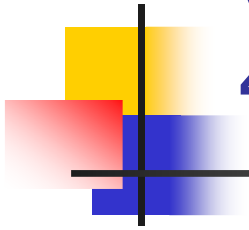
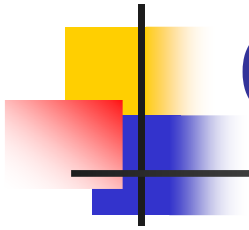
Сложеност алгоритама (Програмирање 2, глава 3, глава 4-4.3)

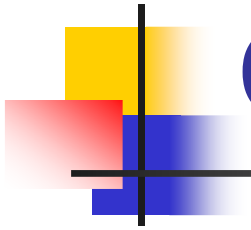


- Проблем: класа задатака истог типа
- Велики број различитих (коректних) алгоритама
- “Величина” (димензија) проблема
  - нпр. количина података које треба обрадити
  - нпр. за сортирање низа: број елемената, N
- Сложеност алгоритма
  - временска
  - просторна
  - функција димензије проблема



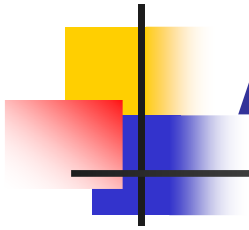
Сложеност алгоритама

- Временска и просторна сложеност алгоритма
  - У терминима времена и простора утрошеног за конкретну улазну величину
  - У терминима асимптотског понашања времена и простора када улазна величина расте



Сложеност алгоритама

- Временска и просторна сложеност алгоритма одређује и његову употребљивост тј. највећу димензију улаза за коју је могуће да се алгоритам изврши у разумном времену и простору
- Мерење утрошеног времена
  - Библиотечке функције, нпр. Clock
  - Процена времена извршавања појединих операција на специфичним процесорима и оперативним системима
  - Профајлери



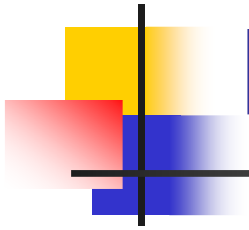
Анализа најгорег случаја

- Често се алгоритми не извршавају исто за све улазне податке исте димензије (величине)
- Нпр. алгоритми сортирања
- Анализа најгорег случаја процењује сложеност алгоритма у најгорем случају – најчешће се примењује за поређење алгоритама
- Просечно време – није увек могуће израчунати

Табела времена извршавања: 1 инструкција – 1 наносекунда (10^{-9} секунди)

$n \setminus f(n)$	$\log n$	n	$n \log n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 *ms*
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 *ms*	77.1 *god*
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 *s*	8.4×10^{15} *god*
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 *min*	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 *dan*	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} *god*	
1,000	0.010 μs	1.0 μs	9.966 μs	1 *ms*		
10,000	0.013 μs	10 μs	130 μs	100 *ms*		
100,000	0.017 μs	0.10 μs	1.67 *ms*	10 *s*		
1,000,000	0.020 μs	1 *ms*	19.93 *ms*	16.7 *min*		
10,000,000	0.023 μs	0.01 *s*	0.23 *s*	1.16 *dan*		
100,000,000	0.027 μs	0.10 *s*	2.66 *s*	115.7 *dan*		
1,000,000,000	0.030 μs	1 *s*	29.9 *s*	31.7 *god*		

f(n) је број инструкција које алгоритам изврши за улаз величине n



Пример

- Ако је број инструкција $n^2 + 2n$, за улаз димензије 1.000.000, члан n^2 троши 16.7 минута а члан 2n само додатне две милисекунде.
- Сложеност (и временска и просторна) је скоро потпуно одређена водећим чланом у изразу за сложеност (број инструкција за временску)
- Мултипликативне и адитивне константе не утичу много

O() – нотација

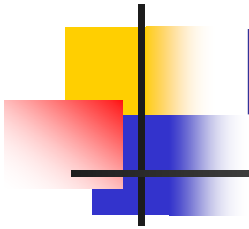
- O() – нотација (Бахман-Ландау; Edmund Landau, Paul Bachmann – немачки научници, почетак 20.в.):
  - гранично понашање функције
  - када аргумент тежи некој специфичној вредности или бесконачности
  - у терминима једноставнијих функција
  - формално: f(x), g(x)
  - f(x) = O(g(x)) $x \to \infty$ akko постоји $c>0$, $x_0 \in R$
    $|f(x)| \leq c * |g(x)|$ за $x > x_0$

Сложеност алгоритама: O() – нотација

- У процени временске или просторне сложености, аргумент је природни број n
- Ако постоји позитивна константа c и природни број n_0 такви да за функције f и g над природним бројевима важи
- $f(n) \leq c * g(n)$ за све вредности $n > n_0$ онда пишемо
- $f = O(g)$
- заправо значи $f \in O(g)$
- и читамо „f је велико о од g“
- На пример, за $n \in N$, $f(n)=2n^2+n-2$, $f=O(n^2)$, али и $f = O(n^3)$;

