Сложност на алгоритмите

SAP GEEKYCAMP 4.0

Божин Кацарски
6.09.17

Алгоритъм

- Поредица от инструкции за решаване на даден проблем, често свързан с изчисление или обработка на данни
- Може да се изрази
  - описателно
  - чрез псевдокод
  - на реален език за програмиране
- Краен алгоритъм
- Терминиращ алгоритъм
- Входни и изходни данни
- Ефективност

Swapping two variables without using another variable

Algorithm
Step1: Start
Step2: Read the value of a, b
Step3: a = a + b
b = a - b
a = a - b
Step4: Print the value of a and b
Step5: Stop

Анализ на алгоритъм (по време)

- Имаме два алгоритъма, които решават даден проблем
- Как избираме кой от тях е по-ефективен (бърз)?
- Можем
  - да пускаме и двата алгоритъма с различни входни данни и да измерваме времето им за изпълнение
  - да хвърляме чоп (?)
  - да сравним техните функции на растежа
    - Става много по-бързо отколкото да ги пускаме много пъти
    - Не е нужно първо да сме имплементирали алгоритмите
    - Не ни трябва монета ☺

Функция на растежа

- Функцията на растежа показва как се изменя времето за изпълнение при различни входни данни
- Първо броим елементарните операции, които извършва нашия алгоритъм
  - Например a=b;, i++; или return 42;
  - Алгоритъма от слайда преди малко имаше 5 елементарни операции
- Всяка операция има някаква „цена“ – време, за което се изпълнява
- После изразяваме този брой като функция на големината на проблема
  - Например ако проблема е сортировка на масив, големината му е дължината на масива
- Тази функция наричаме функция на растежа
- Полезна е, защото по нея сравняваме кой от два алгоритъма е по-ефективен

Константна сложност

```
count = count + 1; // c1
sum = sum + count; // c2
```

Total cost: c1 + c2

И двете елементарни операции отнемат известно време, но то е константно.

Константна сложност

```
if (n < 0) {                // c1        //тук n е входните данни
    absolute = -n;      // c2
} else {
    absolute = n;       // c3
}
```

Total cost: c1 + max(c2, c3)

Независимо от големината на входа n, алгоритъма завършва за константно време

Линейна сложност

```
i = 0;                  // c1
sum = 0;                // c2
while (i <= n) {        // c3      //тук n е входните данни
    sum = sum + i;      // c4
    i++;                // c5
}
```

Total cost: c1 + c2 + (n + 1) * c3 + n * (c4 + c5)

- Броят операции расте пропорционално на n
- Когато n нарастне 10 пъти, необходимото време също ще нарастне 10 пъти
- Линейна зависимост между n и Total cost

Квадратна сложност

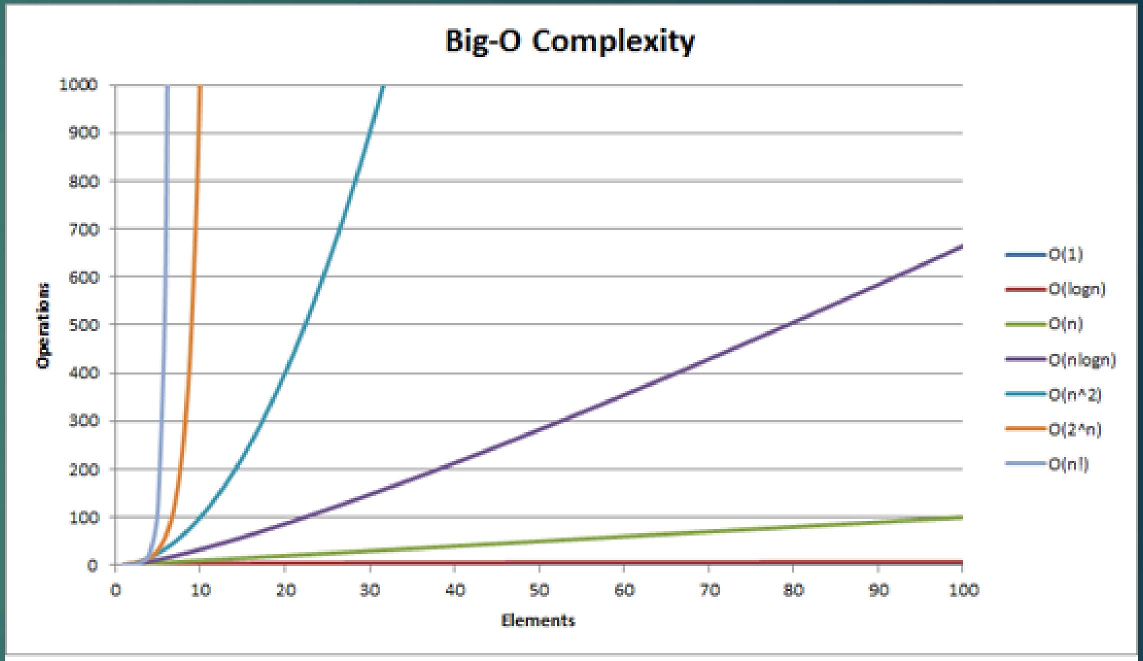
```
i = 0;                          // c1
while (i < n) {                 // c2
    j = 0;                      // c3
    while (j < n) {             // c4
        sum = sum + i;          // c5
        j++;                    // c6
    }
    i++;                        // c7
}
```

