

# Časová složitost

**Časová složitost** nám říká, jak se prodlouží doba běhu programu, když mu předhodíme více dat. Pomáhá programátorům předpovědět, zda jejich kód zvládne reálný provoz (miliony uživatelů) nebo zda „zkolabuje“.

## 1. Tři scénáře náročnosti

U každého algoritmu můžeme sledovat tři různé případy:

- **Best Case ( $O(1)$ ):** Nejlepší možný scénář (např. hledaný prvek je hned na prvním místě).
- **Average Case ( $O(n)$ ):** Průměrný scénář (očekávaný výkon v běžném provozu).
- **Worst Case ( $O(n^2)$ ):** Nejhorší možný scénář (např. prvek v seznamu není, musíme projít vše).  
**Tento scénář je v IT nejdůležitější.**

## 2. Typické příklady v praxi

### Konstantní čas - $O(1)$

Počet operací je pevně daný.

- **Příklad:** Zjištění, zda je číslo sudé nebo liché ( $\text{číslo} \% 2 == 0$ ). Je jedno, jestli má číslo 2 nebo 2 miliony cifer, operace je stejně rychlá.

### Lineární čas - $O(n)$

Čas roste přímo úměrně s daty.

- **Příklad:** Hledání nejmenšího čísla v neseřazeném poli. Musíte se podívat na každé číslo přesně jednou.
- **Vliv:** Pokud se pole 10x zvětší, hledání bude trvat 10x déle.

### Logaritmický čas - $O(\log n)$

Extrémně efektivní růst.

- **Příklad:** Hledání slova ve slovníku (půlením intervalu). I kdyby měl slovník miliardu stran, najdete slovo maximálně na 30 pokusů.

### Kvadratický čas - $O(n^2)$

Čas roste se čtvercem dat.

- **Příklad:** [Bubble Sort](#) (třídění prvků). Pro 100 prvků uděláte 10 000 operací. Pro 100 000 prvků už je to 10 miliard operací – zde už program citelně zamrzá.

### 3. Časová složitost vs. Výkon hardwaru

Častým omylem je myslet si, že pomalý algoritmus vyřešíme rychlejším procesorem.

- Pokud máte algoritmus se složitostí  $O(2^n)$  (exponenciální), přidání 10x silnějšího procesoru vám dovolí zpracovat jen o pár prvků více.
- Naopak přechod z  $O(n^2)$  na  $O(n \log n)$  může zrychlit zpracování velkých dat z hodin na sekundy i na starém počítači.

### 4. Co ovlivňuje časovou složitost?

1. **Cykly:** Jeden cyklus přes  $n$  prvků znamená  $O(n)$ . Dva vnořené cykly znamenají  $O(n^2)$ .
2. **Rekurze:** Pokud funkce volá sama sebe vícekrát, složitost může růst exponenciálně.
3. **Struktura dat:** Vyhledávání v `[[it_encyklopedie:ethernet|síti]]` nebo databázi závisí na tom, jak jsou data indexována.

**Zajímavost:** Existují i algoritmy se složitostí  $O(n!)$  (faktoriál), například u problému obchodního cestujícího (hledání nejkratší cesty mezi všemi městy). Pro pouhých 60 měst je počet operací větší než počet atomů v pozorovatelném vesmíru.

[Zpět na Big O notaci](#)

From:  
<https://serviceit.cz/> - IT ENCYKLOPEDIE

Permanent link:  
[https://serviceit.cz/doku.php?id=casova\\_slozitest](https://serviceit.cz/doku.php?id=casova_slozitest)

Last update: **2025/12/31 17:21**

