

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

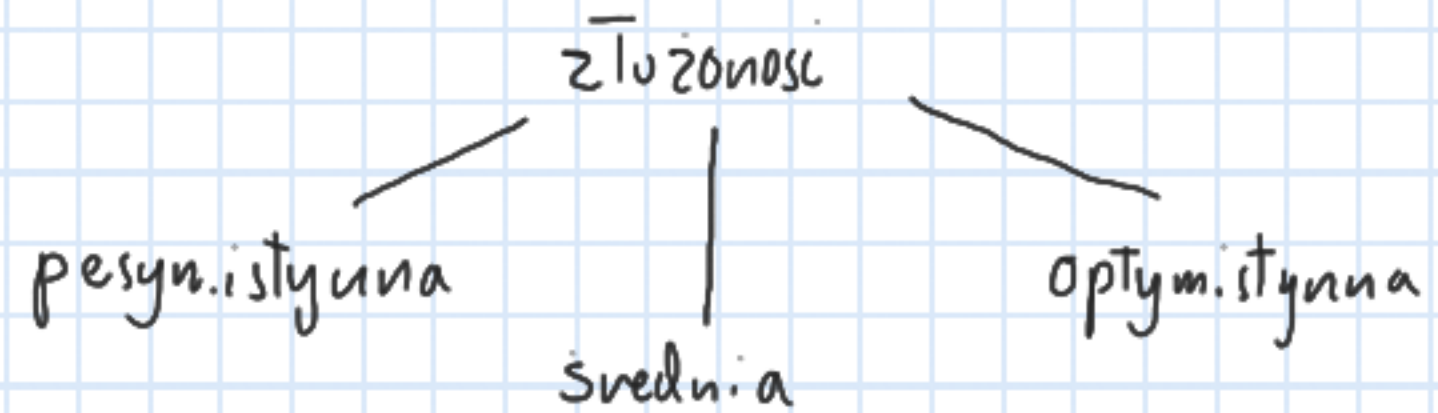
Co nas nie interesuje?

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 mówimy tylko o użytych komórkach ^{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:

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

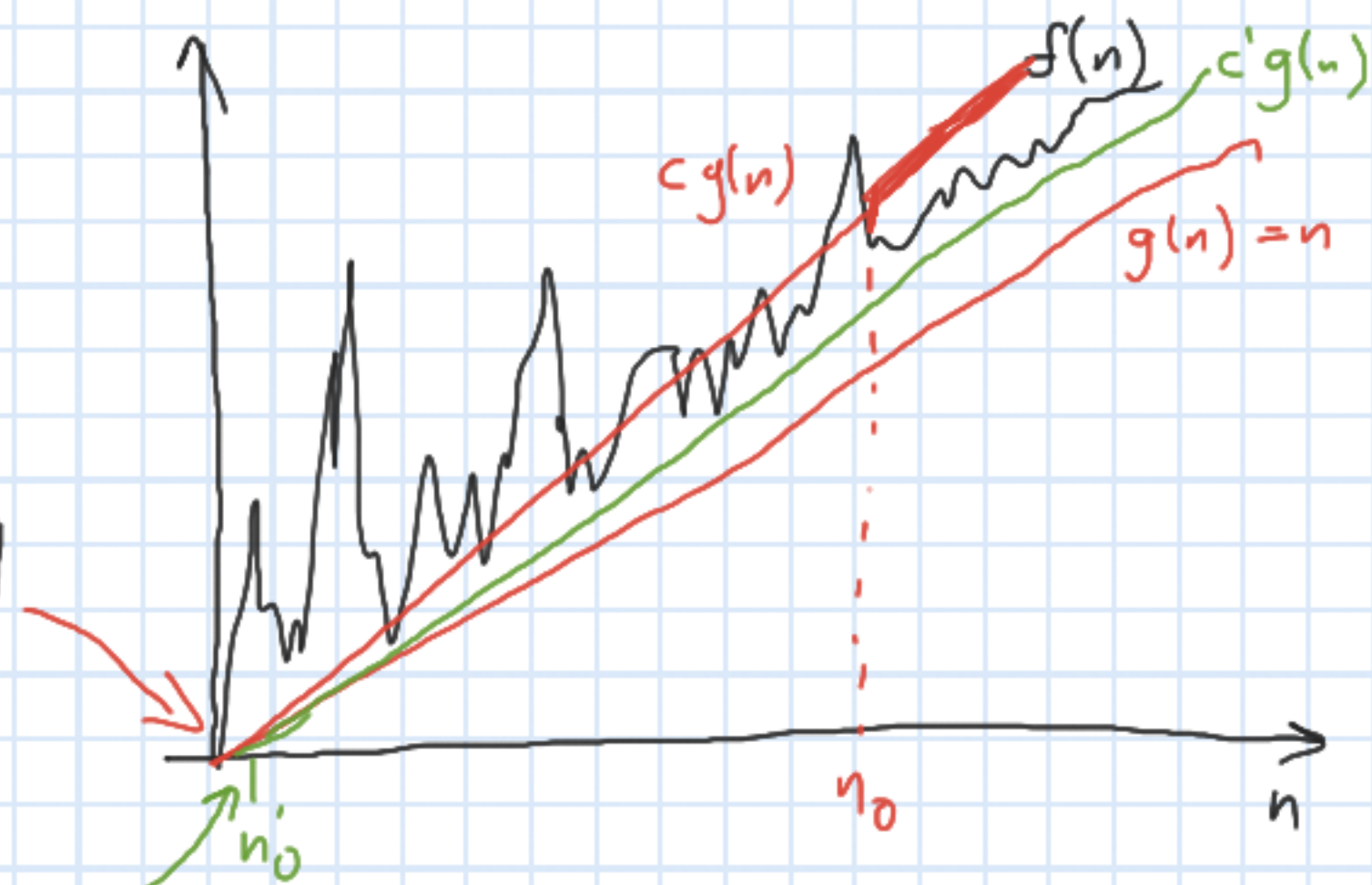
Mówimy, że f jest $\Omega(g(n))$ jeśli:

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

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:

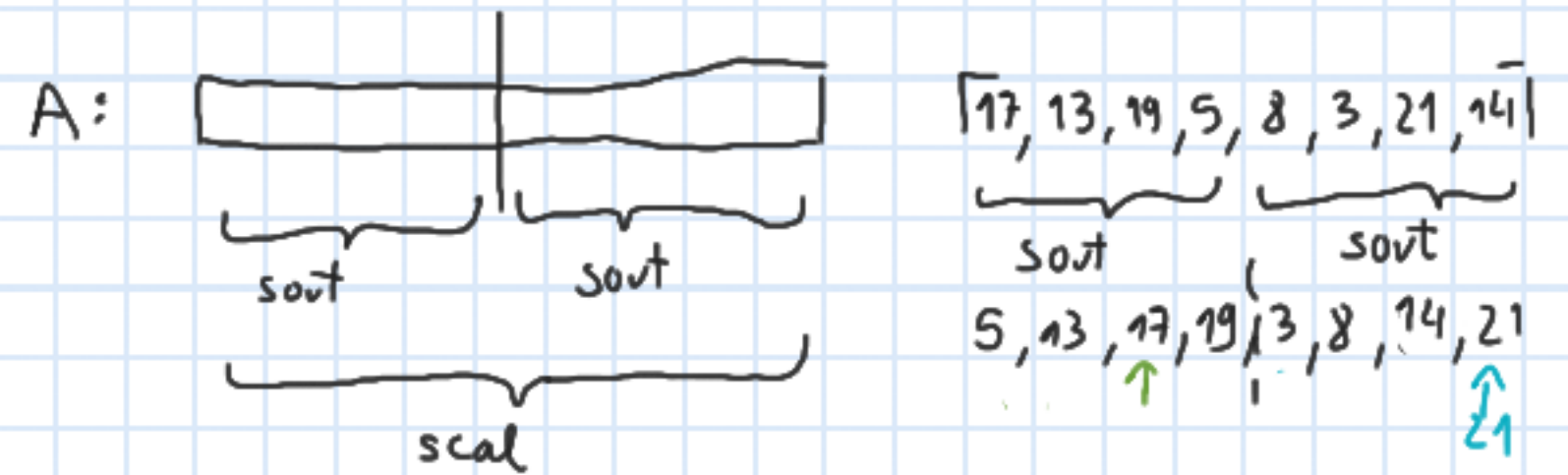
$$a'_1 \leq a'_2 \leq \dots \leq a'_n$$

Uwagi

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

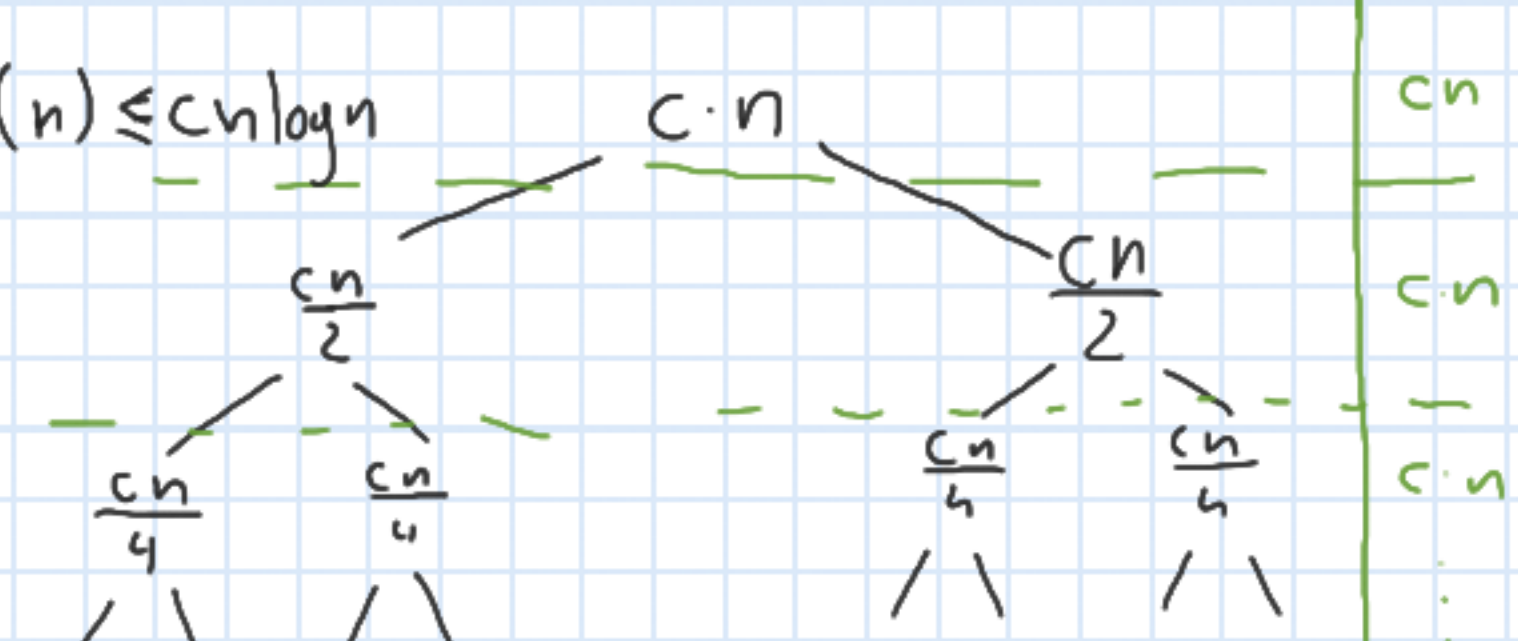
algorytmy sortowania \rightarrow proste $\Theta(n^2)$
 \rightarrow szybkie $\Theta(n \log n)$

