” pe  $1000n^{5.9}$  când  $1000n^{5.9} \leq n^6$ ,  
$$1000 \leq n^{0.1},$$
$$n \geq 1000^{10} = 10^{30} \text{ operații}$$
$$= 10^{30}/10^9 = 10^{21} \text{ secunde}$$
$$(1 \text{ an} = 365 \times 24 \times 3600 = 31536000 \approx 3 \times 10^7 \text{ sec.})$$
$$\approx 10^{21}/(3 \times 10^7) \approx 3 \times 10^{13} \text{ ani}$$
- după  $3 \times 10^{13}$  ani, algoritmul B devine mai performant decât algoritmul A!!!

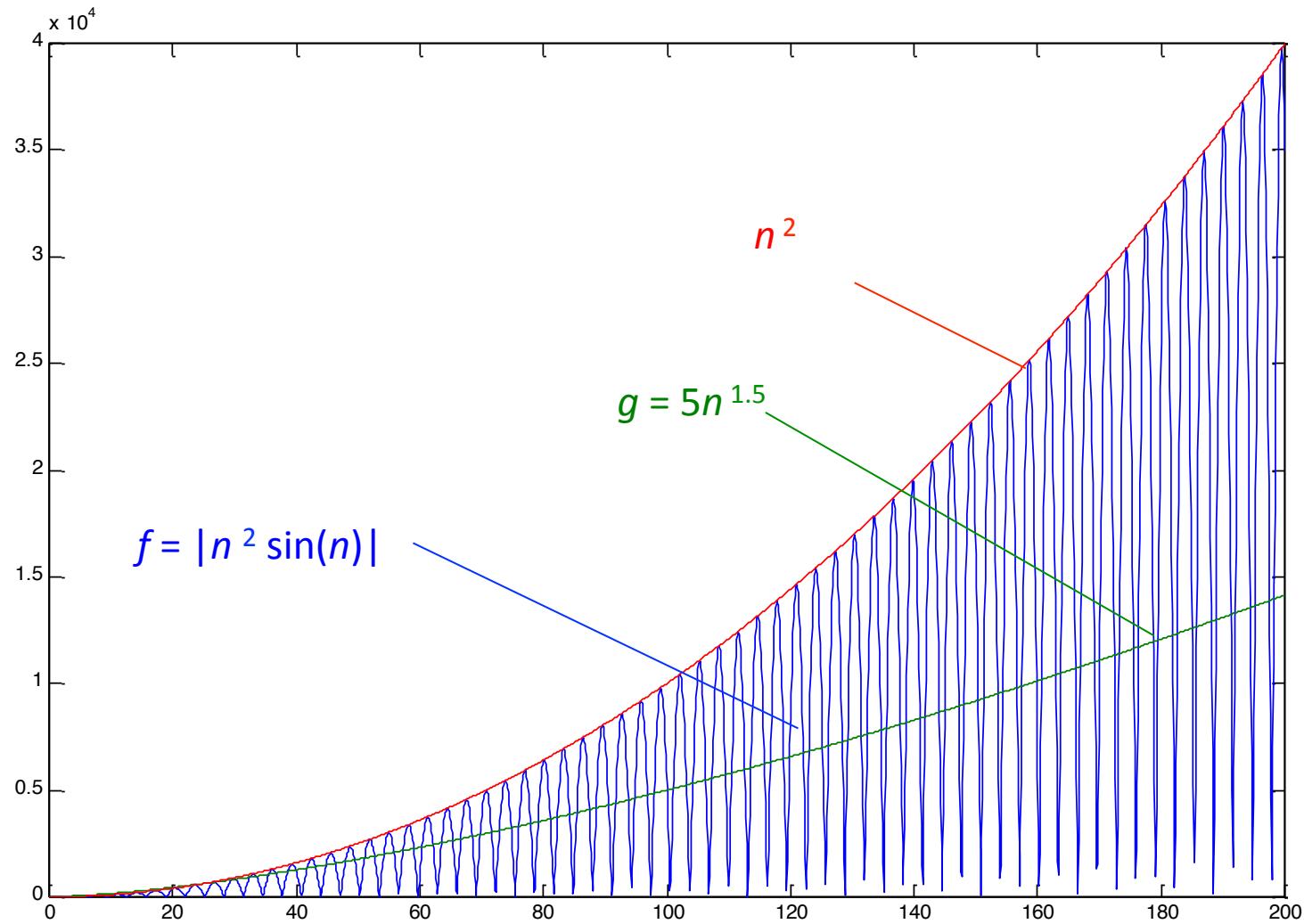
# Functii incomparabile

- pentru două funcții  $f(n)$  și  $g(n)$  nu întotdeauna avem  $f(n) = O(g(n))$  sau  $g(n) = O(f(n))$
- $f$  și  $g$  se numesc **asimptotic incomparabile**
- exemplu:  
 $f(n) = |n^2 \sin(n)|$  și  $g(n) = 5n^{1.5}$

# Functii incomparabile



# Functii incomparabile



# Căutarea unei valori într-un vector ordonat

```
int cautareBinara(int v[], int n, int val)
{
    int stanga = 0, dreapta = n - 1;
    int mijloc = (stanga + dreapta) / 2;

    while (stanga <= dreapta && val != v[mijloc])
    {
        if (val < v[mijloc])
            dreapta = mijloc - 1;
        else
            stanga = mijloc + 1;

        mijloc = floor((stanga + dreapta) / 2);
    }
    if (v[mijloc] == val)
        return mijloc;
    else
        return -1; // valoare nu se afla in vector
}
```

- căutarea binara – complexitate  $O(\log_2(n))$
- este corect codul scris?

# Cuprinsul cursului de azi

1. Introducere în limbajul C. Structura unui program C.
2. Complexitatea algoritmilor.
3. Tipuri de date fundamentale.

# Tipuri de date fundamentale

- ❑ programele manipulează date sub formă de numere, litere, cuvinte, etc.
- ❑ tipul de date specifică
  - ❑ natura datelor care pot fi stocate în variabilele de acel tip
  - ❑ necesarul de memorie
