Сложност на алгоритми

Сложност на алгоритъм – груба преценка за броя операции, които ще се извършат по време на изпълнение на алгоритъма в зависимост от входните данни.

Времева сложност – оценка на времето за изпълнение на алгоритъма в зависимост от входните данни. Обикновено се измерва при обем данни, клонящ към безкрайност ($n \to \infty$). Колкото по-висока е сложността, толкова по-бавен („лош“) е алгоритъмът.

Съществуват и други видове сложност, най-популярната от които е тази по памет, но те са обект на друго изследване.

Асимптотично сравнение – сравняване на графиките на функциите в безкрайност. Използваме следните означения:

- $F > G$ – F нараства по-бързо от G
- $F = G$ – F нараства еднакво бързо с G
- $F < G$ – F нараства по-бавно от G.

Важно уточнение е, че за малки стойности на n, функциите могат да бъдат в различни отношения една спрямо друга. Пример за това са $F=n$ и $G=n.\log(n)$ – за достатъчно малки стойности на n, $G < F$, но в безкрайност, където практически извършваме нашата оценка, $F > G$.

Асимптотични нотации:

$\Omega(F)$ – долна граница за F, най-добър случай.

$\Theta(F)$ – оценка на F за средния случай.

$O(F)$ – горна граница за F, най-лош случай.

Обикновено при оценката на алгоритми разглеждаме O-нотацията, т.е. най-лошия случай – всичко останало се включва и е по-добро от тази оценка.

Правила за пресмятане на сложност[1]:

$O(F+G) = \max(O(F), O(G))$

$O(F.G) = O(F).O(G)$

$O(c.F) = O(F)$, където c е константа[2], $c>0$

$O(F) = O(G) \Leftrightarrow F = c.G$, където c е константа, $c>0$

Основни функции за определяне на сложността на алгоритми:

[1] Важи за всички нотации

[2] Когато $n \to \infty$, константите се пренебрегват

Ще разгледаме някои основни асимптотични класове:

1. **O(c)** или **O(1)** – константна сложност, c - константа

   Това е почти пренебрежимо влияние върху сложността, обикновено елементарна операция, независеща от обема на входните данни.

   Примери:

   - Деклариране, въвеждане, извеждане на променливи
   - Оператора if
   - Извикване на функция

2. **O(log(n))** – логаритмична сложност[3]

   Най-ниската, зависеща от n, сложност, рядко практически достижима за алгоритъм.

   Примери:

   - двоично търсене
   - for(int i=1; i<=n; i*=2){} // променливата на цикъла расте експоненциално, т.е. времето му за изпълнение става логаритмично

3. **O(n)** – линейна сложност

   Единично обхождане на входните данни.

   Примери:

   - Обхождане на масив
   - Линейно търсене

4. **O(n.log(n))** – линейно-логаритмична сложност

   Примери:

   - Merge Sort
   - Средния случай на Quick Sort

5. **O(n^2)** – квадратична сложност

   Примери:

   - Обхождане на матрица
   - Bubble Sort, Selection sort

6. **O(n^3)** – кубична сложност

[3] При оценка на алгоритми се има предвид $\log_2(n)$. Всички останали логаритми са сравними с него.

Обикновено тази сложност се разглежда като последната приемлива за практически приложим алгоритъм – ако той е по-бавен от това, трябва да се оптимизира.

Примери:

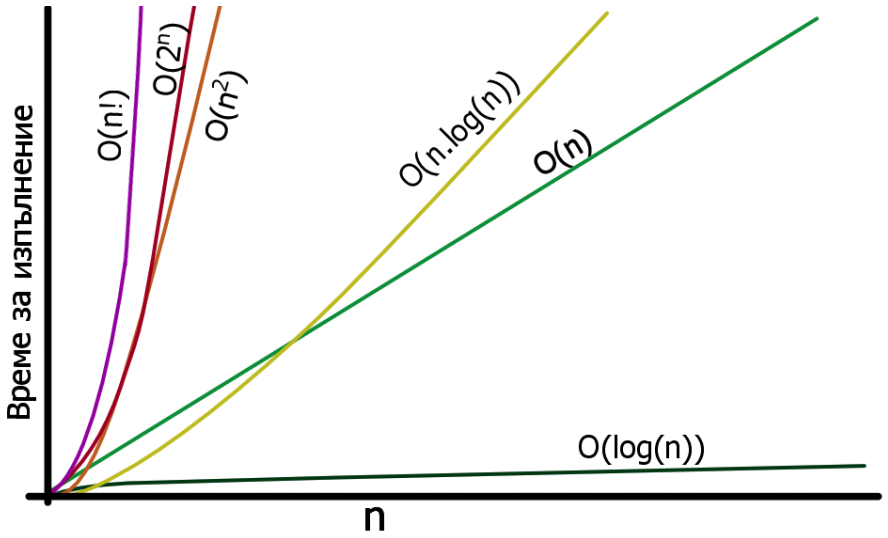
- Флойд-Уоршал, Белман-Форд

7. **O(2^n)** – експоненциална сложност

Пример:

- Ханойски кули

8. **O(n!)** – факториелна сложност
9. **O(n^n)**



Литература:

Асимптотична нотация (статия) | Алгоритми | Кан Академия

infO(N)::Сложност на алгоритми