Total cost = c1 + (n+1)*c2 + n*(c3 + (n+1)*c4 + n*(c5+c6) + c7)

- Когато n нарастне 10 пъти, необходимото време ще нарастне 100 пъти
- Квадратна зависимост между n и Total cost

Голямо-О нотация

- Сравнява функции на растежа на алгоритми
- Най-известната такава нотация (има още няколко)
- Нека $f(n) = n^2$ е функция на растежа
- O(f(n)) са всички функции, които растат не по-бързо от f, например
  - $g(n) = 10n + 11,$
  - $h(n) = \log_2 n$
- Тоест O ни дава *горна граница*



Константна сложност

```
count = count + 1; // takes some time c1 but it is constant
sum = sum + count; // c2
```

Total cost: c1 + c2

$f \in O(1)$

Най-добрата възможна сложност!

Правило: При последователни операции събираме

Константна сложност

```
if (n < 0) {                    // c1
    absolute = -n;     // c2
} else {
    absolute = n;      // c3
}
```

Total cost: c1 + max(c2, c3)

$f \in O(1)$

Правило: При разклонения, взимаме max

Линейна сложност

```
i = 0;                      // c1
sum = 0;                    // c2
while (i <= n) {            // c3
      sum = sum + i;        // c4
      i++;                  // c5
}
```

Total cost: c1 + c2 + (n + 1) * c3 + n * (c4 + c5)

Линейна зависимост между n и Total cost

$f \in O(n)$

Правило: При влагане умножаваме

Квадратна сложност

```
i = 0;                      // c1
while (i < n) {             // c2
    j = 0;                  // c3
    while (j < n) {         // c4
        sum = sum + i;      // c5
        j++;                // c6
    }
    i++;                    // c7
}
```

Total cost = c1 + (n+1)*c2 + n*(c3 + (n+1)*c4 + n*(c5+c6) + c7)

Квадратна зависимост между n и Total cost

$f \in O(n^2)$

Правило: При влагане умножаваме

Логаритмична сложност

```
i = 1;
while (i < n) {
    i = i * 2;
}
```

i = 1, 2, $2^2, 2^3, \dots, 2^{\log_2 n - 1}, 2^{\log_2 n}$

Общо $\log_2 n$ стъпки

Когато n нарастне два пъти, броят стъпки нараства с 1 => логаритмична зависимост

\`Ен лог ен\` сложност - $n * \log n$

linearithmic complexity

```
for(i = 0, i < n, i++) {      // linear
    j = 3 * n;
    while(j > 0) {            // logaritmic
        j = j / 5;
    }
}
```

- Още една често срещана сложност, но без добър превод на български
- Този път
  - Основата на логаритъма ще е 5
  - Пред n има константа 3

Константите нямат значение*

- Дотук дадохме само неформална дефиниция на функциите на растежа
- Всъщност зад тях стои много и добре дефинирана теория
- От нея можем да видим, че
  - Константните множители нямат значение
  - Основата на логаритъма няма значение
- Например
  - $O(100n^2) = O(5n^2 + 40n + 10)$, пишем само $O(n^2)$
  - $O(\log_{10} n) = O(\log_2 n)$, пишем само $O(\log n)$
- За щастие тази теория не ни е необходима в момента ☺

