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 rozwiązujące dobrze zdefiniowane problemy obliczeniowe

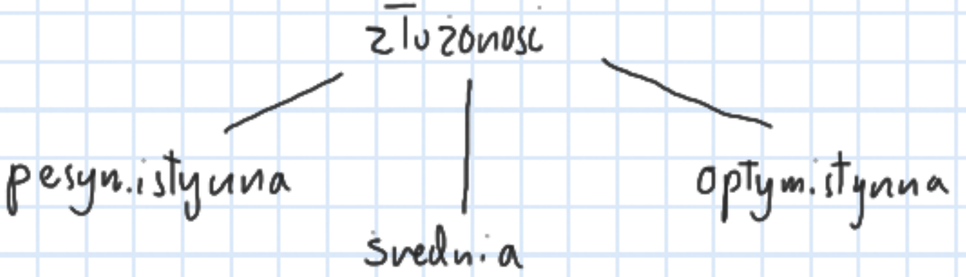
Co nas nie interesuje?

Szczegóły techniczne

Złożoność obliczeniowa

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

Złożoność pamięciowa — j.w., ale mierzy liczbę użytych komórek pamięci



Notacja asymptotyczna

f, g – funkcje

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

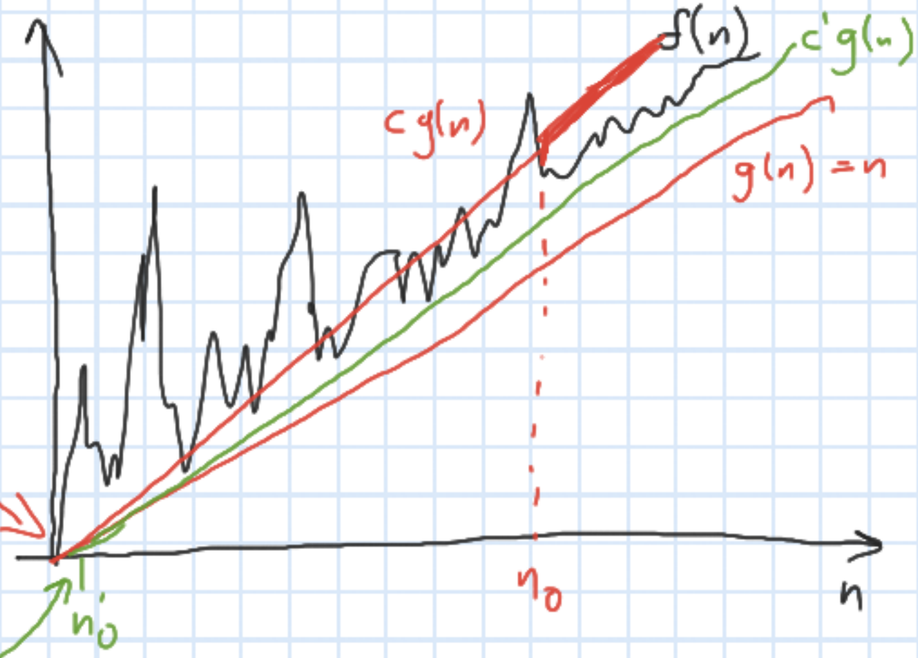
$g : \mathbb{N} \to \mathbb{N}$

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

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

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

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



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

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

Problem sortowania

Dane: ciąg $a_1, \ldots, a_n$ danych z operatorem $\leqslant$

Wynik: permutacja $a_1', \ldots, a_n'$ ciągu wejściowego, taka że:

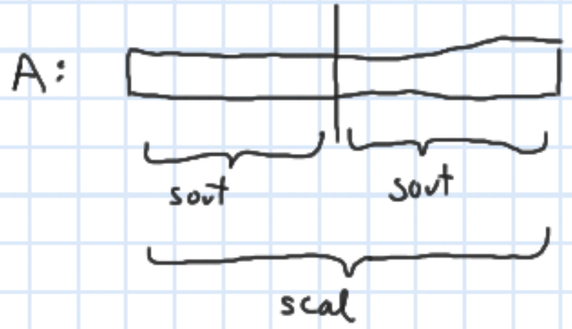
$$a_1' \leqslant a_2' \leqslant \ldots \leqslant a_n'$$