Адитивне и мултипликативне константе

- У изразу $5n^2 + 1$, константа 1 је адитивна а константа 5 мултипликативна
- Ове константе не утичу на класу сложености којој функција припада
- Не говори се да је функција из класе $O(5n^2 + 1)$
- Ако је фунција временске сложености једног алгоритма $5n^2 + 1$ а другог n^2, извршавање првог алгоритма на рачунару који је 6 пута бржи биће брже за сваку величину улаза од извршавања другог алгоритма на 6 пута споријем рачунару
- Ако је функција временске сложености једног алгоритма n^2 а другог n, не постоји рачунар на којем ће се први алгоритам извршавати брже од другог за сваку величину улаза



Примери

$$n^2 = O(n^2)$$
$$n^2 + 10 = O(n^2)$$
$$10 \cdot n^2 + 10 = O(n^2)$$
$$10 \cdot n^2 + 8n + 10 = O(n^2)$$
$$n^2 = O(n^3)$$
$$2^n = O(2^n)$$
$$2^n + 10 = O(2^n)$$
$$10 \cdot 2^n + 10 = O(2^n)$$
$$2^n + n^2 = O(2^n)$$
$$3^n + 2^n = O(3^n)$$
$$2^n + 2^n n = O(2^n n)$$

Нека својства O-нотације

- Ако је $f_1 = O(g_1)$ и $f_2 = O(g_2)$, онда је и
- $f_1 \cdot f_2 = O(g_1 \cdot g_2)$
- $f_1 + f_2 = O(g_1 + g_2)$

- Ако важи $f = O(g)$ и a и b су позитивне константе, онда важи и
- $af + b = O(g)$

Сложеност алгоритама: Ω(), θ() – нотација

- Слично се дефинишу Ω и θ асимптотске нотације:
  - Ако постоје позитивна константа c и природни број n_0 такви да за функције f и g над природним бројевима важи
  - $f(n) \geq c * g(n)$ за све вредности $n > n_0$ онда пишемо
  - $f = \Omega(g)$
  - и читамо „f је велико омега од g“

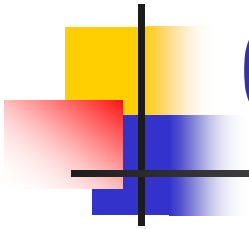
Сложеност алгоритама: Ω(), θ() – нотација

- Ако постоје позитивне константе c_1 и c_2 и природни број n_0 такви да за функције f и g над природним бројевима важи
- $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$ за све вредности $n > n_0$ онда пишемо
- $f = \theta(g)$
- и читамо „f је велико тета од g“

- Важи да је $f = \theta(g)$ ако и само ако је $f = O(g)$ и $f = \Omega(g)$

Ω() – нотација: примери

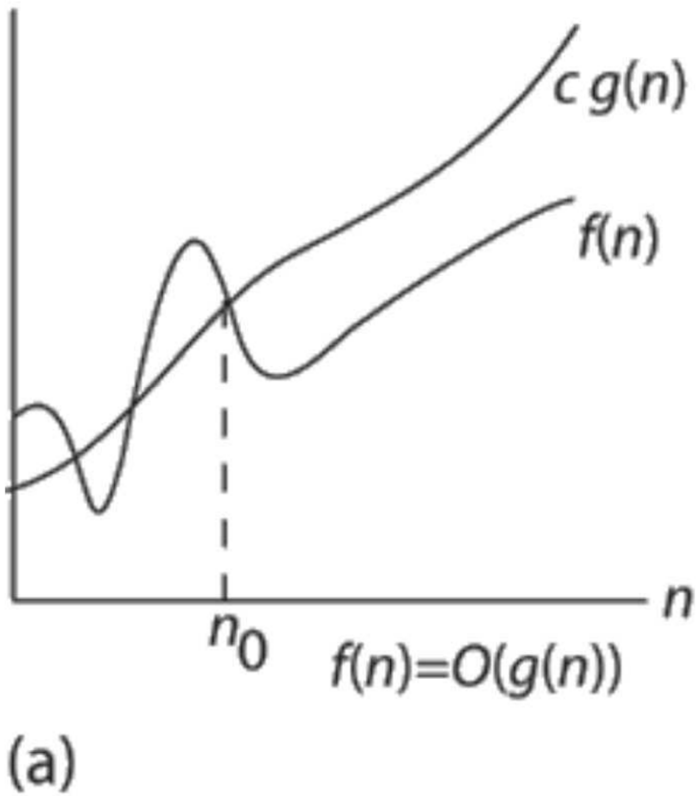
$$n^2 = \Omega(n^2)$$
$$n^2 + 10 = \Omega(n^2)$$
$$10 \cdot n^2 + 10 = \Omega(n^2)$$
$$10 \cdot n^2 + 8n + 10 = \Omega(n^2)$$
$$n^2 \neq \Omega(n^3),\ n^2 = \Omega(n),\ n^3 = \Omega(n^2)$$
$$2^n = \Omega(2^n)$$
$$2^n + 10 = \Omega(2^n)$$
$$10 \cdot 2^n + 10 = \Omega(2^n)$$
$$2^n + n^2 = \Omega(2^n),\ 2^n + n^2 = \Omega(n^2)$$
$$3^n + 2^n = \Omega(3^n),\ 3^n + 2^n = \Omega(2^n)$$
$$2^n + 2^n n = \Omega(2^n n),\ 2^n + 2^n n = \Omega(2^n)$$

