

ASD - Wykład 2

Literatura

T. Cormen, C. Leiserson, R. Rivest, C. Stein,
Wprowadzenie do algorytmów, PWN 2012

S. Dasgupta, C. Papadimitriou, U. Vazirani,
Algorytmy, PWN 2010

S. Skiena, The Algorithm Design Manual,
Springer 2008

Czym się zajmujemy / co nas interesuje?

Efektywne mechaniczne procedury rozwiązyjące
dobrze zdefiniowane problemy obliczeniowe

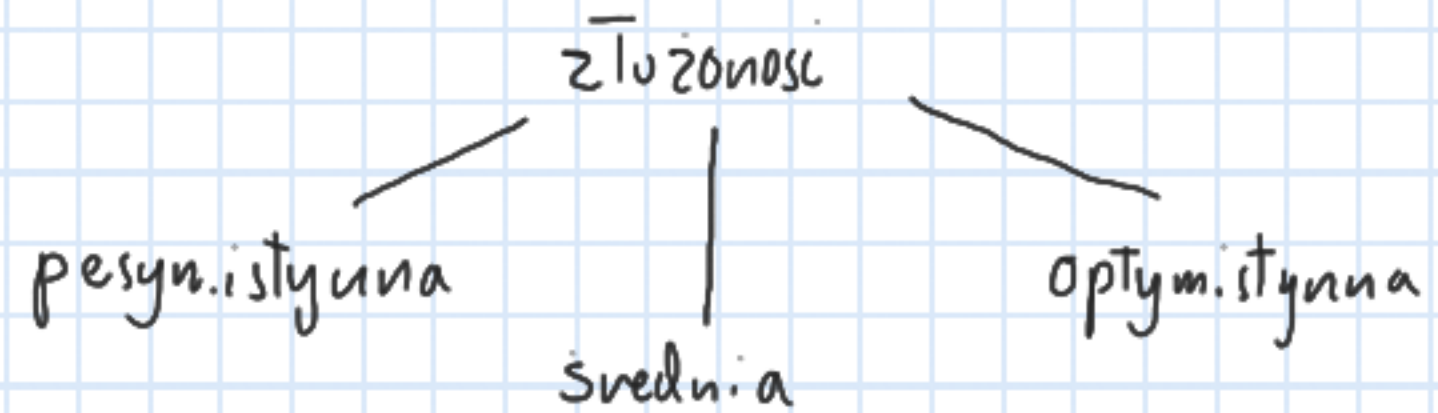
Co nas nie interesuje?

Szczególne techniki

Złożoność obliczeniowa

Złożoność czasowa algorytmu to funkcja, która mówi
ile elementarnych operacji algorytm wykonuje na danych
określonego rozmiaru

Złożoność pamięciowa — j.w., ale mniej lubię użytych komórek ^{pamięci}



Notacja asymptotyczna

f, g - funkcje

$$f: \mathbb{N} \rightarrow \mathbb{N}$$

$$g: \mathbb{N} \rightarrow \mathbb{N}$$

def Mówimy, że f jest $O(g(n))$ jeśli:

$$(\exists c > 0)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0) [f(n) \leq c g(n)]$$

