Algorytmy i Struktury Danych

Prof. dr hab. Stanisław Gawiejnowicz
Wydział Biologii UAM
Semestr letni 2022/2023

Plan wykładu nr 6

- Notacja asymptotyczna
- Problem sortowania
- Sortowanie a relacje porządku
- Elementarne algorytmy sortowania
  - przez wybieranie (ang. selection sort)
  - przez wstawianie (ang. insertion sort)
  - sortowanie bąbelkowe (ang. bubble sort)
- Przykład analizy algorytmu sortowania

Notacja asymptotyczna

- Notacja asymptotyczna służy do symbolicznego oznaczania **rzędu** szacowanych wielkości
- Szacowanymi wielkościami mogą być np. wartości funkcji, liczby operacji elementarnych, moce zbiorów itp.
- Notację asymptotyczną stosuje się wtedy, gdy podanie dokładnych wartości szacowanych wielkości jest trudne bądź niemożliwe

Notacja asymptotyczna

- Notacja asymptotyczna będzie stosowana do szacowania wartości pewnej funkcji, f(n), względem wartości innej funkcji, g(n)
- Funkcje f(n) i g(n) będą funkcjami całkowitych nieujemnych argumentów o wartościach rzeczywistych nieujemnych
- Podstawowe symbole notacji asymptotycznej:
  - $\Theta(f(n))$
  - $\Omega(f(n))$
  - $O(f(n))$

Notacja asymptotyczna

Symbol $\Theta(f(n))$

- $\Theta(g(n))$:= $\{f(n):\mathbb{N}\rightarrow\mathbb{R}^{+}\cup\{0\}$: istnieją stałe c_1, c_2 i n_0 takie, że $c_1 g(n) \le f(n) \le c_2 g(n)$ dla wszystkich $n \ge n_0\}$
- Asymptotyczne ogranicza funkcję od dołu oraz od góry
- Dla wszystkich $n \ge n_0$ funkcja $f(n)$ jest równa $g(n)$ z dokładnością do stałego współczynnika
- $g(n)$ jest **asymptotycznie dokładnym oszacowaniem** dla $f(n)$

Niech $f(n) = \frac{1}{2}n^2 - 3n$ $\quad c_1 = 1/14, \quad c_2 = 1/2, \quad n_0 = 7$

$$\frac{1}{14}n^2 \le \frac{1}{2}n^2 - 3n \le \frac{1}{2}n^2 \qquad f(n) = \Theta(n^2)$$

Dla każdego wielomianu $p(n) = \sum_{i=0}^{d} a_i n^i$ mamy $p(n) = \Theta(n^d)$

Notacja asymptotyczna

Symbol O(f(n))

- $O(g(n)) := \{f(n):\mathbb{N}\rightarrow\mathbb{R}^{+}\cup\{0\}$ takich, że $\exists\ c, n_0: f(n) \leq cg(n)\ \forall\ n \geq n_0\}$
- Asymptotyczne ogranicza funkcję od góry
- Oszacowanie nie musi być dokładne
- **Przykład:** $n = O(n)$, ale też $n = O(n^2)$

Symbol Ω(f(n))

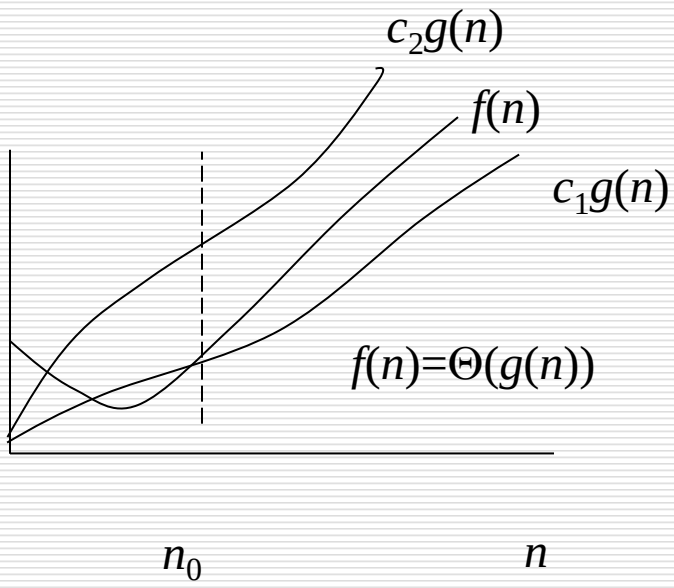
- $\Omega(g(n)) := \{f(n):\mathbb{N}\rightarrow\mathbb{R}^{+}\cup\{0\}$ takich, że $\exists\ c, n_0: 0 \leq cg(n) \leq f(n)\ \forall\ n \geq n_0\}$
- Asymptotyczne ogranicza funkcję od dołu

