Сложност на алгоритми

**Времева сложност** – оценка на времето за изпълнение на алгоритъма в зависимост от входните данни. Обикновено се измерва при обем данни, клонящ към безкрайност (n→∞). 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.

Асимптотични нотации:

Ω(F) – долна граница за F, най-добър случай.

Θ(F) – оценка на F за средния случай.

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

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

Правила за пресмятане на сложност[[1]](#footnote-2):

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]](#footnote-3)

O(F) = O(G)  F = c.G, където c е константа

Основни функции за определяне на сложността на алгоритми:

1. **O(c)** или **O(1)** – константна сложност

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

1. **O(log(n))** – логаритмична сложност[[3]](#footnote-4)

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

1. **O(n)** – линейна сложност

Единично обхождане на входните данни.

Примери:

* Обхождане на масив
* Линейно търсене

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

Примери:

* Merge Sort
* Средния случай на Quick Sort

1. **O(n2)** – квадратична сложност

Примери:

* Обхождане на матрица
* Bubble Sort, Selection sort

1. **O(n3)** – кубична сложност

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

Примери:

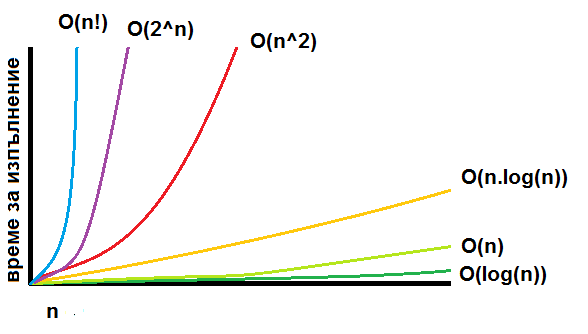
* Флойд-Уоршал, Белман-Форд

1. **O(2n)** – експоненциална сложност

Пример:

* Ханойски кули

1. **O(n!)** – факторелна сложност
2. **O(nn)**

****

1. Важи за всички нотации [↑](#footnote-ref-2)
2. Когато n→∞, константите се пренебрегват [↑](#footnote-ref-3)
3. При оценка на алгоритми се има предвид log2(n). Всички останали логаритми са сравними с него. [↑](#footnote-ref-4)