① Sortowanie przez scalanie / merge sort



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

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

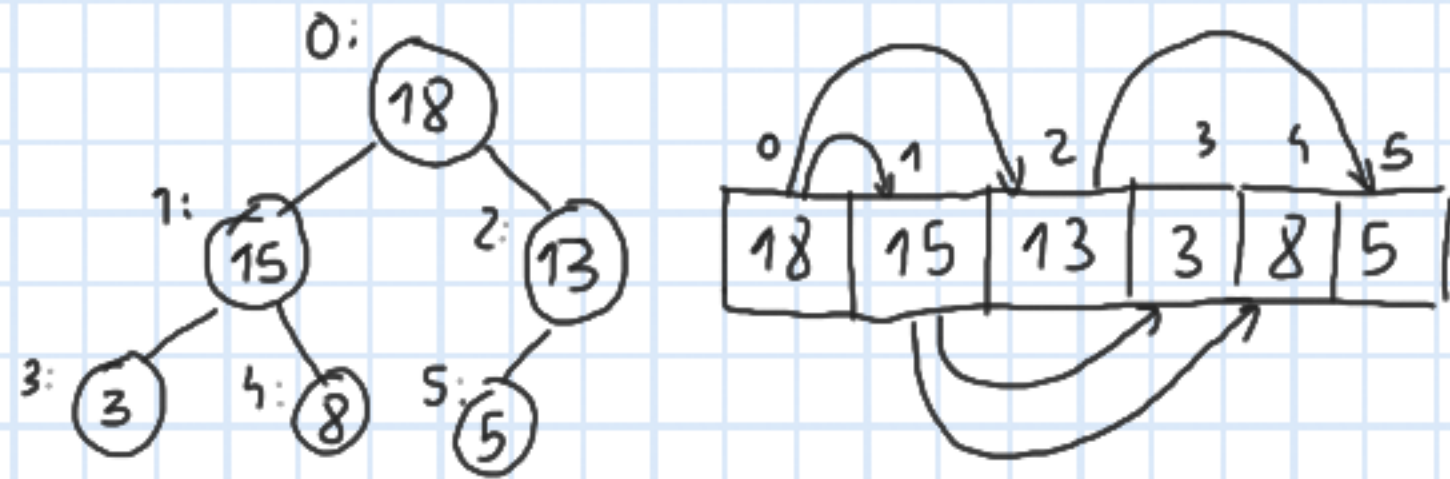


$\frac{n}{2^h}, h = \log n \Rightarrow \frac{n}{2^{\log n}} = \frac{n}{n} = 1$

② Sortowanie kopcowe / heap sort

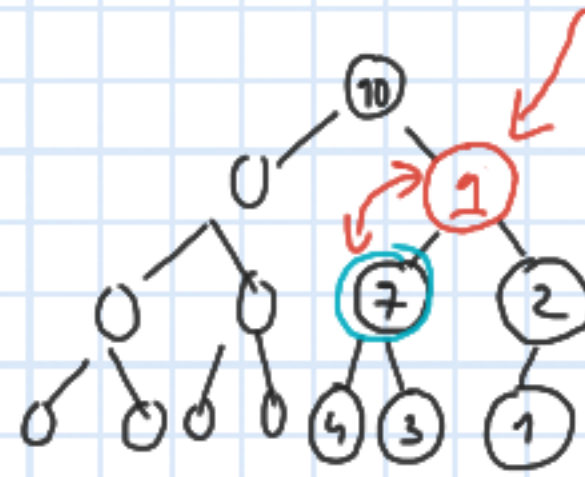
kopiec - drzewo binarne, w którym w każdym węzle spełnionym jest przedyskowane wartości większa lub równa niż u jego dzieciach

Przykład



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

Przywrócenie własności kopca

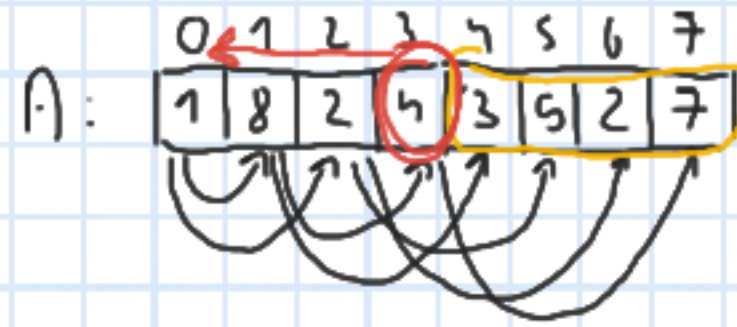


```
def heapify(A, n, i)
    l = left(i)
    r = right(i)
    max_ind = i
    if l < n and A[l] > A[max_ind]:
        max_ind = l
    if r < n and A[r] > A[max_ind]:
        max_ind = r
    if max_ind != i:
        swap(A[i], A[max_ind])
        heapify(A, n, max_ind)
```

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

$\Theta(\log n)$

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):
```

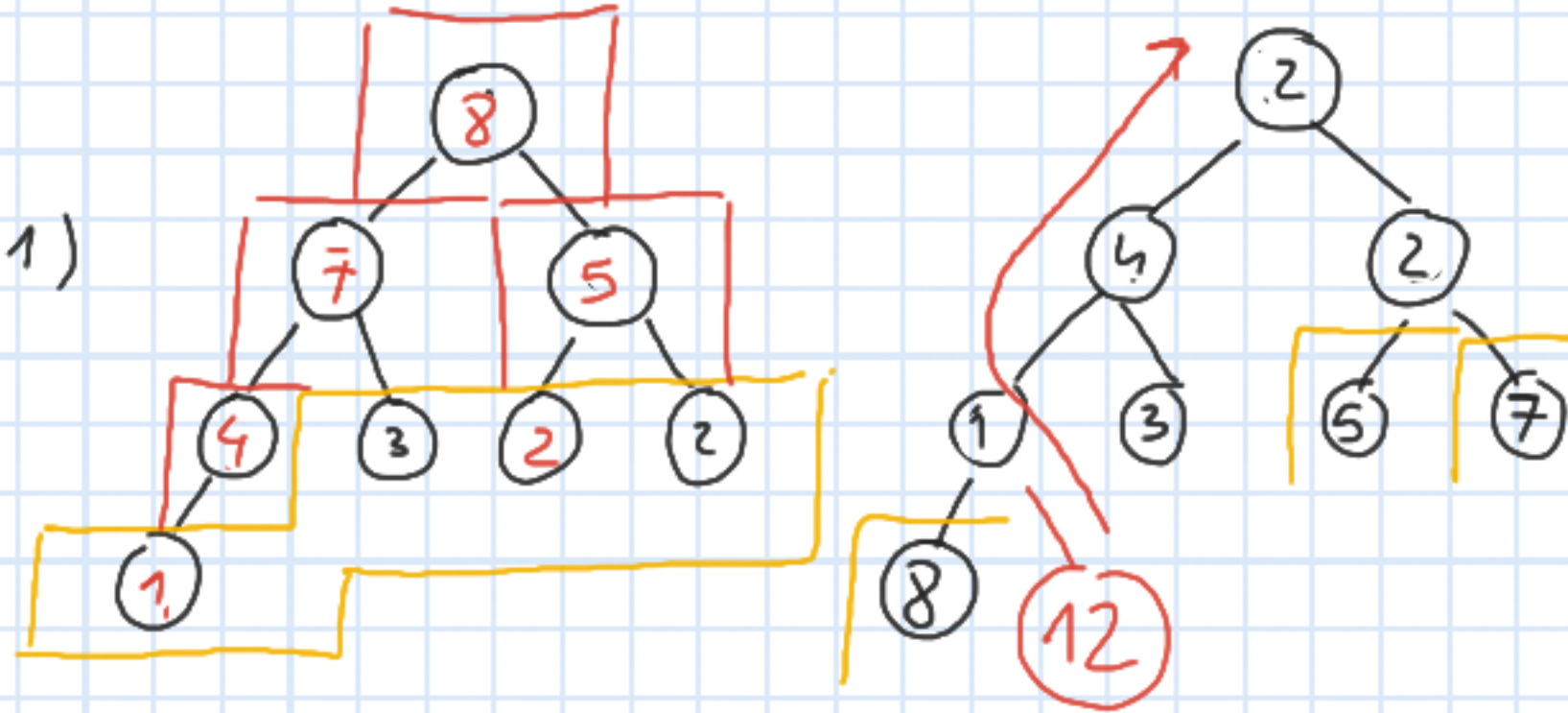
```
    n = len(A)
```

