Big O notace (Asymptotická složitost)

Big O notace je jazyk, který programátoři používají k porovnávání efektivity algoritmů. Zaměřuje se na „nejhorší možný scénář“ (Worst Case). Pomáhá nám odpovědět na otázku: *„Pokud se množství dat zdvojnásobí, zpomalí se program dvakrát, čtyřikrát, nebo vůbec?“*

1. Nejběžnější typy složitosti

Zde jsou seřazeny od nejrychlejších po nejpomalejší:

O(1) – Konstantní složitost

Doba běhu je stále stejná, bez ohledu na to, jak velká jsou data.

- **Příklad:** Přístup k prvku v poli pomocí indexu (vím přesně, kde je).
- **Vnímání:** Extrémně rychlé.

O(log n) – Logaritmická složitost

Doba běhu roste velmi pomalu. Typické pro algoritmy, které při každém kroku rozdělí data na polovinu.

- **Příklad:** Binární vyhledávání v seřazeném seznamu.
- **Vnímání:** Velmi efektivní (i pro miliardy prvků stačí pár desítek kroků).

O(n) – Lineární složitost

Doba běhu roste přímo úměrně s počtem prvků.

- **Příklad:** Prohledávání neseřazeného seznamu (musím se podívat na každý prvek).
- **Vnímání:** Férové, běžné u jednoduchých operací.

O(n log n) – Linearitmetická složitost

O něco pomalejší než lineární, ale stále velmi dobrá.

- **Příklad:** Efektivní třídicí algoritmy jako MergeSort nebo QuickSort.
- **Vnímání:** Standard pro zpracování velkých dat.

O(n²) – Kvadratická složitost

Doba běhu roste se čtvercem velikosti dat. Pokud se data zdvojnásobí, čas vzroste čtyřikrát.

- **Příklad:** Dva vnořené cykly (každý prvek porovnávám s každým jiným).
- **Vnímání:** Pomalé. Pro velké objemy dat (např. miliony záznamů) nepoužitelné.

2. Praktické srovnání

Představte si, že jedna operace trvá 1 milisekundu (1 ms). Jak dlouho bude trvat zpracování $n = 1000$ prvků?

Notace	Počet operací	Čas
$O(1)$	1	1 ms
$O(\log n)$	~10	10 ms
$O(n)$	1 000	1 s
$O(n \log n)$	~10 000	10 s
$O(n^2)$	1 000 000	16.6 min
$O(2^n)$	2^{1000}	Více než věk vesmíru

3. Proč ignorujeme konstanty?

V Big O notaci nás nezajímají drobné detaily. Algoritmus, který provede $2n$ operací, je stále považován za **O(n)**. Stejně tak $n + 100$ je stále **O(n)**. V měřítku miliard prvků jsou tyto konstanty nepodstatné – rozhodující je „tvar“ křivky růstu.

4. Časová vs. Prostorová složitost

- **Časová (Time Complexity):** Jak dlouho to trvá.
- **Prostorová (Space Complexity):** Kolik paměti (RAM) algoritmus potřebuje během svého běhu. Často musíme dělat kompromis: buď je algoritmus rychlý a žere moc paměti, nebo je úsporný, ale pomalý.

> **Zlaté pravidlo:** Při programování se vždy snažte vyhnout složitostem jako $O(n^2)$ a horším, pokud víte, že váš systém bude zpracovávat velká data. Rozdíl mezi $O(n \log n)$ a $O(n^2)$ je rozdílem mezi fungující aplikací a systémem, který „zamrzne“.

Zpět na Algoritmy