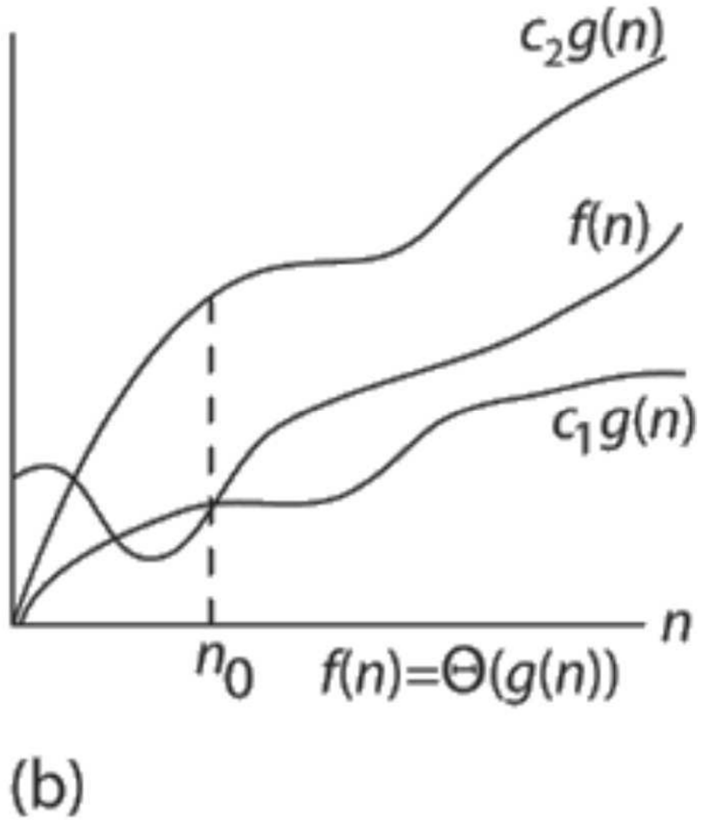


θ() – нотација: примери

$$n^2 = \Theta(n^2)$$
$$n^2 + 10 = \Theta(n^2)$$
$$10 \cdot n^2 + 10 = \Theta(n^2)$$
$$10 \cdot n^2 + 8n + 10 = \Theta(n^2)$$
$$n^2 \neq \Theta(n^3)$$
$$2^n = \Theta(2^n)$$
$$2^n + 10 = \Theta(2^n)$$
$$10 \cdot 2^n + 10 = \Theta(2^n)$$
$$2^n + n^2 = \Theta(2^n)$$
$$3^n + 2^n = \Theta(3^n)$$
$$2^n + 2^n n = \Theta(2^n n)$$

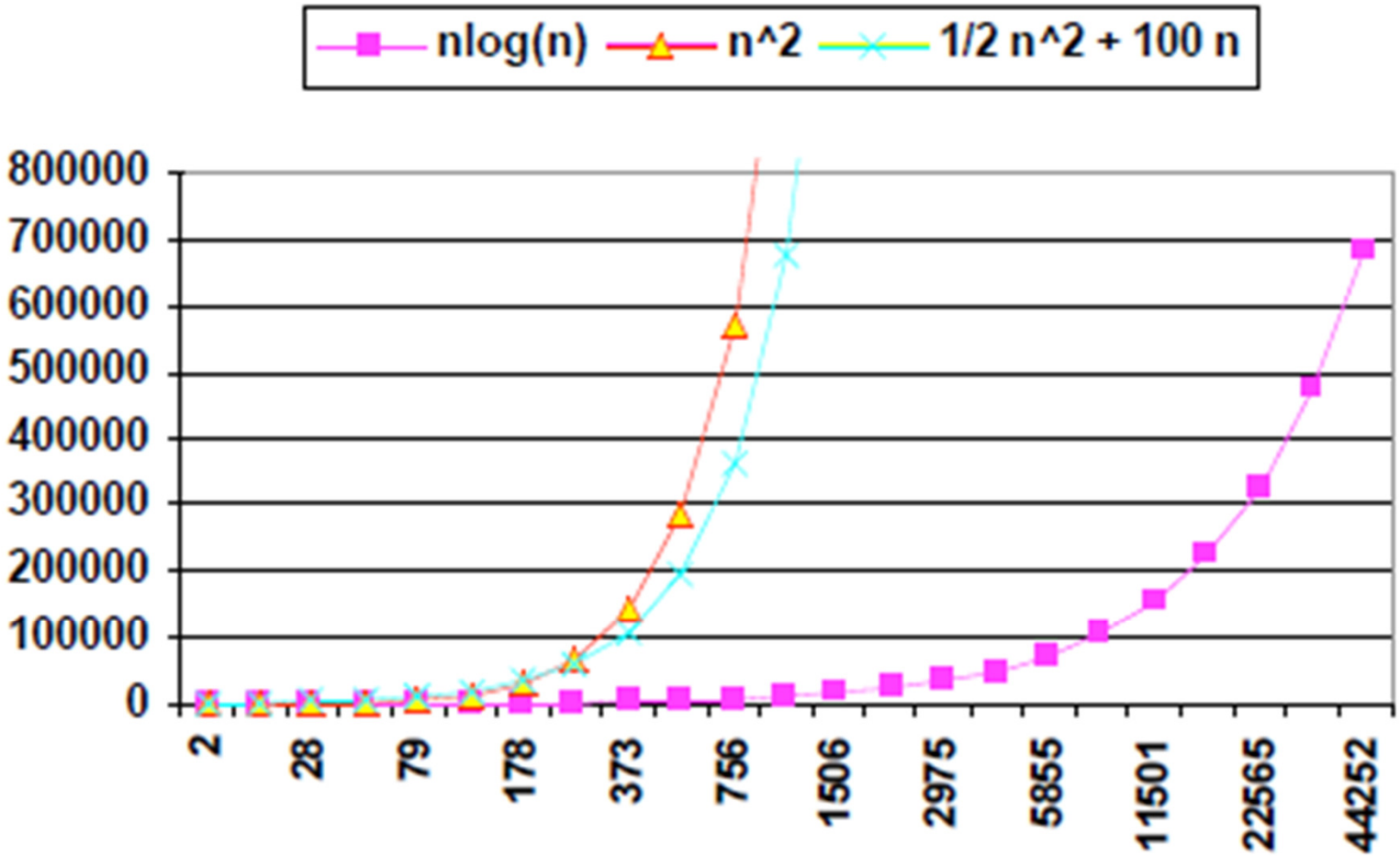
Сложеност алгоритама: O(), θ() – нотација