```
    build_heap(A)
```

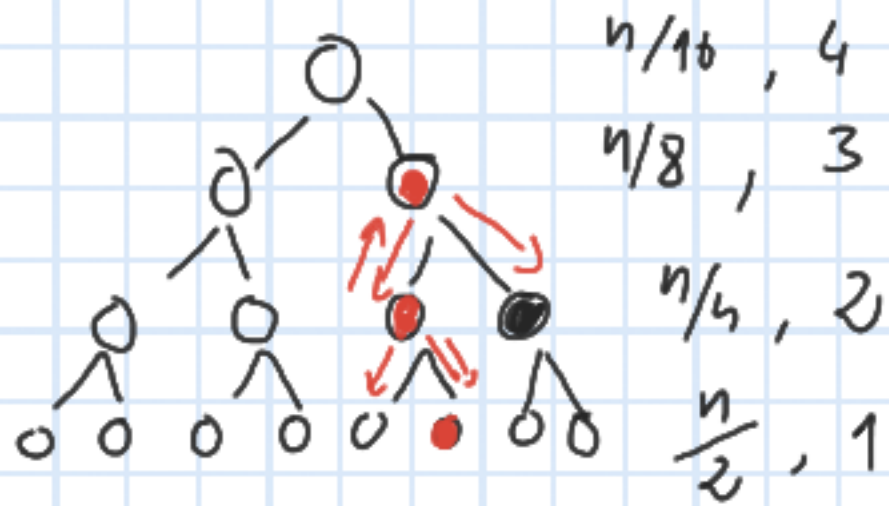
```
    for i in range(n-1, 0, -1):
```

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

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



ASD - Wykład 3



$$\frac{n}{2} + \frac{n}{4} \cdot 2 + \frac{n}{8} \cdot 3 + \dots = \sum_{i=1}^{\lfloor \log n \rfloor} \left(\frac{n}{2^i} \cdot i \right)$$

$$= n \sum_{i=1}^{\lfloor \log n \rfloor} \frac{i}{2^i}$$

$$\leq n \sum_{i=0}^{\infty} \frac{i}{2^i} \leq \boxed{2n}$$

dla $x = \frac{1}{2}$

$$f(x) = 1 + x + x^2 + x^3 + \dots = \frac{1}{1-x}$$

$$f'(x) = 1 + 2x + 3x^2 + 4x^3 + \dots = \frac{1}{(1-x)^2}$$

$$x f'(x) = x + 2x^2 + 3x^3 + 4x^4 + \dots = \frac{x}{(1-x)^2}$$

Kolejka priorytetowa

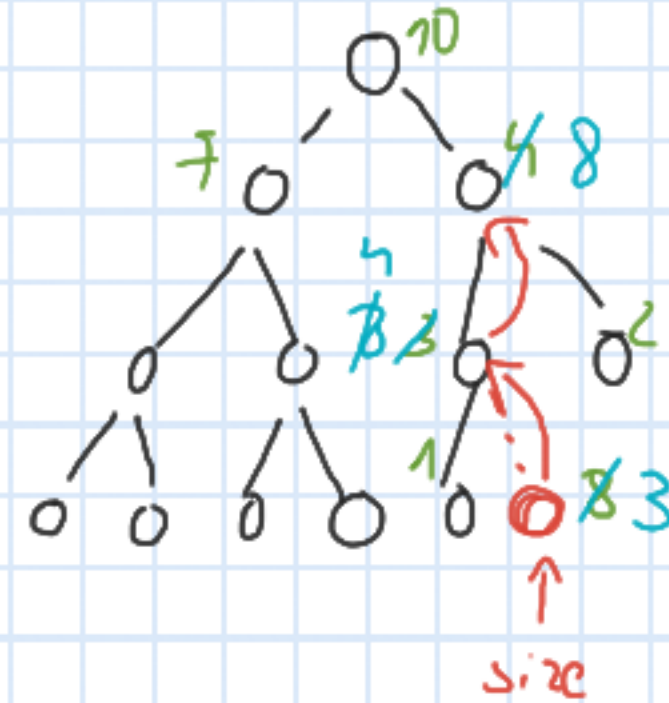
"Coś, do czego można wstawiać elementy w dowolnej kolejności i wyiągać w kolejności zgodnej z priorytetem"

class PQ:

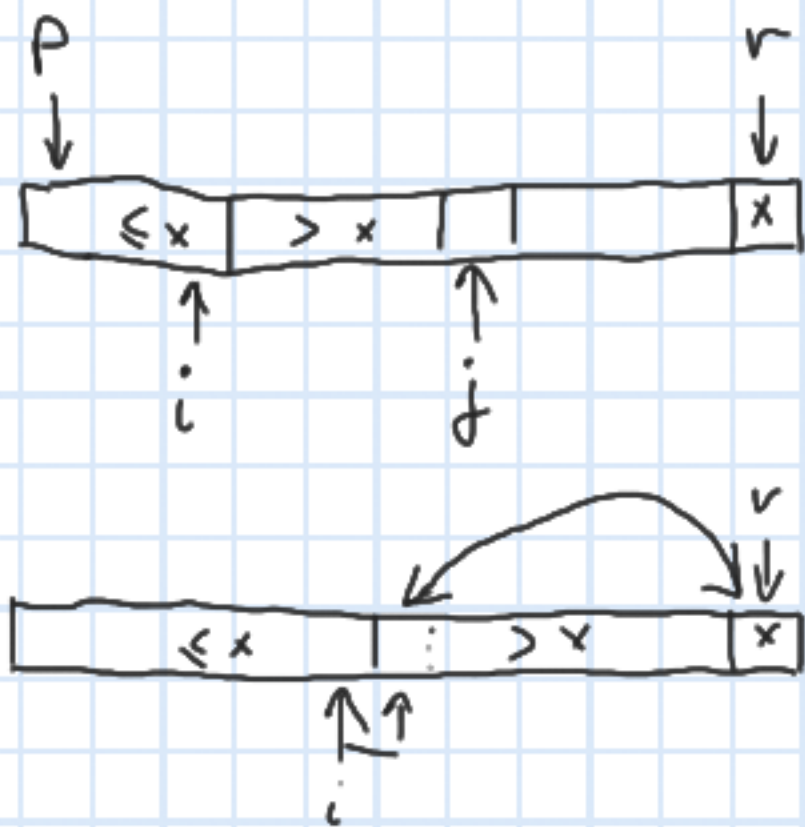
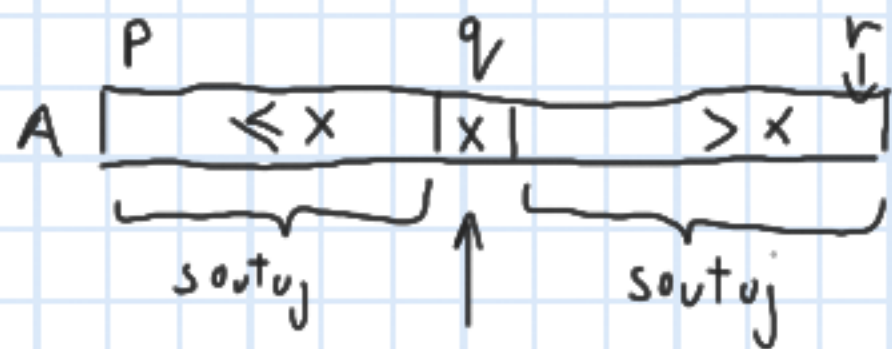
def __init__(self, n):

self.T = [None] * n

self.size = 0



Quick Sort



```
def quick-sort(A, p, r):
```

```
    if p < r:
```

```
        q = partition(A, p, r)
```

```
        quick-sort(A, p, q-1)
```

```
        quick-sort(A, q+1, r)
```

```
def partition(A, p, r):
```

```
    x = A[r] // ← zamień A[r]
```

z losowym elementem

```
    for j in range(p, r):
```

```
        if A[j] ≤ x:
```

```
            i += 1
```

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