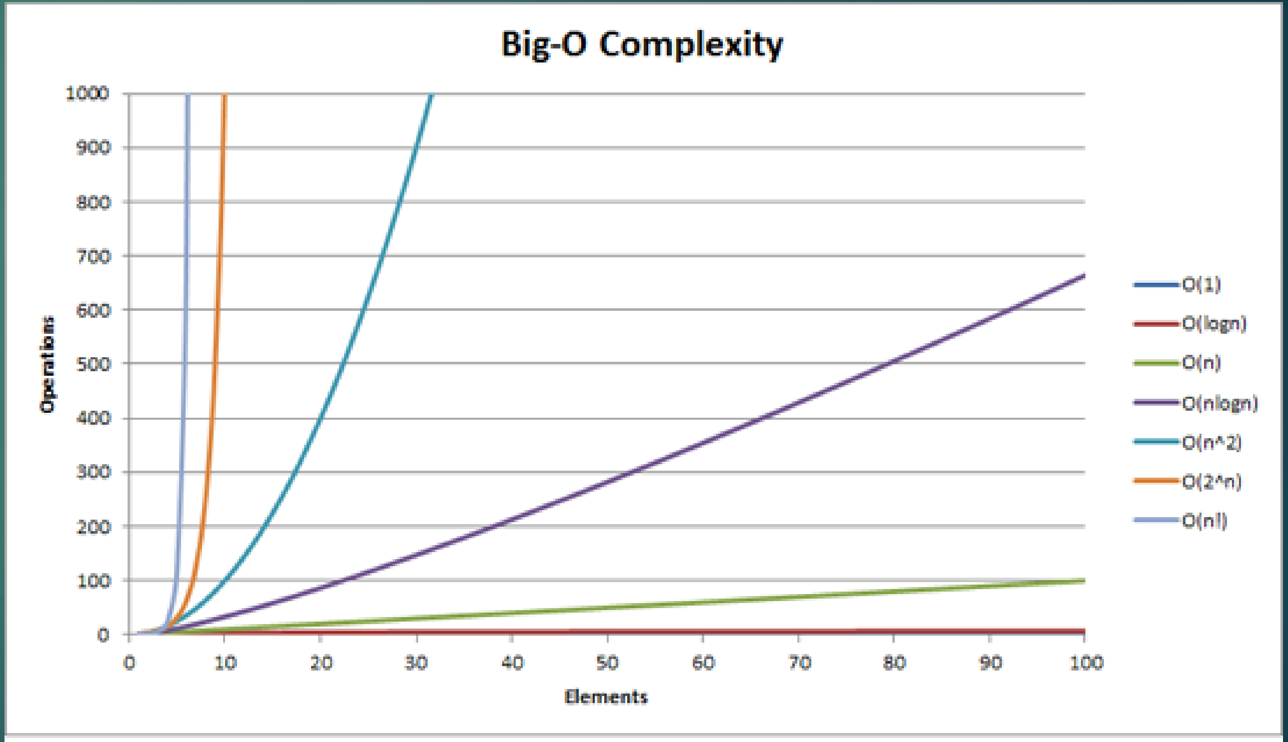
* Понякога имат: $O(2^n) \neq O(3^n)$

Често срещани времеви сложности

Common Growth Rates

Function	Growth Rate Name
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	
N^2	Quadratic
N^3	Cubic
2^N	Exponential

CENG 213 Data Structures 13



Анализ на алгоритми в практиката

- Когато пишем алгоритми и операции в структури от данни трябва да оценяваме времевата им сложност и преценяваме/сравняваме колко са ефективни те
- Обикновено
  - $O(1)$, $O(\log n)$ е бързо
  - $O(n)$, $O(n * \log n)$ става
  - $O(n^2)$, $O(n!)$, $O(2\text{^}n)$ е бавно
- Днес ще
  - се интересуваме от времева сложност
  - се стремим към $O(1)$ и $0(\log n)$ сложност на операциите

Ресурси по темата

- Много добра и достъпна статия http://www.informatika.bg/lectures/complexity
  - В сайта има още много други лекции, важни и полезни са особено за състезателите по информатика
- Сложности на известни структури данни (ще ги видим след малко) http://bigocheatsheet.com/
- Голямо-О нотацията в Уикипедия https://en.wikipedia.org/wiki/Big_O_notation