Uwagi

Reprezentacja danych → tablica
- → lista 1/2-kierunkowa
- → plik

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

① Sortowanie przez scalanie / merge sort



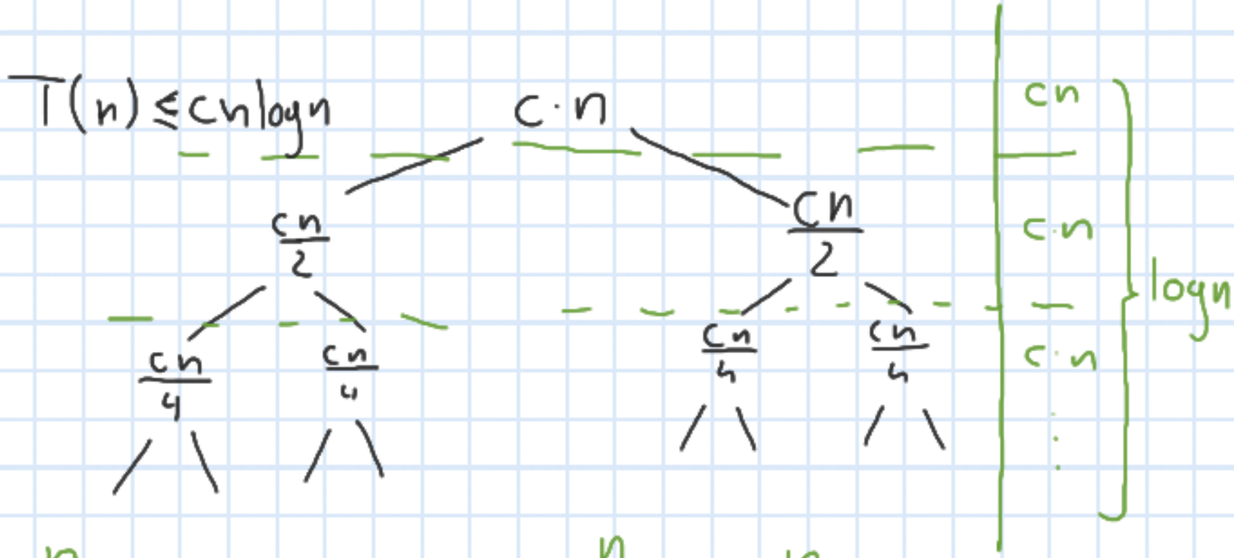
17, 13, 19, 5, 8, 3, 21, 14

sort sort

5, 13, 17, 19 | 3, 8, 14, 21

3, 5, 8, 13, 14, 17, 19, 21

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

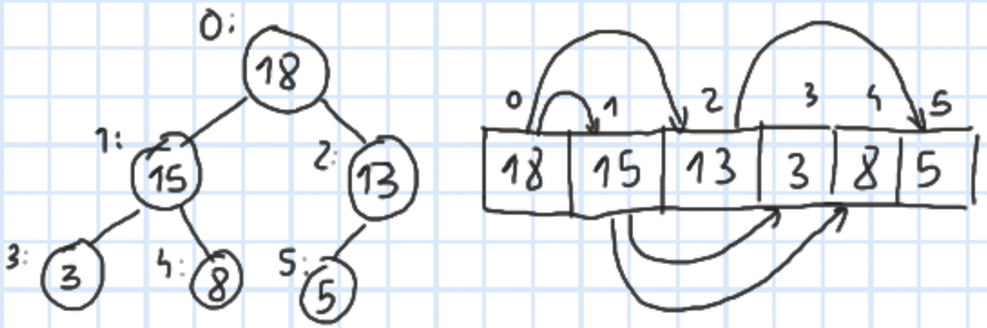


$\frac{n}{2^h}$, $h = \log n \Rightarrow \frac{n}{2^{\log n}} = \frac{n}{n} = 1$

② Sortowanie kopcowe / heap sort

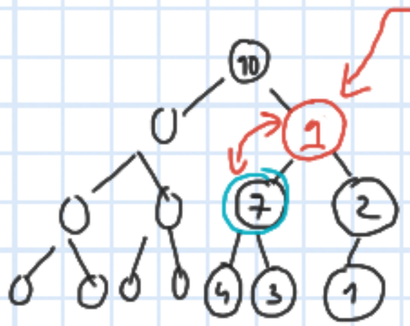
kopiec – drzewo binarne, w którym w każdym węźle wewnętrznym jest przechowywana wartość większa lub równa niż w jego dzieciach

Przykład




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

Przywracanie własności kopca