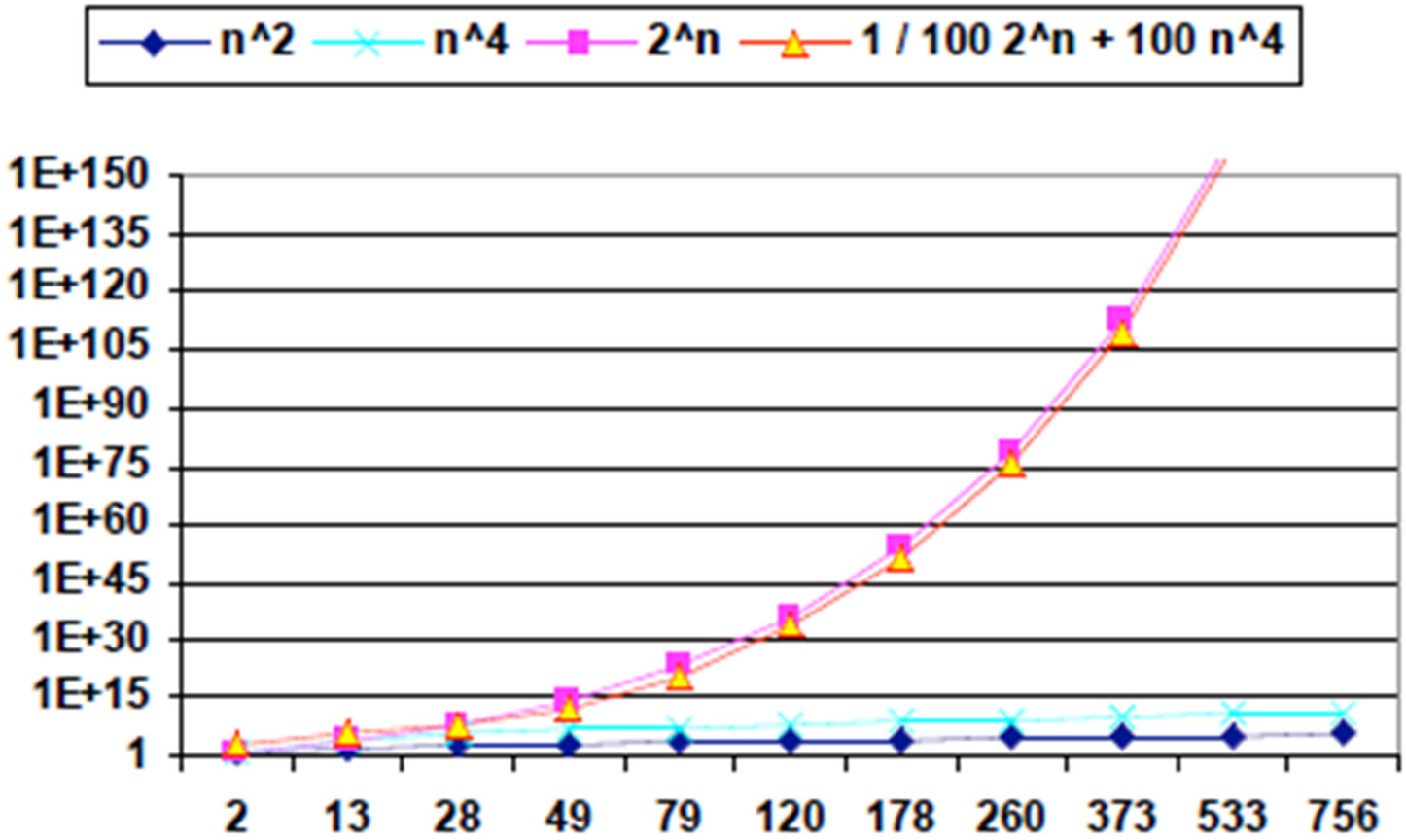
Complexity Examples

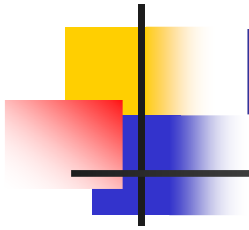
- $\frac{1}{2} n^2 + 100\, n \Rightarrow O(n^2)$



Complexity Examples

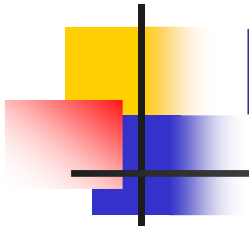
- $1/100\ 2^n + 100\ n^4 \Rightarrow O(2^n)$





Класе сложености

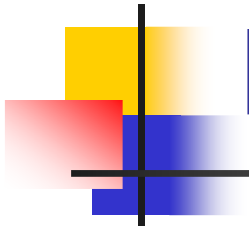
- Класа сложености O (Ω, θ)
- Ако је T(n) време извршавања алгоритма A (улаза димензије n) и ако важи да је T = O(g) (Ω(g), θ(g)) онда кажемо да је алгоритам A сложености (реда) O(g) (Ω(g), θ(g)) или да припада класи O(g) (Ω(g), θ(g)).



Класе сложености

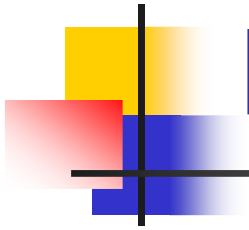
- Однос између нотација
- Пример: $5n = O(n), O(n^2), \theta(n)$ АЛИ НИЈЕ $\theta(n^2)$

- O(g) обично подразумева најмању класу којој алгоритам припада (или најмању за коју то може да се докаже, у претходном примеру O(n))



Класе сложености

- Класе сложености
- O(1) константна сложеност
- O(log n) логаритамска сложеност
- O(n) линеарна сложеност
- $O(n^2)$ квадратна сложеност
- $O(n^3)$ кубна сложеност
- $O(n^k)$ за неко k полиномијална сложеност
- $O(2^n)$ експоненцијална сложеност

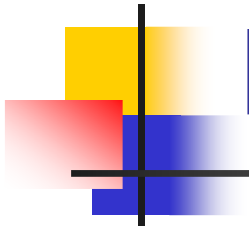


Временска сложеност - класе

- O(1): време извршавања константно;
  - сви искази се извршавају по неколико пута, независно од димензије проблема; најбоље
  - алгоритам директног претраживања у идеалном случају

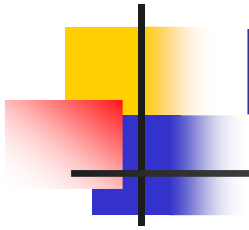
Временска сложеност - класе

