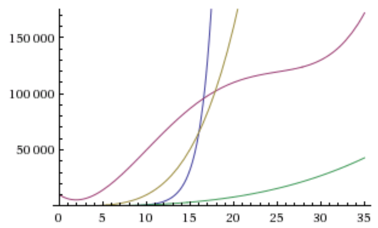
Сложност на алгоритмите (Неформален увод)

11 ноември 2015 г.

“Скорост на растеж” на функция

Нотацията "big O"



- 2^x
- x^4
- $(x - 20)^4 + 10(x - 30)^3 + 120,000$
- x^3

$f \in \mathcal{O}(g)$:

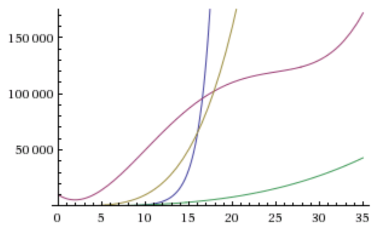
$$\exists M, x_0, \forall x > x_0 : |f(x)| \leq M|g(x)|$$

Търсим проста функция, която ни дава ограничение отгоре на f

$$(x - 20)^4 + 10(x - 30)^3 + 120,000 \in \mathcal{O}(x^4)$$

вярно е и $(x - 20)^4 + 10(x - 30)^3 + 120,000 \in \mathcal{O}(2^x)$

Нотацията "big O"



- 2^x
- x^4
- $(x-20)^4 + 10(x-30)^3 + 120,000$
- x^3

$f \in \mathcal{O}(g)$:

$$\exists M, x_0, \forall x > x_0 : |f(x)| \leq M|g(x)|$$

Търсим проста функция, която ни дава ограничение отгоре на f

$$(x-20)^4 + 10(x-30)^3 + 120,000 \in \mathcal{O}(x^4)$$

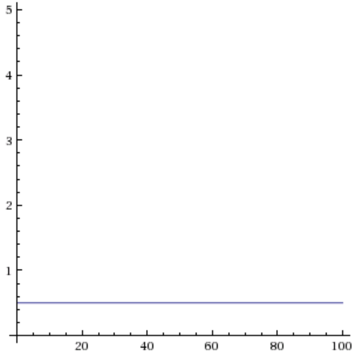
вярно е и $(x-20)^4 + 10(x-30)^3 + 120,000 \in \mathcal{O}(2^x)$

Ресурси, нужни за изпълнението на алгоритъм,
като функция на "обема" на входа

Константно време

```
do something simple;
do something else simple;
do something simple;
```

```
//int a[n];
cin >> i;
cout << "a[" << i
     << "]=" << i <<endl;
```



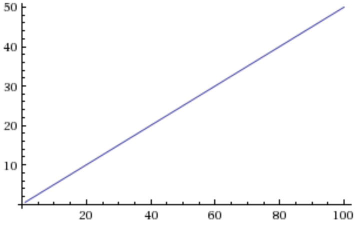
Времето за изпълнение не се изменя с увеличаване на n.
$T_P(n) \in \mathcal{O}(1)$;

Линейно време

$\forall x \in Input$ *do*
something with
constant time complexity;

```
int maxi = 0;
for (int i = 1; i < n; i++)
{
  if (a[i] > a[maxi])
    maxi = i;
}
```

Време за една итерация, c,
Време за 3 итерации, $3 \times c$.



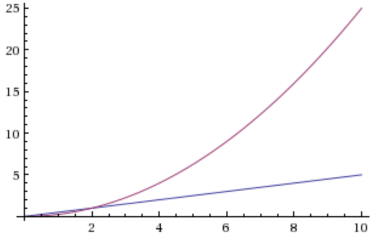
Времето за изпълнение нараства линейно с увеличаване на n.
$T_P(n) \in \mathcal{O}(n)$;

Квадратно време

$\forall x \in Input$ *do*
something with linear time complexity;

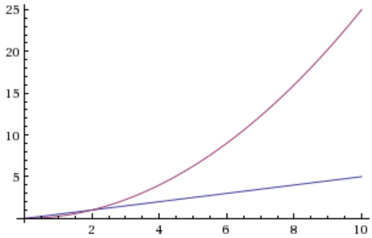
```
for (int i = 0; i < n; i++)
  for (j = i+1; j < n; j++)
  {
    if (a[i] == a[j])
      cout << "Duplicate!";
  }
```

при n=3,
Време за една итерация, $3 \times c$,
Време за 3 итерации, $3 \times 3 \times c''$.