  - ❑ operațiile permise asupra acestor variabile
- ❑ În C89, limbajul C are **cinci categorii fundamentale** de tipuri de date: **int, char, double, float, void**
- ❑ C99 a introdus alte 3 tipurile de date:
  - ❑ **\_Bool** (true, false), de fapt valori întregi (0 = fals, altceva = adevărat)
  - ❑ **\_Complex** pentru numere complexe
  - ❑ **\_Imaginary** pentru numere imaginare

# Tipuri de date fundamentale

- În C89, limbajul C are **cinci categorii fundamentale** de tipuri de date: **int**, **char**, **double**, **float**, **void**
  - tipul **întreg** – **int**: poate reține valori întregi: 2, 0, -532
  - tipul **caracter** – **char**: poate reține un singur caracter sub forma codului elementelor din setul de caractere specific (codul ASCII), sau numere întregi mici
  - tipul **real** (numere în virgulă mobilă) – **simplă precizie** – **float**: pot reține numere care conțin parte fraționară: 4971.185, -0.72561, 2.000, 3.14
  - tipul **real** (numere în virgulă mobilă) **în dublă precizie** – **double**: pot reține valori reale în virgulă mobilă cu o precizie mai mare decât tipul float
  - tipul **void**: indică lipsa unui tip anume

# Tipuri de date fundamentale

- se pot crea noi tipuri de date prin combinarea celor de bază
- reprezentarea și spațiul ocupat în memorie de diferitele tipuri de date depind de:
  - platformă, sistem de operare și compilator
- limitele specifice unui sistem de calcul pot fi aflate din fișierele header [limits.h](#) și [float.h](#)
  - exemplu: [CHAR\\_MAX](#), [INT\\_MAX](#), [INT\\_MIN](#), [FLT\\_MAX](#), [DBL\\_MAX](#)
- pentru determinarea numărului de octeți ocupați de un anumit tip de date se folosește operatorul [sizeof](#)
  - 1 octet = 1 byte = 8 biți