- O(log n*)*: време извршавања лагано расте (мање од неке “велике” константе);
  - алгоритми који проблем решавају тако што га деле на 2, 3... потпроблема и онда решавају само један од тих потпроблема
  - кад се n удвостручи, log n порасте за константу *(*log*2);* кад се n квадрира, log n се тек удвостручи
- Ако је основа логаритма 10, за n = 1000, $\log_{10} n$ је 3, за n =
- 1.000.000, log*n* је само два пута веће, тј. 6.
- Ако је основа логаритма 2, за n = 1000, $\log_2 n$ је приближно 10, што је већа константа од 3, али небитно већа.
- $\log_a n = \log_b n \ \log_a b$.
  - Алгоритам бинарног претраживања



Временска сложеност - класе

- *O(*n*)*: мала количина обраде над сваким улазним податком
  - кад се n удвостручи, и време извршавања се удвостручи
  - алгоритам линеарног претраживања



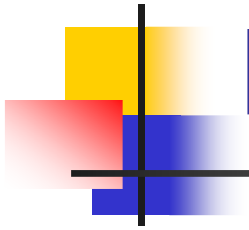
Временска сложеност - класе

- O(*n* log *n*) : алгоритми који проблем решавају тако што га разбију у мање потпроблеме, реше их независно и онда комбинују решење
  - Кад се n удвостручи, *n* log *n* се повећа за нешто више него двоструко
- Када је n милион, n log n је око двадесет милиона.
- Алгоритам брзог сортирања низа

Временска сложеност - класе

- $O(n^2)$: алгоритми који обрађују све парове улазних података (двострука петља)
  - кад се n удвостручи, време извршавања се увећа 4 пута
  - елементарне методе сортирања, нпр. сортирање избором
  - практични за релативно мале димензије проблема