```
    swap(A[i+1], A[r])
```

```
    return i + 1
```

Złożoności czasowa algorytmu

Quick Sort

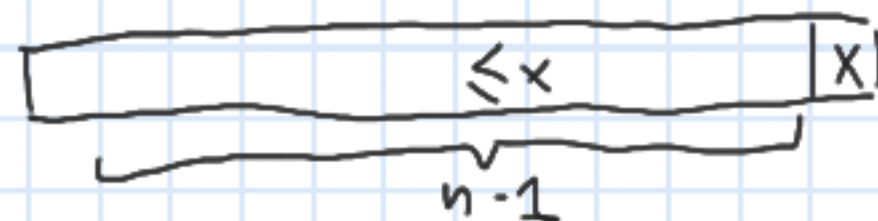
Idealne podziały

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

$$T(n) = \Theta(n \log n)$$

Pedrowe podziały

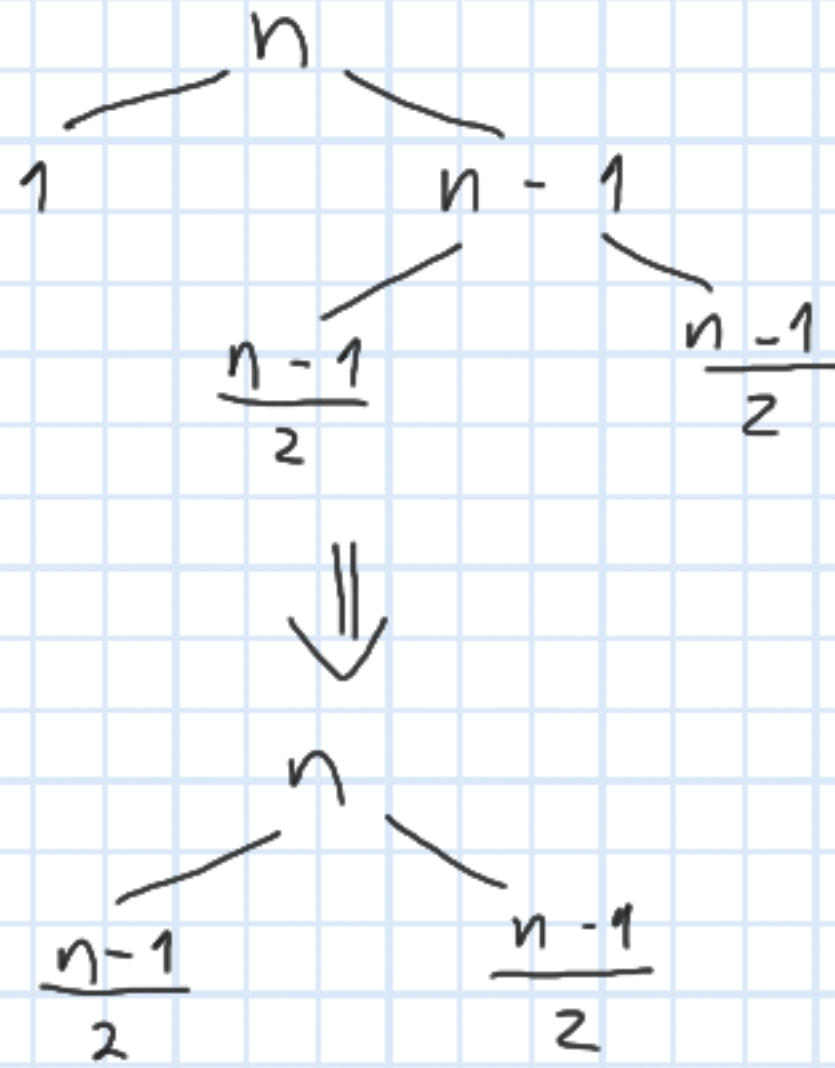
$$T(n) = \begin{cases} c, & n \leq 1 \\ T(n-1) + cn, & n > 1 \end{cases}$$



$$T(n) = T(n-1) + cn = T(n-2) + c(n-1) + cn$$

$$= c(1 + 2 + 3 + \dots + n) = \Theta(n^2)$$

Mieszanka podziału - "co drugi pedzony,
pozostałe idealne"



Statystyki pozycyjne

min/max - ogólny algorytm $\Theta(n)$

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

U ogólności chcemy obliczyć element,
który po posortowaniu byłby pod
indeksem k

$$T(n) = cn + \frac{cn}{2} + \frac{cn}{4} + \dots = \Theta(n)$$

def select(A, p, k, r)

if $p == r$: return A[p]

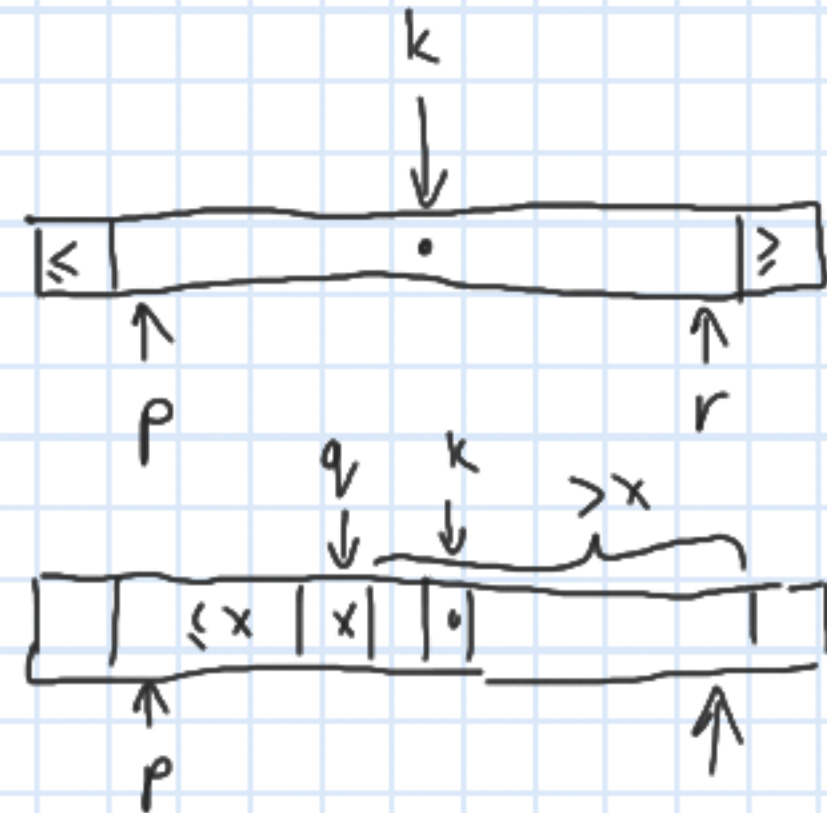
if $p < r$:

q = partition(A, p, r)

if $q == k$: return A[q]

elif $q < k$: return select(A, q+1, k, r)

else : return select(A, p, k, q-1)

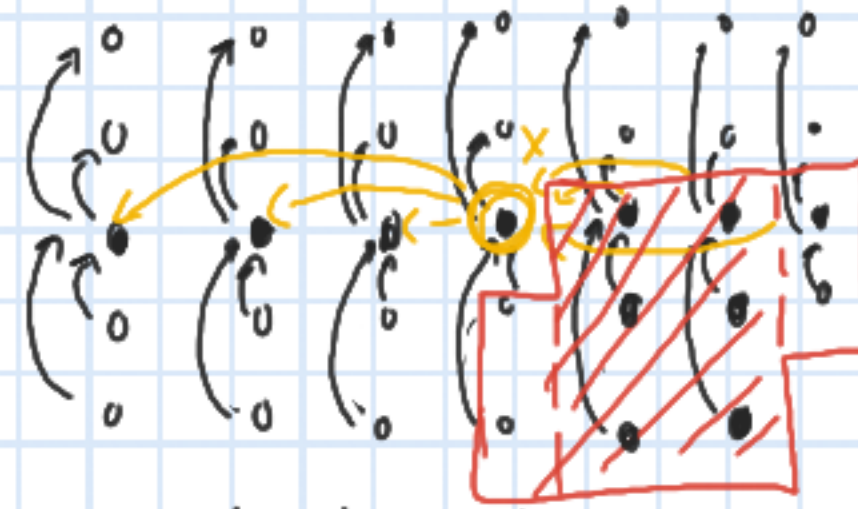


Statystyki pozycyjne w pesymistycznym czasie liniowym — Magiczne Pigtki

Algorytm