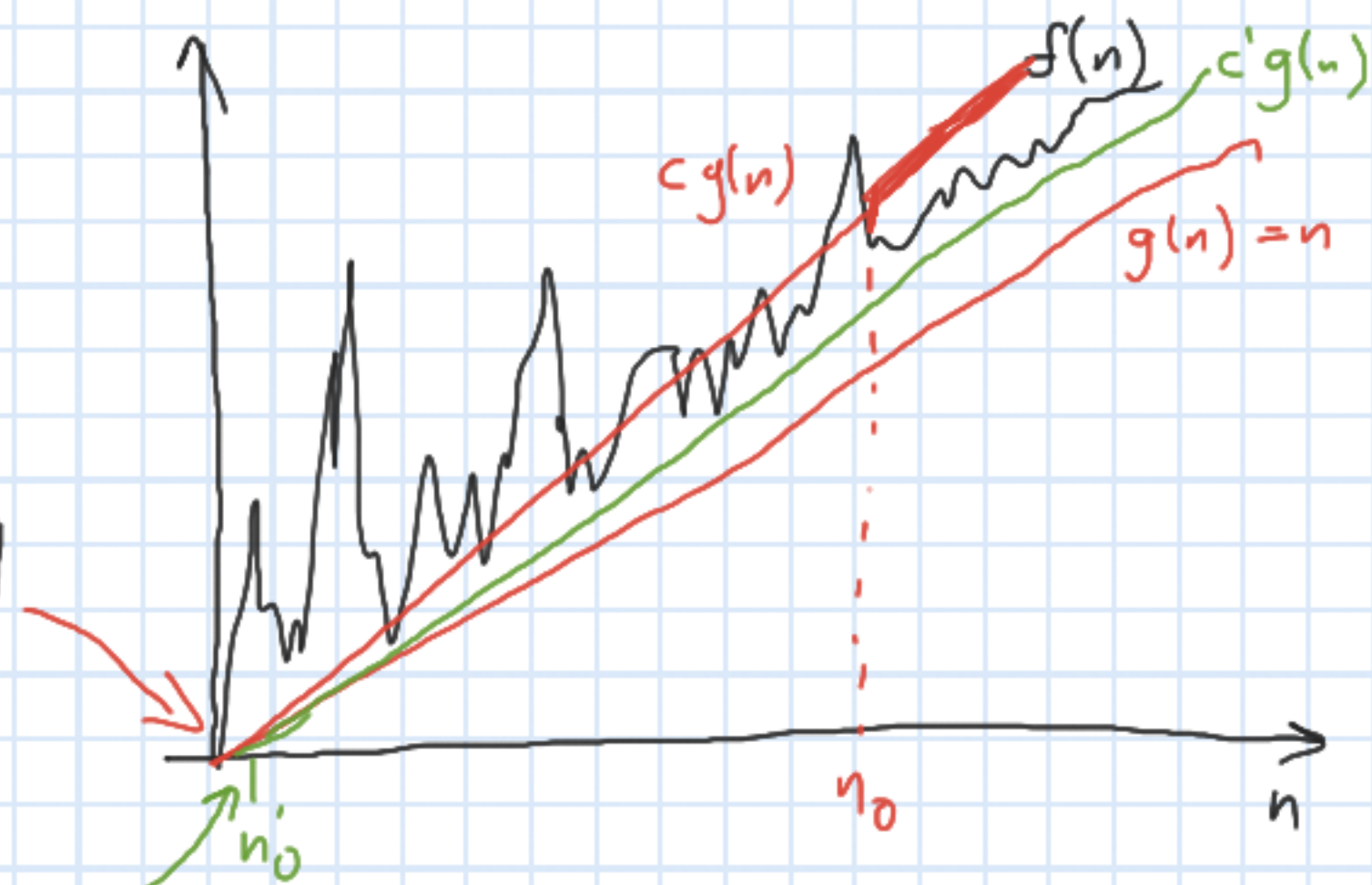
Mówimy, że f jest $\Omega(g(n))$ jeśli:

$$(\exists c > 0)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0) [c g(n) \leq f(n)]$$

Mówimy, że f jest $\Theta(g(n))$ jeśli

- jest $O(g(n))$ i $\Omega(g(n))$

$$- (\exists c_1, c_2 > 0)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0) [c_1 g(n) \leq f(n) \leq c_2 g(n)]$$



Problem sortowania

Dane: ciąg a_1, \dots, a_n danych z operatorem \leq

Wynik: permutacja a'_1, \dots, a'_n ciągu oryginalnego, taka że:

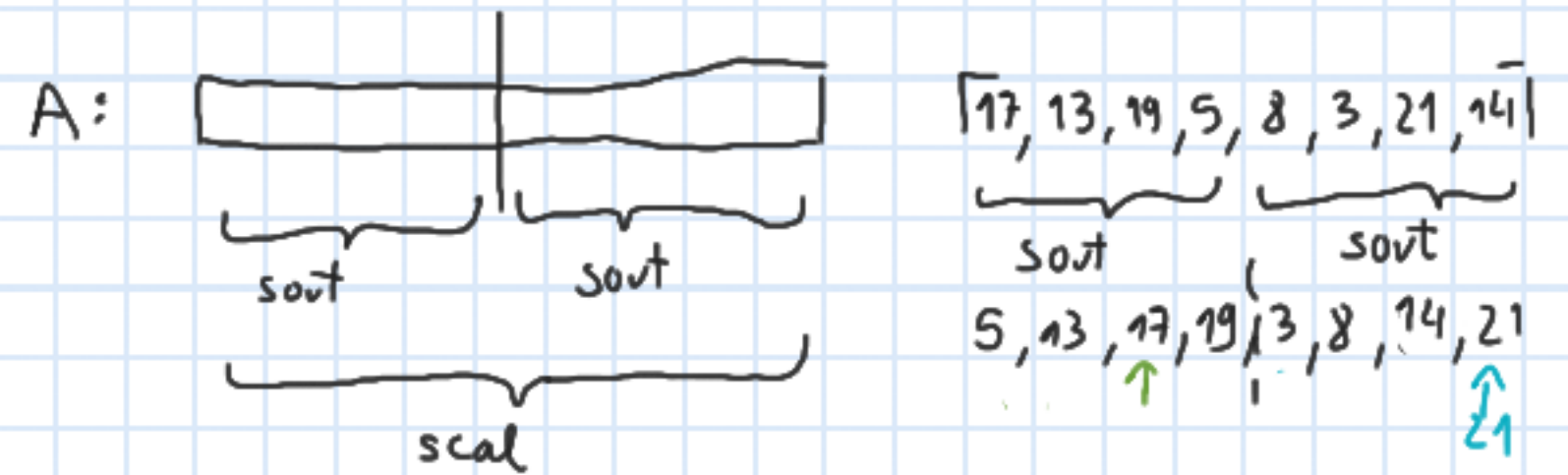
$$a'_1 \leq a'_2 \leq \dots \leq a'_n$$

Uwagi

Reprezentacja danych \rightarrow tablica
 \rightarrow lista 1/2-kierunkowa
 \rightarrow plik