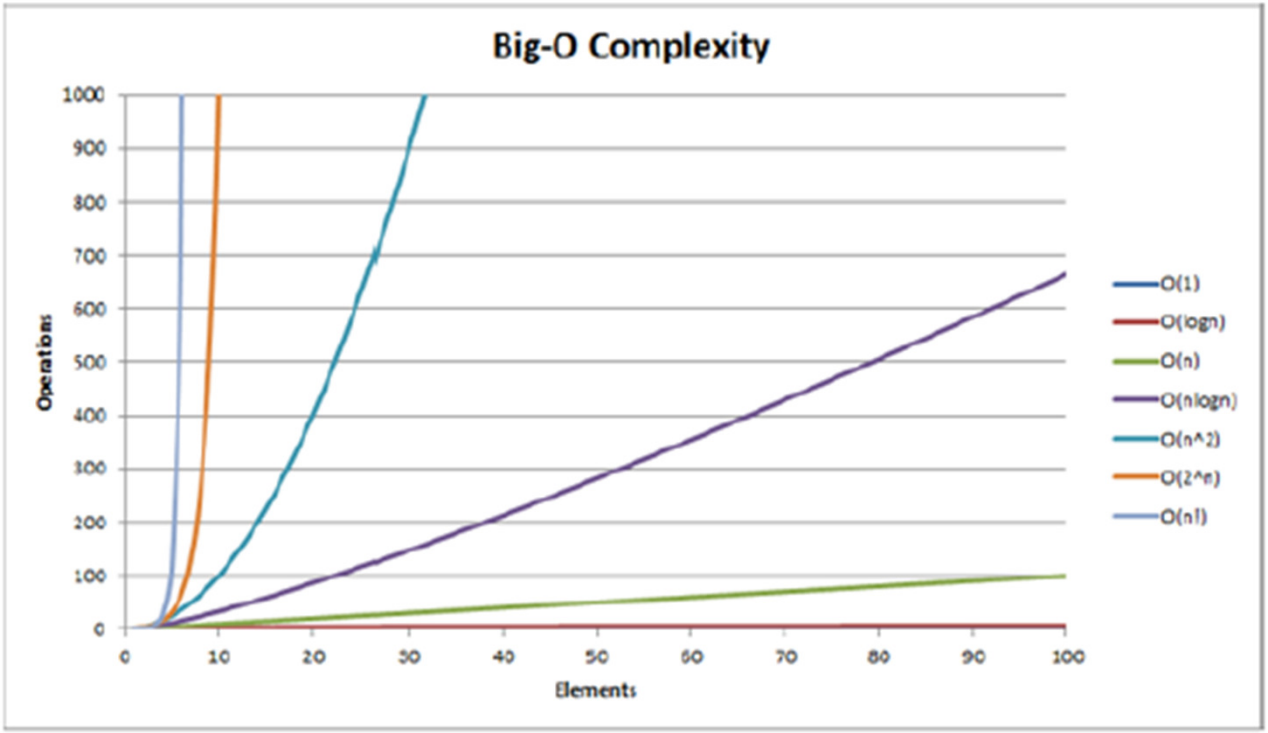
- $O(n^3)$: алгоритми који обрађују тројке улазних података (трострука петља)
  - кад се n удвостручи, време извршавања се увећа 8 пута
  - нпр. за n 100, време извршавања је милион
  - само за мале димензије проблема