- ① Podziel wejściową tablicę na $\lceil \frac{n}{5} \rceil$
grup po 5 elementów
W każdej grupie wyznaczamy medianę
- ② Rekurencyjnie wyznaczamy x jako
medianę median
- ③ Kontynuujemy tak jak w select,
traktując x jak pivot w partition

możemy wybrać tak dwie c ,
że ta wartość jest zawsze
ujemna



Ile jest elementów większych od x ?

$$3 \left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2 \right) \geq \frac{3n}{10} - 6$$

Złożoność czasowa

$$T(n) = \begin{cases} \Theta(1) & , \text{ } n \leq \text{stała} \\ T(\lceil \frac{n}{5} \rceil) + T(\frac{7n}{10} + 6) + \Theta(n) \end{cases}$$

Twierdzimy, że $T(n) \leq cn$, dla pewnego c

Dowód indukcyjny:

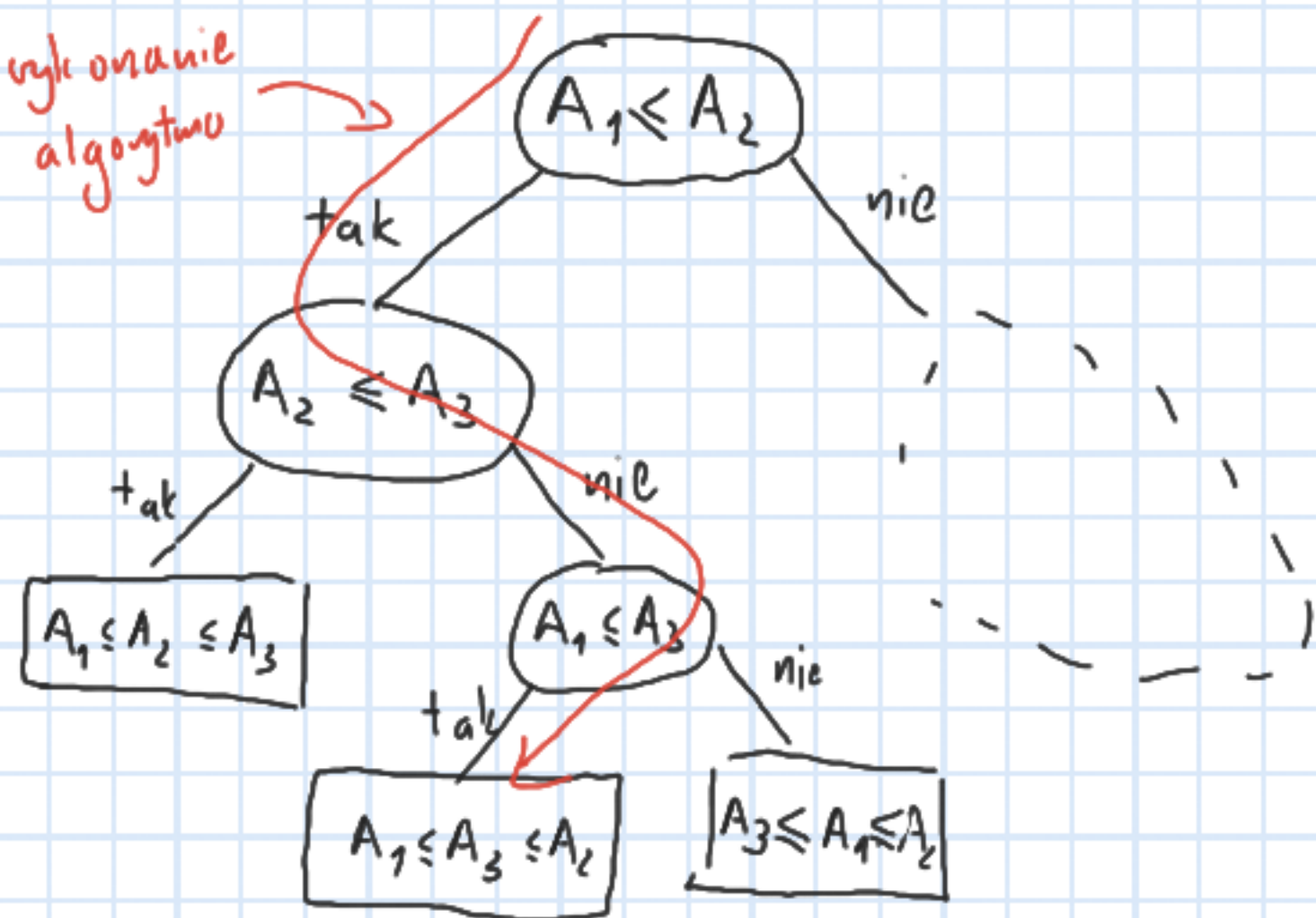
$$\begin{aligned} T(n) &\leq c \lceil \frac{n}{5} \rceil + \frac{7nc}{10} + 6c + an \\ &\leq \frac{2cn}{10} + \frac{7cn}{10} + 7c + an \\ &= cn + \left(-\frac{1}{10}cn + 7c + an \right) \end{aligned}$$

ASD - Wykład 4

Dolne ograniczenie na złożoność czasową sortowania

A: $[A_1 | A_2 | A_3]$ ← tablica n elementów

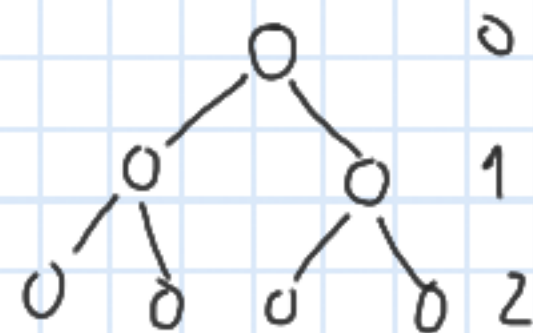
wykonanie algorytmu



↙ wysokość drzewa
 $h \geq \log(n!)$

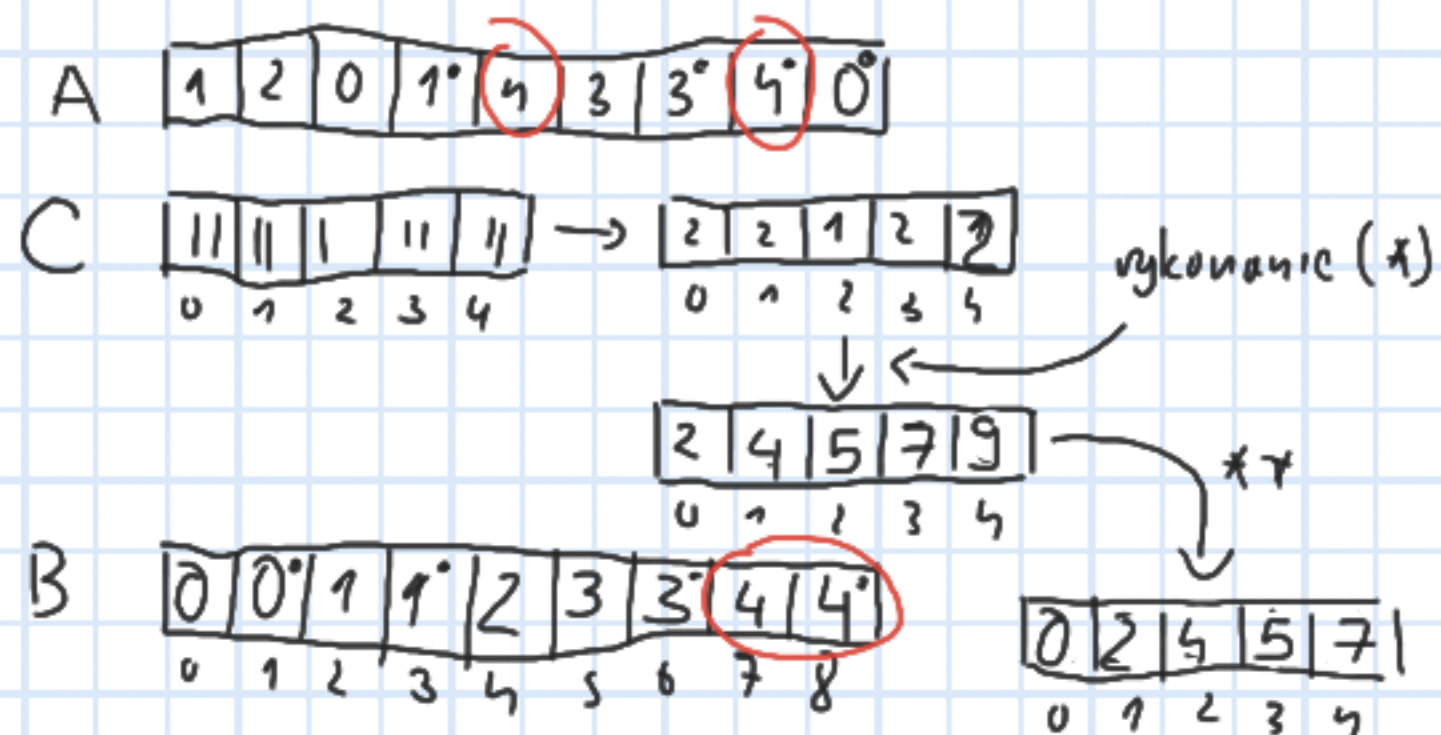