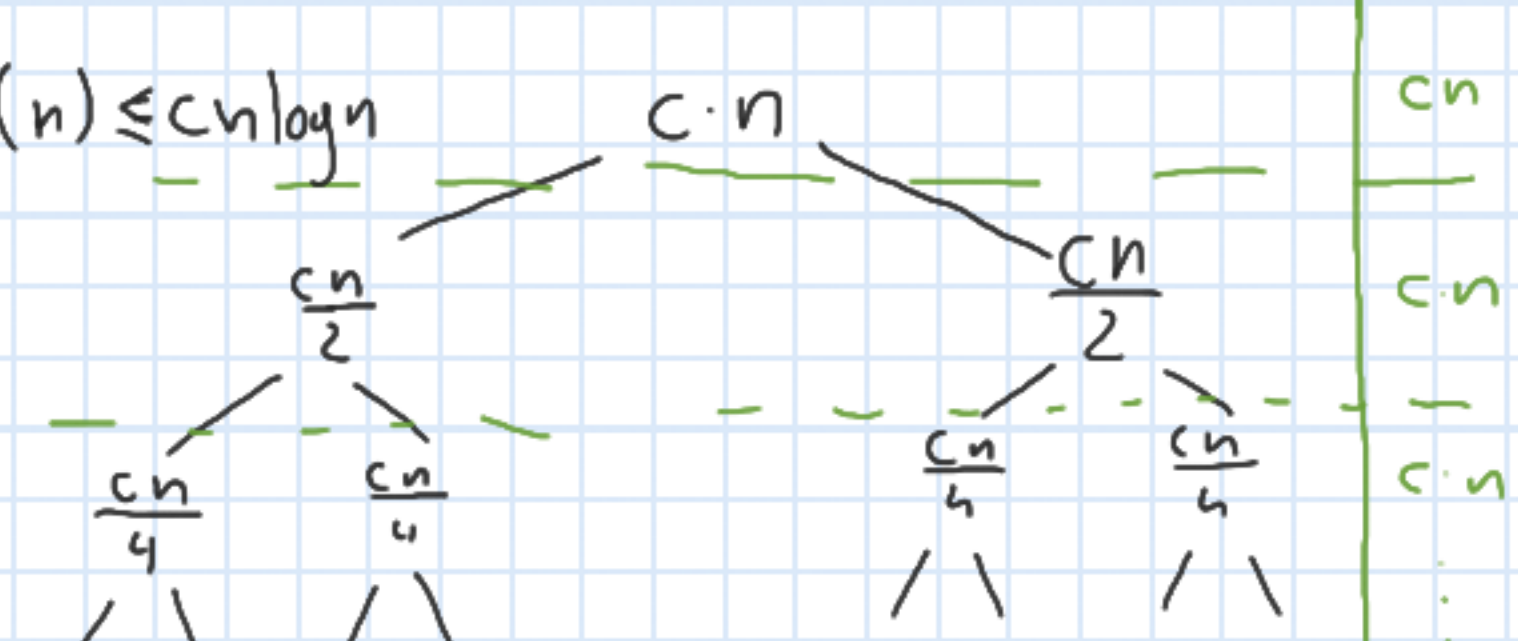
algorytmy sortowania \rightarrow proste $\Theta(n^2)$
 \rightarrow szybkie $\Theta(n \log n)$

① Sortowanie przez scalanie / merge sort



$$T(n) = \begin{cases} c, & n=1 \\ 2 \cdot T(\frac{n}{2}) + cn, & n>1 \end{cases} \Rightarrow T(n) = O(n \log n)$$