Времето за изпълнение нараства квадратно с увеличаване на n.
$T_P(n) \in \mathcal{O}(n^2)$;

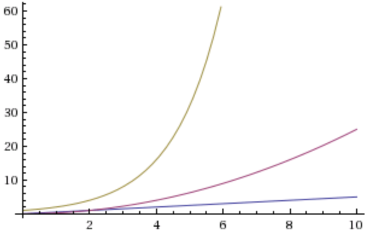
Има ли нещо по-лошо от степенната функция?



Екпоненциално време

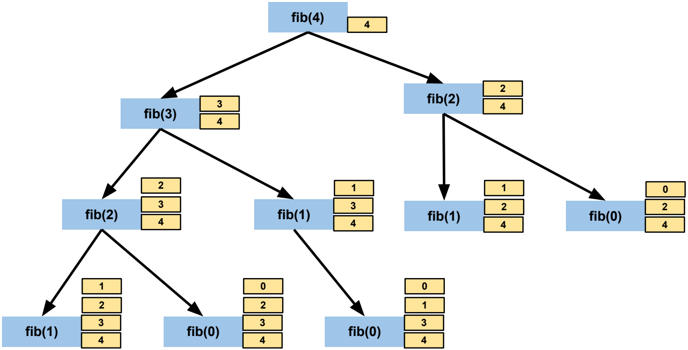
k times ***do***
something with
complexity $\mathcal{O}(k^{n-1})$;

```
int fib (int n)
{
  if (n <= 1) return n;
  return fib (n-1) + fib (n-2);
}
```



Времето за изпълнение нараства експоненциално с увеличаване на n. $T_P(n) \in \mathcal{O}(a^n)$;

n-то число на Фибоначи

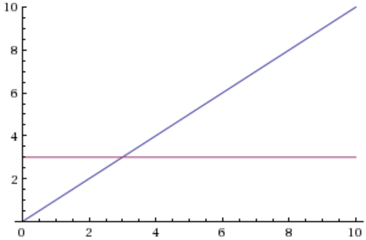



```
int fib_n (int n)
{
  if (n <= 1)
    return 1;
  return fib_n(n-2) + fin_n(n-1);
}
```

n-то число на Фибоначи за линейно време

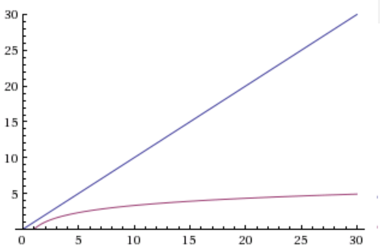
```
int fib_n (int n)
{
  int a = 0, b = 1;
  while (n > 0)
  { //(a,b) -> (b,a+b)
    b += a;
    a = b - a;
    n--;
  }
  return a;
}
```

Има ли нещо по-добро от линейното, но не чак като константното?



Логаритмично време

```
do something simple;
solve a
  two times simpler sub-problem;
```




```
bool find (int x, int arr[], int n)
//arr is ordered
{
  if (n == 0)
    return false;

  if (x >= arr[n/2])
    return x==arr[n/2] ||
          find (x,arr+ceil(n/2.0),n/2);

  return find (x,arr,n/2);


}
```

Сложността на проблема намалява двойно на всяка стъпка. Времето за изпълнение нараства логаритмично с увеличаване на n. $T_P(n) \in \mathcal{O}(log_2(n))$;

Благодаря за вниманието!