

Algorytmy i struktury danych

Złożoność obliczeniowa Podstawy

Aleksander Lamża
ZKSB · Instytut Informatyki
Uniwersytet Śląski w Katowicach

aleksander.lamza@us.edu.pl

- Złożoność obliczeniowa algorytmów
- W czym problem?
- Złożoność praktyczna T
- Złożoność teoretyczna O

Złożoność obliczeniowa algorytmów

W dużym uproszczeniu szacowanie **złożoności obliczeniowej** ma dać odpowiedź na pytanie:

Który z algorytmów jest efektywniejszy?



wykonujących to samo zadanie

Nie chodzi jednak o określenie czasu wykonania, ponieważ zależy on od wielu „zewnętrznych” czynników (choćby sprzętowych).

Chodzi raczej o określenie **klasy** złożoności obliczeniowej danego algorytmu.

Ale czy naprawdę różnice w zastosowanych podejściach (algorytmach) mogą być na tyle duże, by przejmować się tą złożonością?

Zobaczmy...

W czym problem?

Parametrem decydującym o czasie wykonania algorytmu jest najczęściej **rozmiar danych wejściowych (n)**.



„Rozmiar” ma tutaj dosyć szerokie znaczenie.
W przypadku algorytmu operującego na tablicy będzie to **liczba elementów** tablicy, ale dla algorytmu numerycznego (np. wyliczającego silnię) będą to **wartości** danych wejściowych.

Dane wejściowe mogą być przetwarzane na wiele różnych sposobów, a operacje te są w mniejszym bądź większym stopniu „czułe” na rozmiar tych danych.

Przeprowadźmy mały eksperyment.

Założmy, że wykonanie operacji jednostkowej (na „jednej danej”) trwa jedną mikrosekundę (0,000001).

Zobaczmy, ile czasu zajęłoby algorytmom różnych klas złożoności przetworzenie kilku danych...

W czym problem?

Oto wyniki:

Rozmiar danych wejściowych (np. liczba elementów tablicy)

	10	20	30	40	50	60
n	0,00001 s	0,00002 s	0,00003 s	0,00004 s	0,00005 s	0,00006 s
n²	0,0001 s	0,0004 s	0,0009 s	0,0016 s	0,0025 s	0,0036 s
n³	0,001 s	0,008 s	0,027 s	0,064 s	0,125 s	0,216 s
2ⁿ	0,001 s	1 s	18 min	13 dni	36 lat	366 w.
3ⁿ	0,59 s	58 min	6 lat	3855 w.	10 ⁸ w.	10 ¹³ w.
n!	3,6 s	768 w.	10 ¹⁶ w.	10 ³² w.	10 ⁴⁹ w.	10 ⁶⁶ w.

Klasa złożoności algorytmu

I co, robi wrażenie?

W czym problem?

Rozmiar danych wejściowych (np. liczba elementów tablicy)

	10	20	30	40	50	60
n	0,00001 s	0,00002 s	0,00003 s	0,00004 s	0,00005 s	0,00006 s
n²	0,0001 s	0,0004 s	0,0009 s	0,0016 s	0,0025 s	0,0036 s
n³	0,001 s	0,008 s	0,027 s	0,064 s	0,125 s	0,216 s
2ⁿ	0,001 s	1 s	18 min	13 dni	36 lat	366 w.
3ⁿ	0,59 s	58 min	6 lat	3855 w.	10 ⁸ w.	10 ¹³ w.
n!	3,6 s	768 w.	10 ¹⁶ w.	10 ³² w.	10 ⁴⁹ w.	10 ⁶⁶ w.