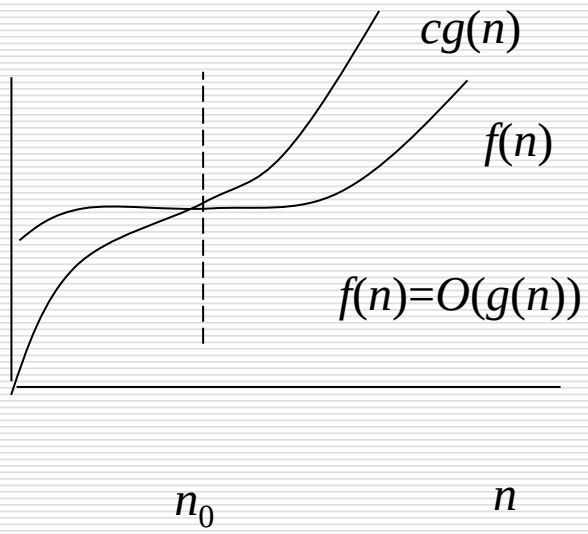
- **Dla każdych dwóch funkcji $f(n)$ i $g(n)$ zachodzi zależność**

$$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$$

Notacja asymptotyczna

- Graficzna ilustracja definicji $\Theta(f(n))$ oraz $O(f(n))$ wygląda tak





Problem sortowania

- Problem sortowania polega na ustawieniu sortowanych elementów w określonej kolejności
- Sortowanymi elementami mogą być liczby, napisy, wierzchołki grafu itp.
- Problem sortowania często występuje w praktyce, znanych jest wiele algorytmów sortowania

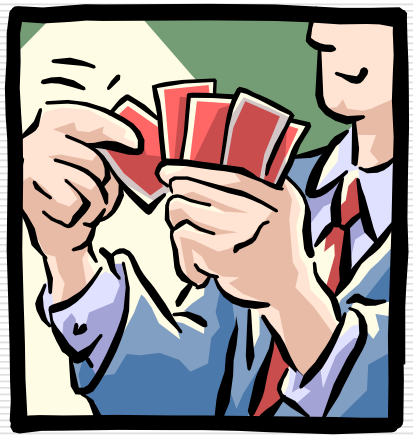
Problem sortowania

WEJŚCIE

ciąg *n* liczb

$a_1, a_2, a_3, \ldots, a_n$

2 5 4 10 7



WYJŚCIE

permutacja wejściowego ciągu liczb

$b_1, b_2, b_3, \ldots, b_n$

2 4 5 7 10

Poprawność algorytmu (wymagania dotyczące wyjścia)

Dla dowolnego danego wejścia algorytm kończy generuje wyjście:

- $b_1 \leq b_2 \leq b_3 \leq \ldots \leq b_n$
- ciąg $b_1, b_2, b_3, \ldots, b_n$ jest permutacją ciągu $a_1, a_2, a_3, \ldots, a_n$

Problem sortowania a relacje porządku

- Rozwiązanie problemu sortowania wymaga znajomości **relacji częściowego porządku**, która określa wynikowe ustawienie sortowanych elementów
- Relacja częściowego porządku jest **zwrotna**, **antysymetryczna** i **przechodnia**
- Definicja tej relacji zależy od typu sortowanych elementów

Problem sortowania a relacje porządku

- **Przykłady:**
- W przypadku, gdy sortowane elementy są liczbami stosujemy np. **relację ≤**
- W przypadku, gdy sortowane są napisy stosujemy np. **relację porządku leksykograficznego**
- W dalszym toku będziemy rozważać głównie sortowanie liczb

Problem sortowania a relacje porządku

- W omawianych dalej algorytmach sortowana jest tablica A[1..n] liczb, przyjmujemy, że jest ona globalna
- Posortowane liczby są ustawione **niemalejąco** (od najmniejszej wartości do największej wartości)
- **Elementem minimalnym** będzie nazywany element najmniejszy (w sensie przyjętej relacji porządku)

Sortowanie przez wybieranie

- Sortowanie przez wybieranie polega na
  - wielokrotnym wybieraniu elementu minimalnego
  - wstawianiu go na odpowiednie miejsce w tablicy w części posortowanej

Sortowanie przez wybieranie

```
procedure SELECTION-SORT(n, A[1..n])

for i=1 to n-1 do
  naj=i
  for j=i+1 to n do
    if A[j]<A[naj]
    then naj=j
  if i<>naj
  then A[i] <-> A[naj] // wymiana
return
```

