Asymptotické paměťové a časové složitosti

Asymptotická složitost (AS) je způsob **klasifikace** určitého algoritmu. Chápe se jako „mezní hodnota“ .

Algoritmus je přesně daný **postup**, jak vyřešit určitý problém. Jedná se o nástroj, který nám umožňuje **porovnat efektivitu** všech algoritmů na základě velikosti dat, které přijdou na vstup.

Asymptotická časová složitost

Zapisuje se pomocí **Landauovi** notace, která je známá více pod pojmem „**Big O**“ (Ačkoliv je to špatné označení, protože Big O je nejhorší možný scénář) nebo „**Omikron notace**“.

Pozor!

Big O - je nejhorší scénář!
Théta (Θ) je průměr!
Omega – Nejlepší scénář!

Značí se **Θ(N)**, kde N je zvýšení času s nárůstem dat.

Časové složitosti mohou být různé. Některé mají i pojmenování

Konstantní – Θ (1)

Konstantní složitost nám říká, že na počtu dat nezáleží, algoritmus bude trvat stále stejně. Říká se mu také dokonalý / perfektní algoritmus. Nemusí být ovšem vždy výhrou, pokud bude trvat i tak pět let.

Příklad takového algoritmu může být například skok na index v poli – Je jedno, kolik bude prvků, první bude vždy první, osmý vždy osmý.

Logaritmická - Θ (Log n)

Logaritmický algoritmus je algoritmus „Rozděl a panuj“. Dá se pochopit, že takový algoritmus si rozdělí prvky jak potřebuje, příklad může být hledání jména v telefoním seznamu. Nemusíme kontrolovat každé jméno, když víme, že příjmení začíná na určité písmeno. Rozdělíme na polovinu a zjistíme, v jaké polovině leží toto jméno.

Příkladem je například binární vyhledání. Na to je potřeba setříděné pole prvků – [1,2,3,4,5,6,7]. Následně vezme střed jako číslo 4 – A pokud je hodnota menší, rozpůlí seznam a vezme si [1,2,3] a pokud je větší, tak uzme [5,6,7]. Pokud je hodnota 4, ukončí se jako správně nalezený. Pokud ne, následně znovu hledá střed a rozlišuje, jestli je menší nebo větší do té doby, dokud nerozdělní pole na samostatnou hodnotu, která bude naše správná.

Odmocninová - Θ (√n)

Prohledávání prvků v k-dimenzionálním stromu.

Lineární – Θ(n)

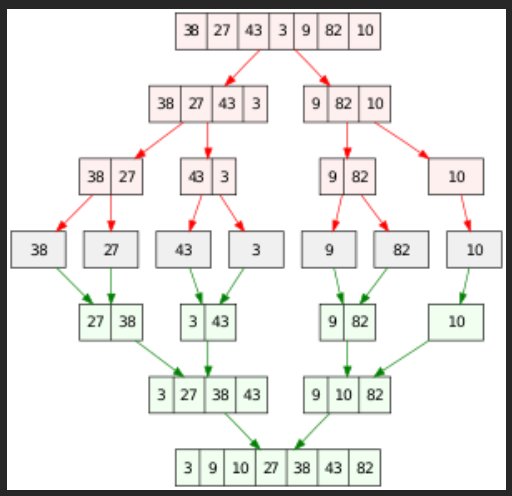
Lineární časová složitost je taková složitost, kde pro deset prvků se provádí deset operací, pro sto prvků sto operací a dále.

Dobrým příkladem může být vypsání všech prvků v poli. Tolikrát, kolik je prvků, se prvek vypíše.

Pokud sto tisíc prvků se vypíše za sekundu, milion prvků se vypíše za deset sekund.

Linearitmická Θ(n log n)

Tato lineární složitost je známá jako složitost Merge Sortu a Quick sortu – řadícího algoritmu. I tento algoritmus je známý jako „Rozděl a panuj“. Tento algoritmus vezme nesetříděné pole a dělí je na poloviny tak dlouho, dokud nejsou pouze dvojice. Následně dvojici seřadí a poté všechny dvojice spolu s další dvojicí spojuje tak dlouho, dokud nemá celé pole.



Kvadratická – Θ(n^2)

Touto časovou složitostí je známý například Bubble sort nebo Insertion sort. Nebo také dva for cykly v sobě ;)

Bubble sort vezme první dva prvky, porovná, který je větší, načež jej posune blíže ke konci. Poté vezme onen prvek a porovná ho s dalším A takhle provádí tak dlouho, dokud nedojde největší prvek na poslední místo. Poté jede znovu od začátku, takto dojede předposlední nejvyšší na předposlení místo. Proto je jeho složitost kvadratická – Pro sedm prvků sedmkrát proběhne sedm míst. Ano, projíždí i ty prvky, u kterých si je jistý, že jsou seřazené na konci správně.

Insertion sort je jednoduchý, nejlepší z kvadratických algoritmů, efektivní na malých datech. Insertion sort postupně prochází všechny prvky a každý nesetříděný prvek zařadí mezi dva prvky tam, kam patří.

Faktoriálová – Θ(n!)

Faktoriálová je na grafech udávána vždy jako nejhorší možnost, i když jsou ještě horší. Jedná se o řešení čehokoliv pomocí hrubé síly – Brute force. Příklad faktoriálového algoritmu je například řešení problému obchodního cestujícího, kde postupně projdeme absolutně všechny kombinace cest, které existují, a poté vybere tu nejsnazší.

Asymptonická paměťová složitost

Konstantní – Θ (1)

Příkladem může být for cyklus, kde tiskneme „i“ . V tomto případě se totiž s pamětí neděje nic jiného, než že jsme jednou vytvořili proměnnou i.

```
for (int i = 0; i < 10; i++) {
System.out.println(i);
}
```

Pozor! I v tomto případě bude konstantní paměťová složitost 1, protože se nevytvoří vždy nová proměnná, ale jenom se její obsah změní, což paměť nebere, jen upraví.

```
for (int i = 0; i < 10; i++) {
int i = 0
}
```

Lineární – Θ(n)

Tato paměťová složitost může být příkladem rekurzivního řešení faktoriálu. Pokud máme například faktoriál 6, je to 6x5x4x3x2x1 – Tudíž pro faktoriál šesti potřebujeme šestkrát uchovat největší hodnotu. Další příklad můžeme být například přidávání prvků do pole ve for cyklu.

Můžeme hodně spekulovat, že se jedná o O(1) či O(n), jelikož teoreticky deklarujeme pole, které už hned zabírá pamět – Konkrétně 40 bytů a pak do něj jen hodnoty vkládáme. Jednalo by se ovšem o ArrayList, byla by jasně lineární

```
int[] array = new int[10]
for (int i = 0; i < 10; i++) {
array[i] = i;
}
```

Kvadratická - $\Theta(n^2)$

Například násobení matic, jelikož se jedná o dvojdimenzionální pole.