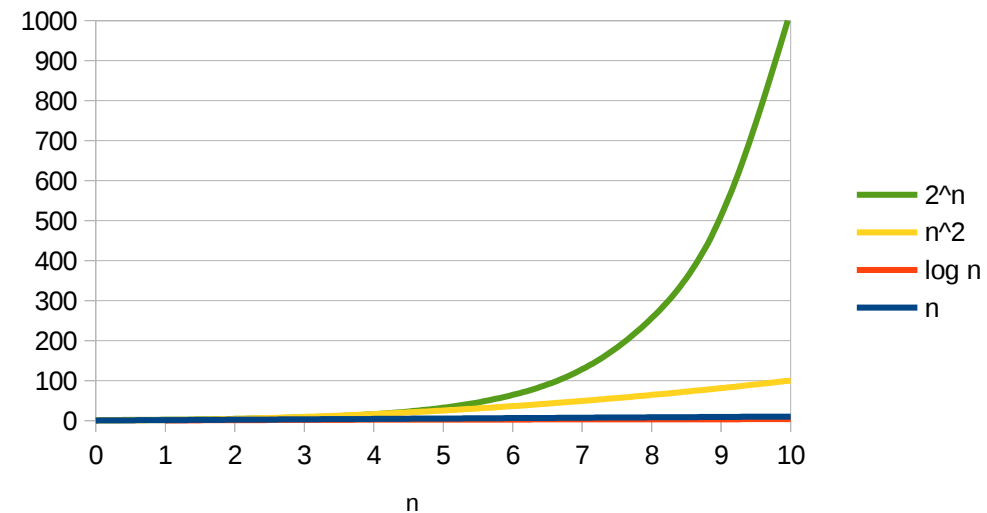
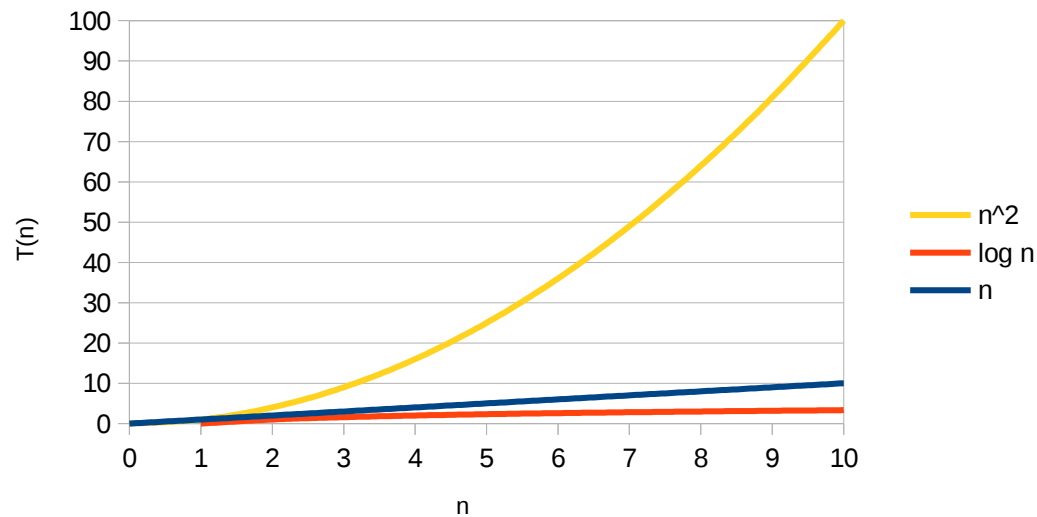
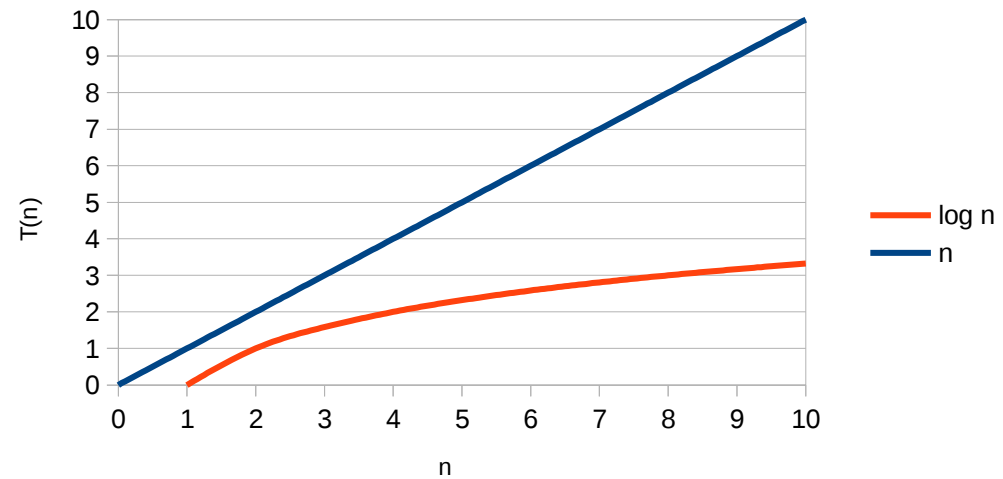
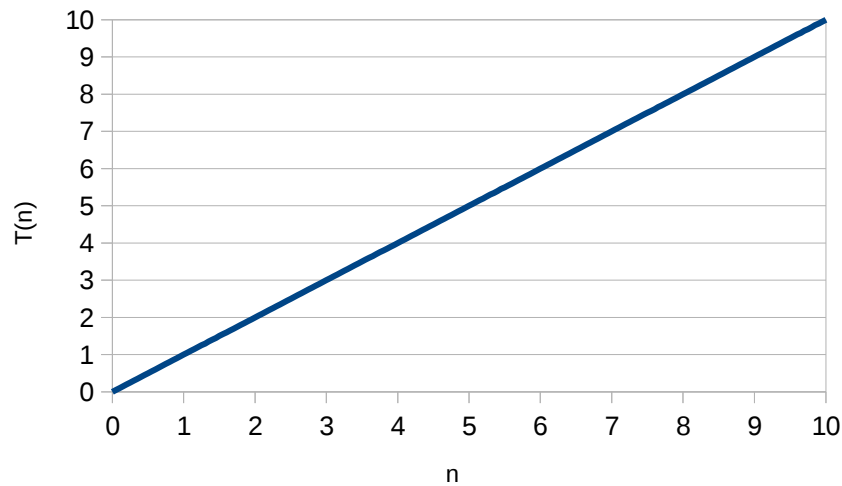
Klasa złożoności algorytmu