Sortowanie przez wybieranie

Przykład: n=9, A=[3, 8, 1, 2, 5, 13, 20, 4, 9]

1: **1**, 8, 3, 2, 5, 13, 20, 4, 9
2: **1**, **2**, 3, 8, 5, 13, 20, 4, 9
3: **1**, **2**, **3**, 8, 5, 13, 20, 4, 9
4: **1**, **2**, **3**, **4**, 5, 13, 20, 8, 9
5: **1**, **2**, **3**, **4**, **5**, 13, 20, 8, 9
6: **1**, **2**, **3**, **4**, **5**, **8**, 20, 13, 9
7: **1**, **2**, **3**, **4**, **5**, **8**, **9**, 13, 20
8: **1**, **2**, **3**, **4**, **5**, **8**, **9**, **13**, 20
9: **1**, **2**, **3**, **4**, **5**, **8**, **9**, **13**, **20**

Sortowanie przez wstawianie

- Sortowanie przez wstawianie polega na
  - wielokrotnym wstawianiu elementu minimalnego
  - wstawianiu go na odpowiednie miejsce w tablicy w części posortowanej
- Elementy są sortowane **w miejscu**
- **Sortowanie przez wstawianie jest przykładem metody przyrostowej:** mając posortowaną tablicę A[1..j-1] wstawiamy element A[j] we właściwe miejsce otrzymując posortowaną tablicę A[1..j]

Sortowanie przez wybieranie

```
procedure INSERTION-SORT(n, A[1..n])

for j=2 to n do
  naj=A[j]
  i=j-1
  while (i>0) and (A[i]>naj) do
    A[i+1]=A[i]
    i=i-1
  A[i+1] <-> naj // wymiana
return
```

Sortowanie przez wstawianie

Przykład: n=9, A=[3, 8, 1, 2, 5, 13, 20, 4, 9]

1: **3**, 8, 1, 2, 5, 13, 20, 4, 9
2: **3**, **8**, 1, 2, 5, 13, 20, 4, 9
3: **1**, **3**, **8**, 2, 5, 13, 20, 4, 9
4: **1**, **2**, **3**, **8**, 5, 13, 20, 4, 9
5: **1**, **2**, **3**, **5**, **8**, 13, 20, 4, 9
6: **1**, **2**, **3**, **5**, **8**, **13**, 20, 4, 9
7: **1**, **2**, **3**, **5**, **8**, **13**, **20**, 4, 9
8: **1**, **2**, **3**, **4**, **5**, **8**, **13**, **20**, 9
9: **1**, **2**, **3**, **4**, **5**, **8**, **9**, **13**, **20**

Sortowanie bąbelkowe

- Sortowanie bąbelkowe polega na
  - wielokrotnym porównywaniu par elementów
  - zamianie elementów porównywanej pary, jeśli są w niewłaściwej kolejności
- Sortowanie bąbelkowe opiera się na spostrzeżeniu, iż jeżeli w danej iteracji będziemy przeglądać tablicę liczb po kolei zamieniając miejscami liczby będące w odwrotnym porządku, to po zakończeniu tej iteracji największa liczba znajdzie się na końcu tablicy

Sortowanie bąbelkowe

```
procedure BUBBLE-SORT-1(n, A[1..n]) // z wartownikiem

wart=n  // wartownik
while wart <> 0 do
  k=0
  for i=1 to wart-1 do
    if A[i]>A[i+1]
    then A[i] <-> A[i+1] // wymiana
              k=i
    wart=k // aktualizacja wartownika
return
```

Sortowanie bąbelkowe

```
procedure BUBBLE-SORT-2(n, A[1..n]) // bez wartownika

for i=n downto 2 do
  for j=1 to i-1 do
    if A[j]>A[j+1]
    then A[j] <-> A[j+1] // wymiana
return
```

Sortowanie bąbelkowe

Przykład: n=9, A=[3, 8, 1, 2, 5, 13, 20, 4, 9]

1: 3, 8|, 1, 2, 5, 13, 20, 4, 9
1: 3, 1, 8|, 2, 5, 13, 20, 4, 9
1: 3, 1, 2, 8|, 5, 13, 20, 4, 9
1: 3, 1, 2, 5, 8|, 13, 20, 4, 9
1: 3, 1, 2, 5, 8, 13|, 20, 4, 9
1: 3, 1, 2, 5, 8, 13, 20|, 4, 9
1: 3, 1, 2, 5, 8, 13, 4, 20|, 9
1: 3, 1, 2, 5, 8, 13, 4, 9, **20**|