# Spațiu ocupat în memorie

dimensiuneOcteti.c

```
1 #include <stdio.h>
2 #include <limits.h>
3 #include <float.h>
4
5 int main()
6 {
7     //tipul char
8     printf("\nsizeof(char) = %d \n", sizeof(char));
9     printf("valoarea minima pt o variabila de tip char este %d \n", CHAR_MIN);
10    printf("valoarea maxima pt o variabila de tip char este %d \n\n", CHAR_MAX);
11
12    //tipul int
13    printf("sizeof(int) = %d \n", sizeof(int));
14    printf("valoarea minima pt o variabila de tip int este %d \n", INT_MIN);
15    printf("valoarea maxima pt o variabila de tip int este %d \n\n", INT_MAX);
16
17    //tipul float
18    printf("sizeof(float) = %d \n", sizeof(float));
19    printf("valoarea minima > 0 pt o variabila de tip float este %E \n", FLT_MIN);
20    printf("valoarea maxima pt o variabila de tip float este %E \n", FLT_MAX);
21    printf("valoarea maxima pt o variabila de tip float este %lf \n", FLT_MAX);
22    printf("Precizia folosita pentru variabile de tip float este de %d zecimale\n\n\n", FLT_DIG);
23
24    //tipul double
25    printf("sizeof(double) = %d \n", sizeof(double));
26    printf("valoarea minima > 0 pt o variabila de tip double este %E \n", DBL_MIN);
27    printf("valoarea maxima pt o variabila de tip double este %E \n", DBL_MAX);
28    printf("valoarea maxima pt o variabila de tip double este %lf \n", DBL_MAX);
29    printf("Precizia folosita pentru variabile de tip double este de %d zecimale\n\n\n", DBL_DIG);
30
31    return 0;
32 }
```

# Spațiu ocupat în memorie

```
sizeof(char) = 1
valoarea minima pt o variabila de tip char este -128
valoarea maxima pt o variabila de tip char este 127

sizeof(int) = 4
valoarea minima pt o variabila de tip int este -2147483648
valoarea maxima pt o variabila de tip int este 2147483647

sizeof(float) = 4
valoarea minima > 0 pt o variabila de tip float este 1.175494E-38
valoarea maxima pt o variabila de tip float este 3.402823E+38
valoarea maxima pt o variabila de tip float este 340282346638528859811704183484516925440.000000
Precizia folosita pentru variabile de tip float este de 6 zecimale

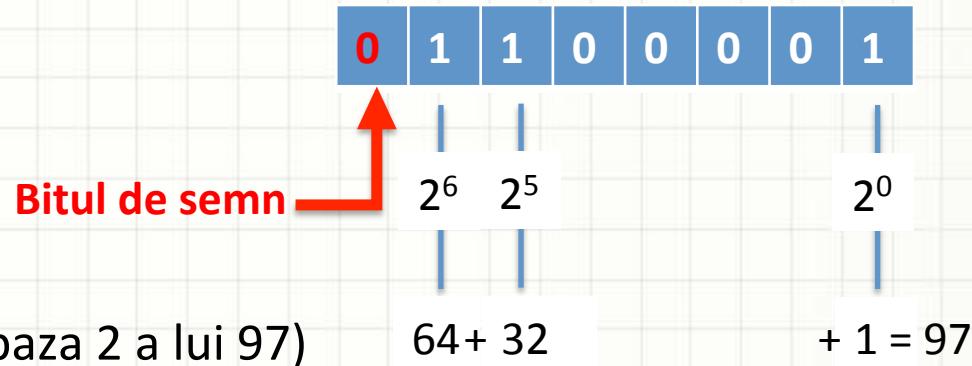
sizeof(double) = 8
valoarea minima > 0 pt o variabila de tip double este 2.225074E-308
valoarea maxima pt o variabila de tip double este 1.797693E+308
valoarea maxima pt o variabila de tip double este 1797693134862315708145274237317043567980705675
258449965989174768031572607800285387605895586327668781715404589535143824642343213268894641827684
675467035375169860499105765512820762454900903893289440758685084551339423045832369032229481658085
59332123348274797826204144723168738177180919299881250404026184124858368.000000
Precizia folosita pentru variabile de tip double este de 15 zecimale
```

# Reprezentarea în memorie

- **char**: ocupă 1 octet = 8 biți, valori între  $-2^7 = -128$  și  $2^7 - 1 = 127$