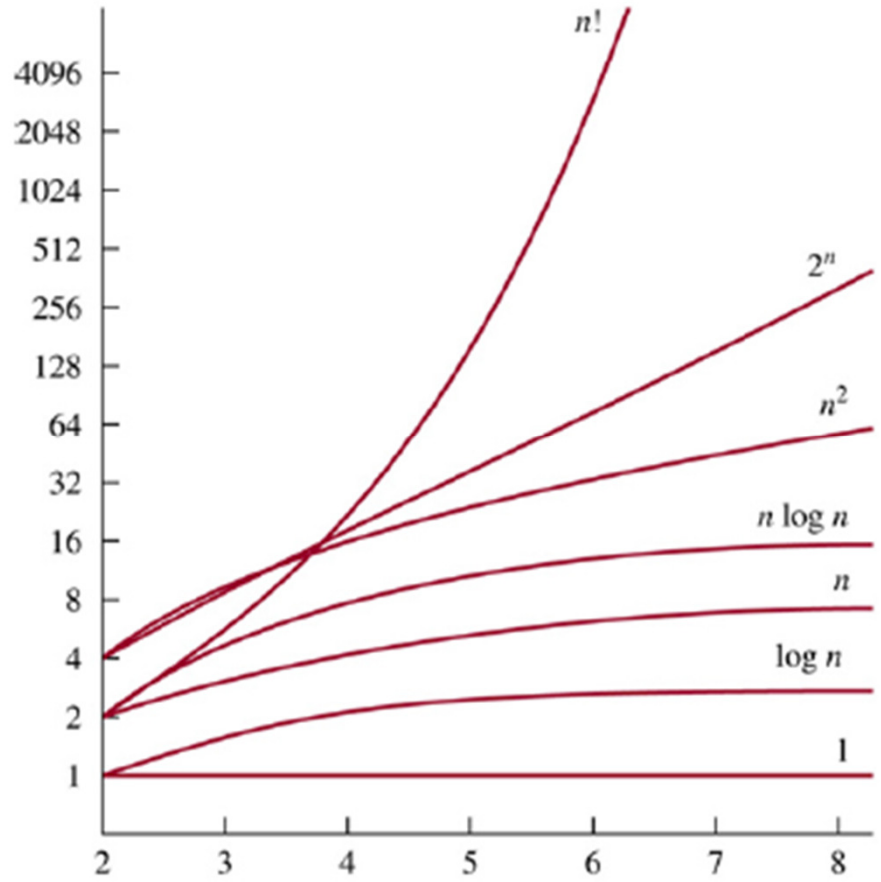
Временска сложеност - класе

- $O(2^n)$: експоненцијална сложеност
  - алгоритми грубе силе
  - ретко практични за употребу
  - када се n удвостручи, време се квадрира (за n 20, време извршавања је милион)
  - За неке проблеме нису познати бољи алгоритми

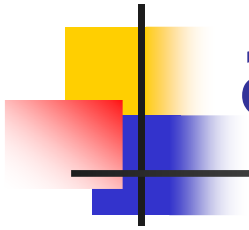
Примери неких класа сложености



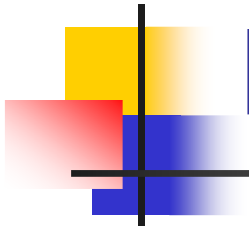
Примери неких класа сложености



Просторна сложеност алгоритама



- Функција улаза: захтевани ДОДАТНИ меморијски простор (поред простора за смештање улазних података)
- Исте асимптотске нотације као за временску сложеност (O(), Ω(), θ())
- Исте класе алгоритама као за временску сложеност
- На пример:
  - O(1) – алгоритам сортирања избором
  - O(log *N*) – алгоритам брзог сортирања (локације за обраду рекурзије)
  - O(*N*) – алгоритам сортирања спајањем – локације за премештање елемената



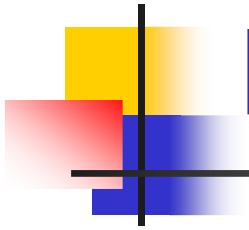
Примери

- Оценити временску сложеност следећих кодова. Одговоре образложити.
- ```
  int zbir(int x, int y) {
        return x+y;
   }
  ```

- ```
  int k=n;
  while(k>0){
        printf("%d ", k);
        k=k/2;
  }
  ```

Примери

- ```
  int suma(int niz[], int n) {
        int i, suma = 0;
        for(i=0; i<n; i++)
              suma+=niz[i];
        return suma;
  }
  ```
- ```
  for(i=0; i<n; i++) {
        int k=n;
        while(k>0){
              printf("%d ", k);
              k=k/2;
        }
  }
  ```



Примери

- ```
  for(i=0; i<n; i++)
      for(j=0; j<i; j++)
          a[i]+=j;
  ```

- ```
  for(i=0; i<n; i++)
      for(j=0; j<n; j++)
          for(k=0; k<n; k++)
              a[i][j] = a[i][k]+a[k][j];
  ```

Израчунавање временске и просторне сложености алгоритама

- Поједностављења
- Јединичне инструкције
- Део програма временске сложености
  - Неколико инструкција без гранања: O(1)
  - Петља која се извршава n пута
    (тело константне сложености): O(n)
  - Једна петља n пута, за њом друга m пута
    (тела константне сложености): O(n+m)
  - Један за другим сложености f, g редом: O(f+g)

Израчунавање временске и просторне сложености алгоритама

- Део програма временске сложености
  - Гранање са гранама O(n), O(m),
    - Max{O(n), O(m)} = O(n+m)
  - Двострука петља (m, n пута, тело унутрашње константне сложености): O(m * n)
  - Аналогно за друге комбинације линеарног кода, гранања и петљи

Израчунавање сложености алгоритама: примери

- Оценити временску и просторну сложеност следећих кодова и образложити.

```
int suma_prvih_5(int niz[], int n) {
  int i, suma = 0;
  for(i=0; i<n; i++) {
    if(i>4) break;
    suma+=niz[i];
  }
  return suma;
}
```

Израчунавање сложености алгоритама: примери

```
int suma_prvih_5(int niz[], int n) {
  int i, suma = 0;
  for(i=0; i<n; i++) {
    if(i>4) break;
    suma+=niz[i];
  }
  return suma;
}
```

- Временска сложеност: O(1)
- Просторна сложеност: O(1)

Израчунавање сложености алгоритама: примери

```
int suma_parnih(int niz[], int n) {
  int i, suma = 0;
  for(i=0; i<n; i++) {
    if(niz[i]%2) continue;
    suma+=niz[i];
      }
      return suma;
}
```

Израчунавање сложености алгоритама: примери

```
int suma_parnih(int niz[], int n) {
  int i, suma = 0;
  for(i=0; i<n; i++) {
    if(niz[i]%2) continue;
    suma+=niz[i];
      }
      return suma;
}
```

- Временска сложеност: O(n)
- Просторна сложеност: O(1)