Sortowanie bąbelkowe

Przykład: n=9, A=[3, 8, 1, 2, 5, 13, 20, 4, 9]

2: 1, 3|, 2, 5, 8, 13, 4, 9, **20**
2: 1, 2, 3|, 5, 8, 13, 4, 9, **20**
2: 1, 2, 3, 5|, 8, 13, 4, 9, **20**
2: 1, 2, 3, 5, 8|, 13, 4, 9, **20**
2: 1, 2, 3, 5, 8, 13|, 4, 9, **20**
2: 1, 2, 3, 5, 8, 4, 13|, 9, **20**
2: 1, 2, 3, 5, 8, 4, 9, **13**|, **20**

Sortowanie bąbelkowe

Przykład: n=9, A=[3, 8, 1, 2, 5, 13, 20, 4, 9]

3: 1, 2|, 3, 5, 8, 4, 9, **13**, **20**
3: 1, 2, 3|, 5, 8, 4, 9, **13**, **20**
3: 1, 2, 3, 5|, 8, 4, 9, **13**, **20**
3: 1, 2, 3, 5, 8|, 4, 9, **13**, **20**
3: 1, 2, 3, 5, 4, 8|, 9, **13**, **20**
3: 1, 2, 3, 5, 4, 8, **9**|, **13**, **20**

Sortowanie bąbelkowe

Przykład: n=9, A=[3, 8, 1, 2, 5, 13, 20, 4, 9]

4: 1, 2|, 3, 5, 4, 8, **9**, **13**, **20**

4: 1, 2, 3|, 5, 4, 8, **9**, **13**, **20**

4: 1, 2, 3, 5|, 4, 8, **9**, **13**, **20**

4: 1, 2, 3, 4, 5|, 8, **9**, **13**, **20**

4: 1, 2, 3, 4, 5, **8**|, **9**, **13**, **20**

5: 1, 2|, 3, 4, 5, **8**, **9**, **13**, **20**

5: 1, 2, 3|, 4, 5, **8**, **9**, **13**, **20**

5: 1, 2, 3, 4|, 5, **8**, **9**, **13**, **20**

5: 1, 2, 3, 4, **5**|, **8**, **9**, **13**, **20**

Sortowanie bąbelkowe

Przykład: n=9, A=[3, 8, 1, 2, 5, 13, 20, 4, 9]

6: 1, 2|, 3, 4, **5**, **8**, **9**, **13**, **20**

6: 1, 2, 3|, 4, **5**, **8**, **9**, **13**, **20**

6: 1, 2, 3, **4**|, **5**, **8**, **9**, **13**, **20**

7: 1, 2|, 3, **4**, **5**, **8**, **9**, **13**, **20**

7: 1, 2, **3**|, **4**, **5**, **8**, **9**, **13**, **20**

8: 1, **2**|, **3**, **4**, **5**, **8**, **9**, **13**, **20**

9: **1**, **2**, **3**, **4**, **5**, **8**, **9**, **13**, **20**

Przykład analizy algorytmu sortowania

- Czas $T(n)$ działania algorytmu dla danych wejściowych rozmiaru n zależy od przyjętej metody obliczeń
- Wyróżniamy 3 przypadki:
  - Czas **optymistyczny**
  - Czas **średni**
  - Czas **pesymistyczny**
- Rozważymy ten ostatni

Przykład analizy algorytmu sortowania

	koszt	*liczba wykonań*
1. **for** j=2 **to** n **do**	c_1	n
2. $naj = A[i]$	c_2	n-1
3. $i = j$-1	c_3	n-1
4. **while** (i>0) **and** ($A[i]$>naj)	c_4	$\sum_{j=2}^{n}(t_j - 1)$
5. **do** $A[i+1] = A[i]$	c_5	$\sum_{j=2}^{n} t_j$
6. $i = i$-1	c_6	$\sum_{j=2}^{n}(t_j - 1)$
7. $A[i+1]$ <-> naj	c_7	n-1

Przykład analizy algorytmu sortowania

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1)$$

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 \qquad \sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7(n-1)$$

$$= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right)n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right)n - (c_2 + c_3 + c_4 + c_7)$$

$$T(n) = an^2 + bn + c = \Theta(n^2)$$

Przykład analizy algorytmu sortowania

- Podany przykład analizy dotyczy pesymistycznego czasu wykonania algorytmu
- W podobny sposób dokonuje się analizy optymistycznego czasu wykonania
- Analiza średniego czasu wykonania algorytmu wymaga znajomości **rozkładu danych wejściowych**