Stosowanie niektórych algorytmów przy określonym rozmiarze danych nie ma sensu!

Trzeba szukać alternatywnych rozwiązań, a do tego przydaje się umiejętność szacowania złożoności obliczeniowej algorytmów.

W czym problem?

A tutaj możemy porównać na wykresach cztery przypadki złożoności:
 n , $\log n$, n^2 i 2^n :



Złożoność praktyczna





Zaczniemy od przeanalizowania iteracyjnego algorytmu liczenia silni:

```
int silnia(int n) {  
    int wynik = 1;  
  
    while (n > 0) {  
        wynik = wynik * n;  
        n = n - 1;  
    }  
  
    return wynik;  
}
```

Obliczymy tzw. **złożoność praktyczną** oznaczaną przez **$T(n)$** .

Złożoność praktyczna

Weźmiemy pod uwagę czasy wykonania operacji przypisania (t_a) i porównania (t_c):

```
int silnia(int n) {  
    int wynik = 1;   $t_a$   
  
    while (n > 0) {   $t_c$   
        wynik = wynik * n;   $t_a$   
        n = n - 1;   $t_a$   
    }  
  
    return wynik;  
}
```

Teraz trochę matematyki:

$$T(n) = t_a + t_c + n \cdot (t_c + t_a + t_a)$$

$$T(n) = t_a + t_c + n \cdot t_c + 2n \cdot t_a$$

$$T(n) = t_a \cdot (2n + 1) + t_c \cdot (n + 1)$$

A jak będzie z algorytmem rekurencyjnym?

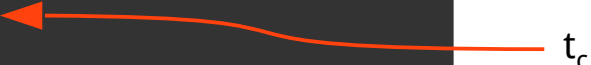
Złożoność praktyczna

Dla przypomnienia, rekurencyjny algorytm liczenia silni wygląda tak:

```
int silnia(int n) {  
    if (n == 0) {  
        return 1;  
    } else {  
        return n * silnia(n-1);  
    }  
}
```

Tym razem weźmiemy pod uwagę tylko czas wykonania operacji porównania (t_c),
a pominiemy przypisania (t_a):

```
int silnia(int n) {  
    if (n == 0) {  
        return 1;  
    } else {  
        return n * silnia(n-1);  
    }  
}
```



Złożoność praktyczna

Ponieważ jest to algorytm rekurencyjny, czas jego wykonania również możemy zapisać rekurencyjnie:

$$\begin{aligned}T(0) &= t_c \\ T(n) &= t_c + T(n-1) \quad \text{dla } n \geq 1\end{aligned}$$

Możemy to rozpisać w następujący sposób:

$$\begin{aligned}T(n) &= t_c + T(n-1) \\ T(n-1) &= t_c + T(n-2) \\ T(n-2) &= t_c + T(n-3) \\ &\vdots \\ T(1) &= t_c + T(0) \\ T(0) &= t_c\end{aligned}$$

Po zsumowaniu stronami mamy:

$$T(n) + T(n-1) + T(n-2) + \dots + T(0) = t_c \cdot (n+1) + T(n-1) + T(n-2) + \dots + T(0)$$

A to daje:

$$T(n) = t_c \cdot (n+1)$$

Złożoność praktyczna

Z analizy algorytmu iteracyjnego wyszło nam:

$$T(n) = t_a \cdot (2n+1) + t_c \cdot (n+1)$$

a z rekurencyjnego:

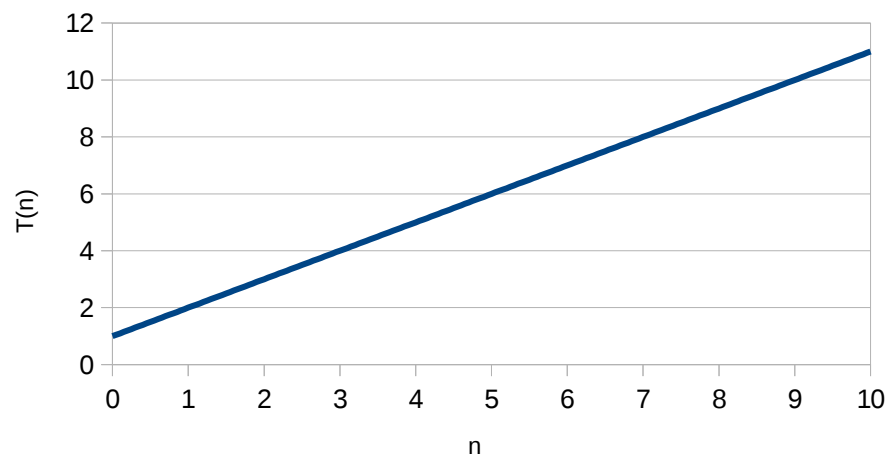
$$T(n) = t_c \cdot (n+1)$$

Jeżeli w pierwszym przypadku usuniemy składnik związany z t_a ,
otrzymamy identyczną złożoność obu algorytmów.

Jeśli uznamy, że t_c jest operacją jednostkową i zastąpimy ją przez 1, otrzymamy:

$$T(n) = n + 1$$

Jest to więc typowa **funkcja liniowa**:



Złożoność teoretyczna

Zazwyczaj nie zależy nam na dokładnym określeniu liczby wykonanych operacji.

Chcemy raczej znać **klasę algorytmu**, czyli **funkcję matematyczną, która w największym stopniu wpływa na czas wykonania** wyznaczony przez złożoność praktyczną.

Tę funkcję nazywa się **złożonością teoretyczną** i oznacza przez **O** (tzw. O-notacja).

Poniżej kilka przykładów przekształcenia złożoności praktycznej $T(n)$ na złożoność teoretyczną O :

$$T(n) = 10 \longrightarrow O(1)$$

$$T(n) = 10n+1 \longrightarrow O(n)$$

$$T(n) = n^2+3n+5 \longrightarrow O(n^2)$$

$$T(n) = 2^n+n^2+1 \longrightarrow O(2^n)$$

wzrost złożoności

Złożoność teoretyczna

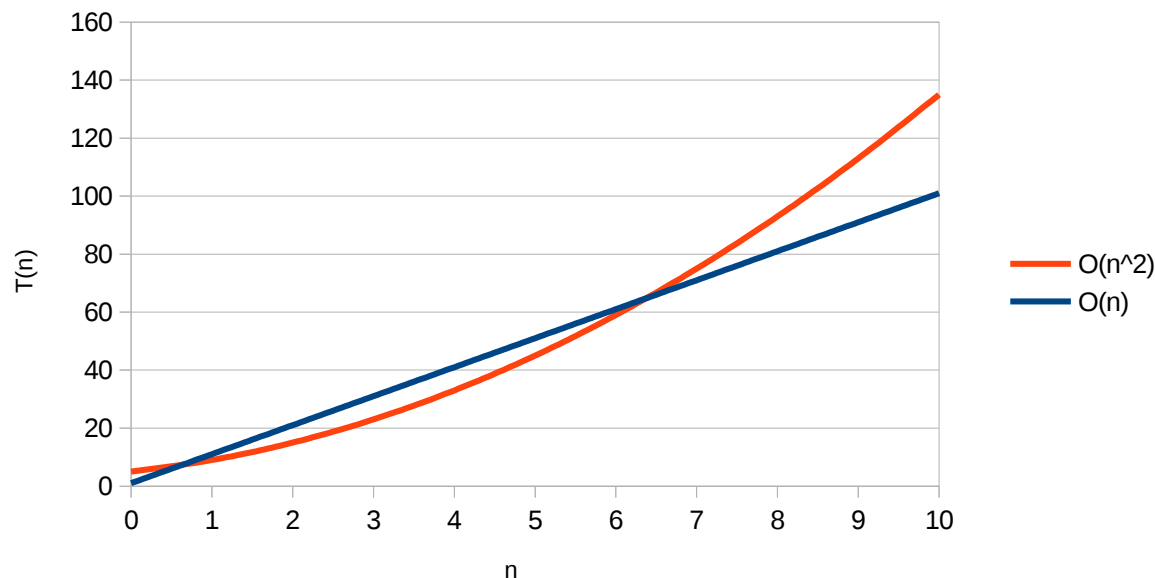
Trzeba sobie zdawać sprawę, że nie w każdym przypadku algorytm o większej złożoności będzie „wolniejszy” od tego o mniejszej złożoności.

Weźmy dwie funkcje z poprzedniego zestawienia:

$$O(n): T(n) = 10n + 1$$

$$O(n^2): T(n) = n^2 + 3n + 5$$

Okazuje się, że dla niektórych wartości n algorytm klasy $O(n)$ jest lepszy niż $O(n^2)$:



A teraz czas na eksperymenty.

Sprawdzimy w praktyce wybrane algorytmy i spróbujemy określić ich złożoność.