Израчунавање сложености алгоритама: примери

```
for(i=0; i<n; i++) {
  if(i<5) continue;
  for(j=0; j<n; j++)
    s++;
}
```

Израчунавање сложености алгоритама: примери

```
for(i=0; i<n; i++) {
  if(i<5) continue;
  for(j=0; j<n; j++)
    s++;
}
```

- Временска сложеност: $O(n^2)$
- Просторна сложеност: $O(1)$

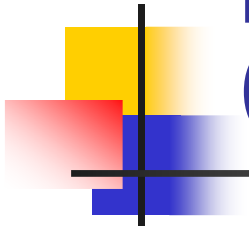
Израчунавање сложености алгоритама: примери

```
for(i=0; i<n; i++)
  for(j=0; j<i*i; j++) {
    k=1; m=n;
    while(m>k) {
      k=k*3;
      m=m/2;
    }
  }
```

Израчунавање сложености алгоритама: примери

```
for(i=0; i<n; i++)
  for(j=0; j<i*i; j++) {
    k=1; m=n;
    while(m>k) {
      k=k*3;
      m=m/2;
    }
  }
```

- Временска сложеност: O($n^3 \log n$)
- Просторна сложеност: O(1)



Израчунавање сложености рекурзивних алгоритама (**Програмирање 2, тачка 4.3**)

- Рекурентне једначине

Израчунавање сложености рекурзивних алгоритама: пример

```
unsigned faktorijel(unsigned n) {
if (n == 0)
return 1;
else
return n*faktorijel(n-1);
}
```

- Нека T(n) означава број инструкција које захтева позив функције faktorijel за улазну вредност n.
- За n = 0, важи T(n) = 2 (једно поређење и једна наредба return).
- За n > 0, важи T(n) = 4 + T(n - 1) (једно поређење, једно множење, једно одузимање, број инструкција за faktorijel за n-1 и једна наредба return).

Израчунавање сложености рекурзивних алгоритама: пример

- T(n) = 4+ T(n-1)
- = 4 + 4+ T(n-2)
- = ...
- = 4n + T(0) = 4n + 2 = *O(n)*

- Просторна сложеност: број стек оквира, сваки са константним простором: *O(n)*
- Линеарна временска и просторна сложеност

Израчунавање сложености рекурзивних алгоритама: пример

```
unsigned suma2(unsigned n) {
if (n <= 0)
return 0;
else
return n + suma2(n-2);
}
```

- Број рекурзивних позива: *n/2*
- Временска и просторна сложеност: *O(n)*

Израчунавање сложености рекурзивних алгоритама: пример

- Оценити временску и просторну сложеност следећих кодова и образложити.

```
int f(int n) {
  int x;
  if(n==0) return 1;
  x=n%10;
  if(x>1) return x*f(n/x);
  else return 2*f(n/2);
}
```

Израчунавање сложености рекурзивних алгоритама: пример

- Оценити временску и просторну сложеност следећих кодова и образложити.

```
int f(int n) {
  int x;
  if(n==0) return 1;
  x=n%10;
  if(x>1) return x*f(n/x);
  else return 2*f(n/2);
}
```

- Временска сложеност: O(*log n*)
- Просторна сложеност: O(*log n*)

Израчунавање сложености рекурзивних алгоритама:

```
int f(int n) {
        if (n <= 1) {
                return 1;
        }
        return f(n - 1) + f(n - 1);
}
```

Израчунавање сложености рекурзивних алгоритама:

```
int f(int n) {
        if (n <= 1) {
                return 1;
        }
        return f(n - 1) + f(n - 1);
}
```

- Временска сложеност $O(2^n)$
- Просторна сложеност $O(n)$

Израчунавање сложености рекурзивних алгоритама:

```
int f(int n) {
   int x;
   if(n==0) return 1;
   x=n%10;
   y=(n/10)%10;
   return x*f(n-1)+y*f(n-1);
}
```

Израчунавање сложености рекурзивних алгоритама:

```
int f(int n) {
  int x;
  if(n==0) return 1;
  x=n%10;
  y=(n/10)%10;
  return x*f(n-1)+y*f(n-1);
}
```

- Временска сложеност $O(2^n)$
- Просторна сложеност O(n)

Израчунавање сложености рекурзивних алгоритама:

```
int f(int n) {
  if(n<=0) return 1;
  return f(n-2)*f(n-2);
}
```

Израчунавање сложености рекурзивних алгоритама:

```
int f(int n) {
  if(n<=0) return 1;
  return f(n-2)*f(n-2);
}
```

- Временска сложеност $O(2^{n/2})$
- Просторна сложеност O(n)

Израчунавање сложености рекурзивних алгоритама:

```
int f(int n) {
  if(n<=0) return 1;
  return f(n-2)*f(n-2)*f(n-2)*f(n-2);
}
```

Израчунавање сложености рекурзивних алгоритама:

```
int f(int n) {
  if(n<=0) return 1;
  return f(n-2)*f(n-2)*f(n-2)*f(n-2);
}
```

- Временска сложеност $O(4^{n/2}) = O(2^n)$
- Просторна сложеност O(n)