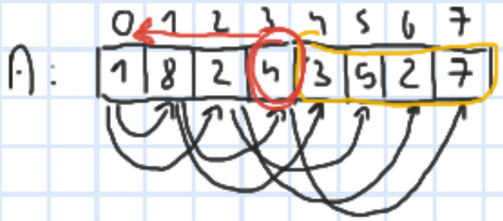



```
def heapify(A, n, i)
    l = left(i)
    r = right(i)
    max_ind = i
    if l < n and A[l] > A[max_ind]:
        max_ind = l
    if r < n and A[r] > A[max_ind]:
        max_ind = r
    if max_ind ≠ i:
        swap(A[i], A[max_ind])
        heapify(A, n, max_ind)
```

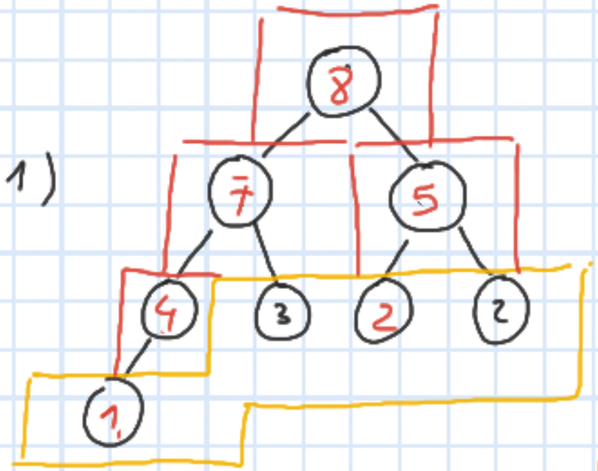
A[i], A[max_ind] = A[max_ind], A[i]

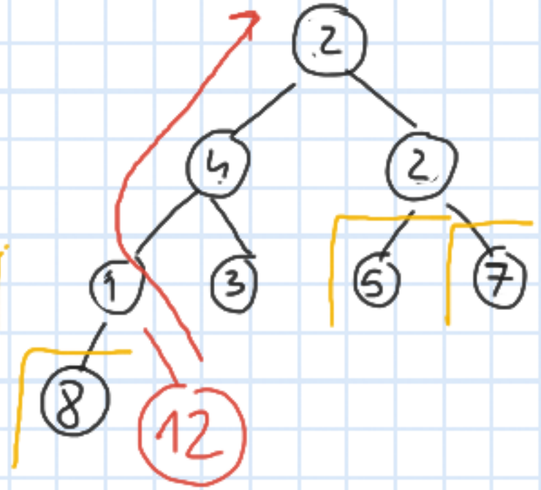
$\Theta(\log n)$

Jak zbudovať kopiec?




```
def build_heap ( A ):
    n = len ( A )
    for i in range (parent(n-1), -1, -1)
        heapify ( A, n, i )
```





$O(n \log n)$

$\Omega(n)$

Sotovanie

```
def heap_sout( A ):
    n = len (A)
    build_heap (A)
    for i in range( n - 1, 0, -1):
        swap ( A[0], A[i])
        heapify ( A, i , 0)
```