$$\left\{ \begin{aligned} \frac{n}{2}(\log n - 1) &= \log\left(\frac{n}{2}\right)^{\frac{n}{2}} \leq \log n! \leq \log n^n = n \log n \\ &\quad \parallel \\ &\quad \Theta(n \log n) \end{aligned} \right.$$

Mamy drzewo binarne, które ma co najmniej $n!$ liści
Drzewo binarne o wysokości h ma $\leq 2^h$



Sortowanie w czasie liniowym

Counting Sort - sortowanie przez zliczanie
- sortujemy n elementów tablicę z
kluczami, które są liczbami naturalnymi
między 0 a $k-1$



def counting_sort(A, k):

n = len(A)

C = [0] * k

B = [0] * n

for x in A: C[x] += 1

(*) for i in range(1, k): C[i] = C[i] + C[i-1]

(**) for i in range(n-1, -1, -1):

B[C[A[i]] - 1] = A[i]

C[A[i]] -= 1

for i in range(n):

A[i] = B[i]

$\Theta(n + k)$

Radix Sort - sortowanie pozycyjne

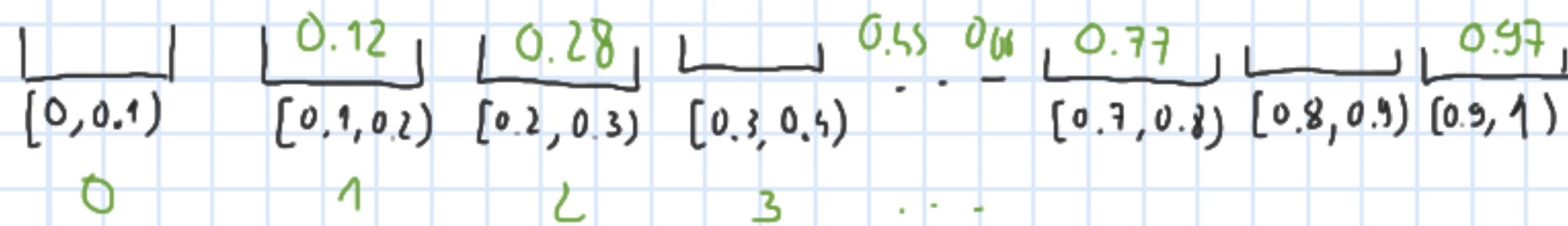
$\begin{array}{c} kva \\ art \\ kot \\ kit \\ ati \\ kil \end{array} \Rightarrow \begin{array}{c} kva \\ ati \\ kil \\ art \\ kot \\ kit \end{array} \Rightarrow \begin{array}{c} kil \\ kit \\ kot \\ kva \\ art \\ ati \end{array} \Rightarrow \begin{array}{c} art \\ ati \\ kil \\ kit \\ kot \\ kva \end{array}$