`char ch = 'a';`

'a' are codul ASCII 97



$$97 = 2^6 + 2^5 + 2^0 \text{ (scrierea în baza 2 a lui 97)}$$

$$64 + 32$$

$$+ 1 = 97$$

- **int**: ocupă 4 octeți = 32 biți, valori între  $-2^{31}$  și  $2^{31} - 1$

`int i = 190;`



Reprezentarea binara a lui 190 in memoria calculatorului

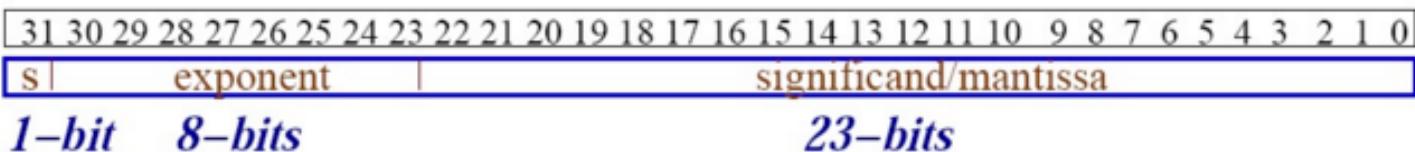
$$190 = 2^7 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 \text{ (scrierea în baza 2 a lui 190)}$$

# Reprezentarea în memorie



$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent-Bias})}$$

- float: ocupă 4 octeți = 32 biți, precizie simplă, bias = 127



float f = 7.0;       $7.0 = 1.75 * 4 = (-1)^0 * (1+0.75) * 2^{(129-127)}$

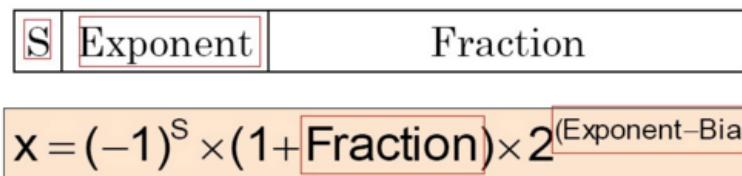
Reprezentare binara exponent:  $129 = 128 + 1 = 2^7 + 2^0$

Reprezentare binara fractie:  $0.75 = 0.5 + 0.25 = 2^{-1} + 2^{-2}$

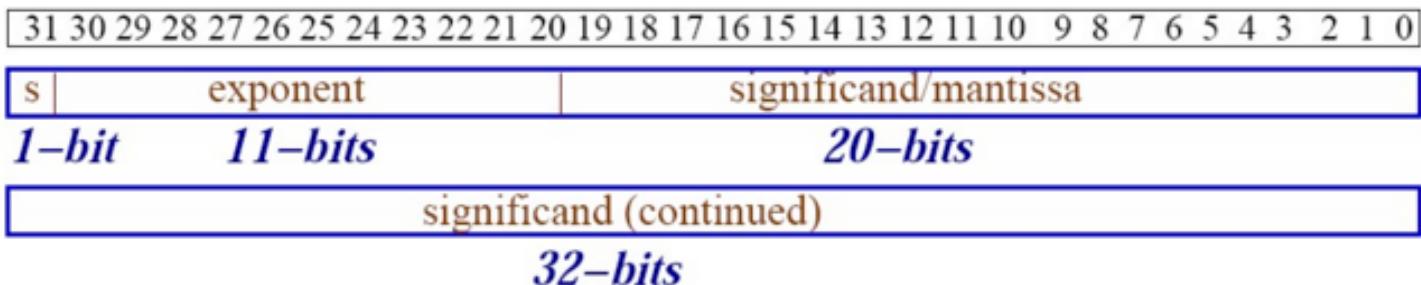


Reprezentarea binara a lui 7.0 (float) in memoria calculatorului

# Reprezentarea în memorie



- **double**: ocupă 8 octeți = 64 biți, precizie dublă, bias = 1023



**double d = 7.0;**     $7.0 = 1.75 * 4 = (-1)^0 * (1+0.75) * 2^{(1025-1023)}$

**Reprezentare binara exponent:  $1025 = 1024 + 1 = 2^{10} + 2^0$**

**Reprezentare binara fractie:  $0.75 = 0.5 + 0.25 = 2^{-1} + 2^{-2}$**