$T(n) \leq cn \log n$

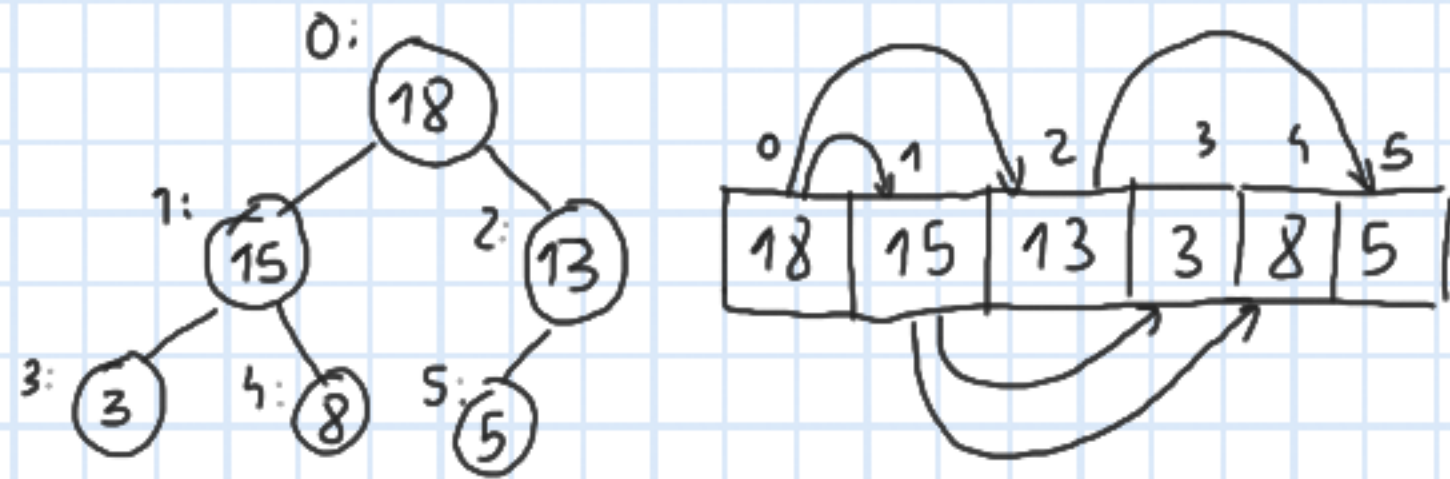


$\frac{n}{2^h}, h = \log n \Rightarrow \frac{n}{2^{\log n}} = \frac{n}{n} = 1$

② Sortowanie kopcowe / heap sort

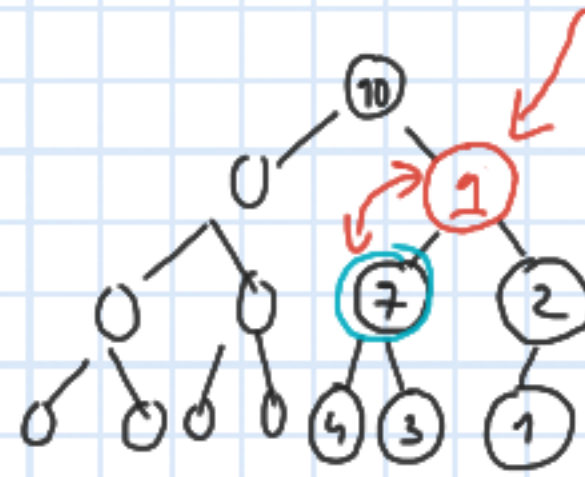
kopiec - drzewo binarne, w którym w każdym węzle spełnionym jest przedmiotowa wartość większa lub równa niż u jego dzieciach

Przykład



```
def left(i): return 2*i + 1
def right(i): return 2*i + 2
def parent(i): return (i-1)//2
```

Przywrócenie własności kopca

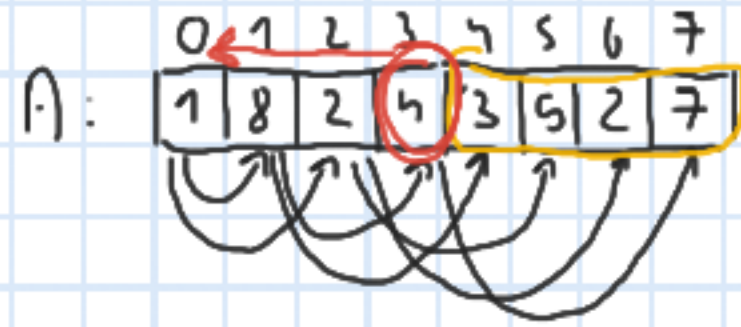


```
def heapify(A, n, i)
    l = left(i)
    r = right(i)
    max_ind = i
    if l < n and A[l] > A[max_ind]:
        max_ind = l
    if r < n and A[r] > A[max_ind]:
        max_ind = r
    if max_ind != i:
        swap(A[i], A[max_ind])
        heapify(A, n, max_ind)
```

$A[i], A[\text{max_ind}] =$
 $A[\text{max_ind}], A[i]$

$\Theta(\log n)$

Jak zbudovat koprec?



```
def build_heap(A):
```

```
    n = len(A)
```

```
    for i in range(parent(n-1), -1, -1):
```

```
        heapify(A, n, i)
```

$O(n \log n)$

$\Theta(n)$

Sortovanie

```
def heap_sort(A):
```

```
    n = len(A)
```

```
    build_heap(A)
```

```
    for i in range(n-1, 0, -1):
```

```
        swap(A[0], A[i])
```

```
        heapify(A, i, 0)
```