Sortowanie kubełkowe - algorytm stosowany wtedy, gdy wiemy, że dane pochodzą z rozkładu jednostajnego na pewnym przedziale

Rozważmy n elementów tablicy A , których klucze pochodzą z rozkładu jednostajnego na $[0,1)$

$$n = 10$$

$0.12, 0.77, 0.45, 0.66, 0.28, 0.97, \dots$



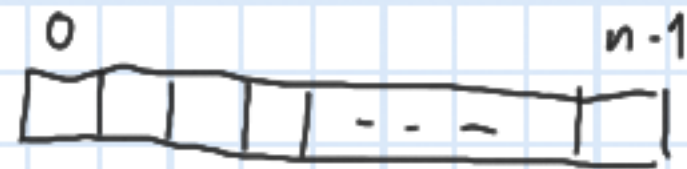
Tworzymy n kubełków

Abstrakcyjne struktury danych

coś co oferuje pewien "kontrakt" opisujący
zbiór operacji w jaki sposób
będzie działał

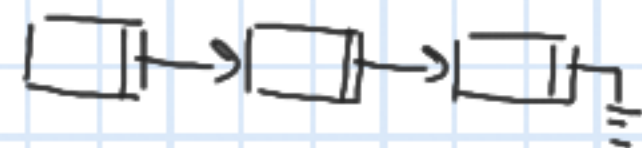
fizyczna realizacja

Tablica



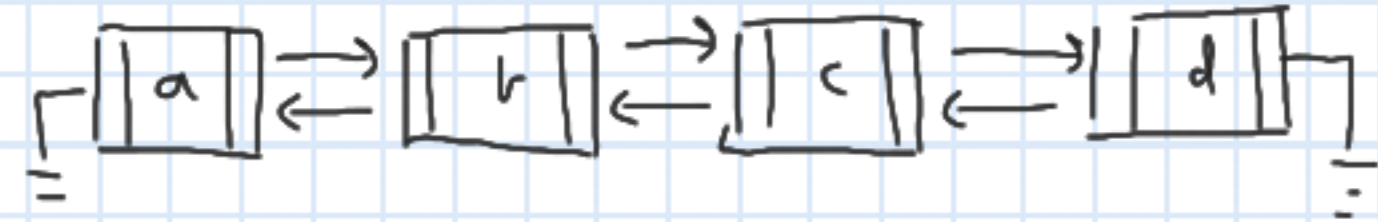
"coś, do czego można się odwoływać
po numerach komórek"

Lista jedno/dwukierunkowa



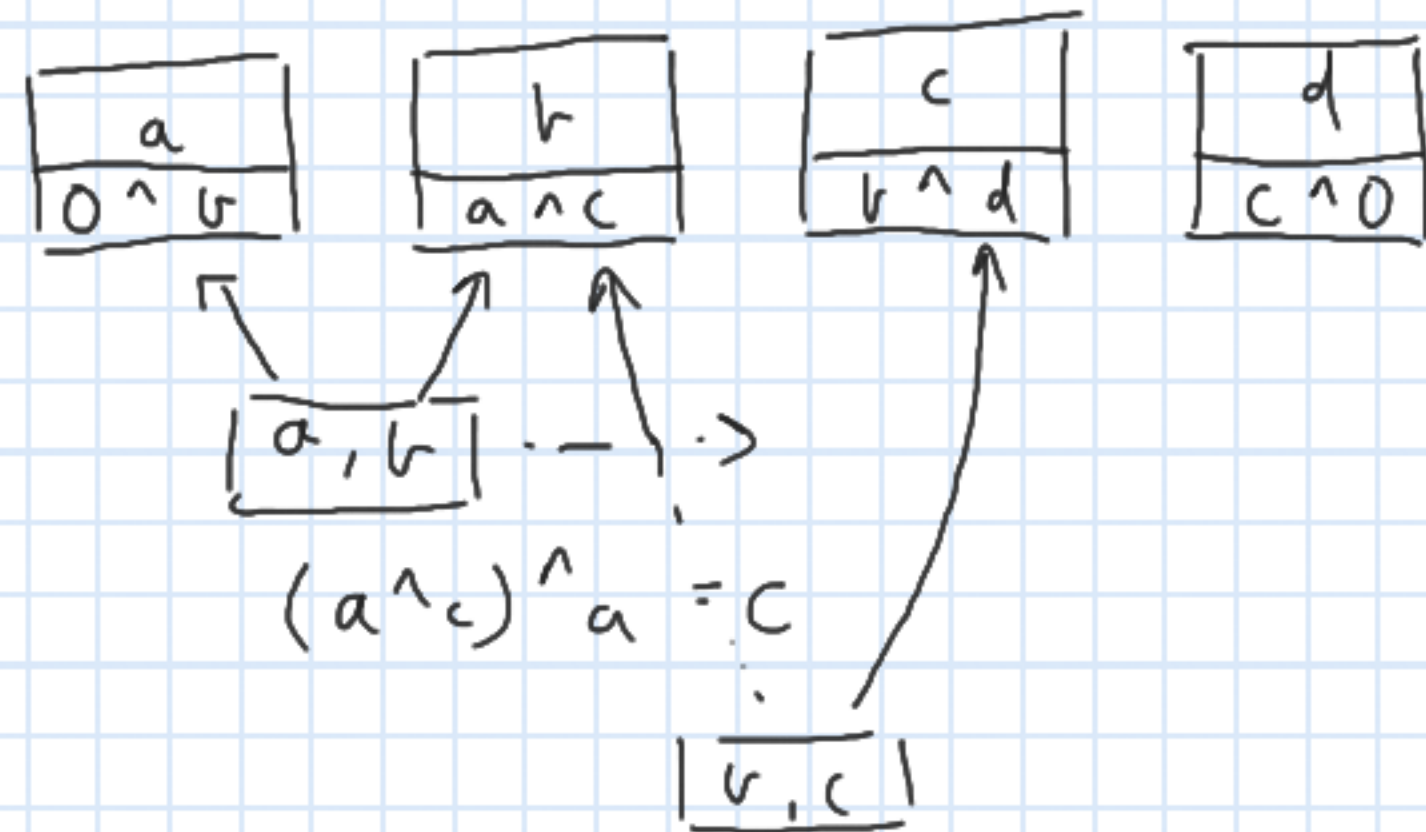
"coś, co pozwala przechodzić się od punktu do
konca i wpinać/usuwać elementy"

Lista dwukierunkowa z jednym wskaźnikiem



$$\begin{array}{r} \text{xor} \quad 10110 \quad A \\ \quad \quad 01100 \quad B \\ \hline \quad \quad 11010 \end{array}$$
$$\begin{array}{r} 11010 \quad (A \text{ xor } B) \\ \quad \quad 01100 \quad B \\ \hline \quad \quad 10110 \quad A \end{array}$$

$$\text{xor} \equiv \wedge$$



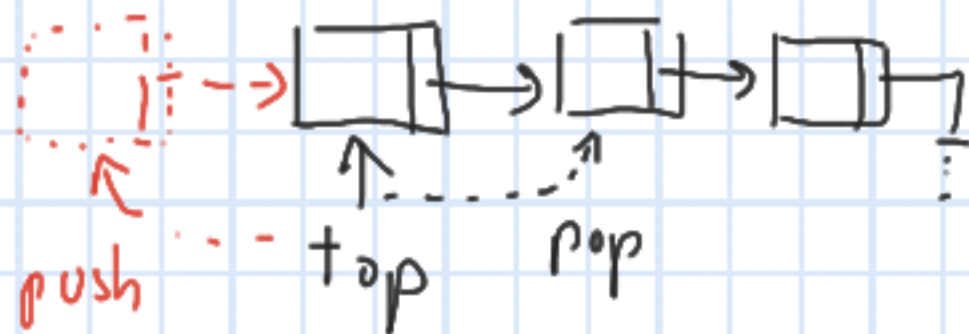
Stos

"Coś, co pozwala odkładać elementy na szczyt i z niego zdejmować"

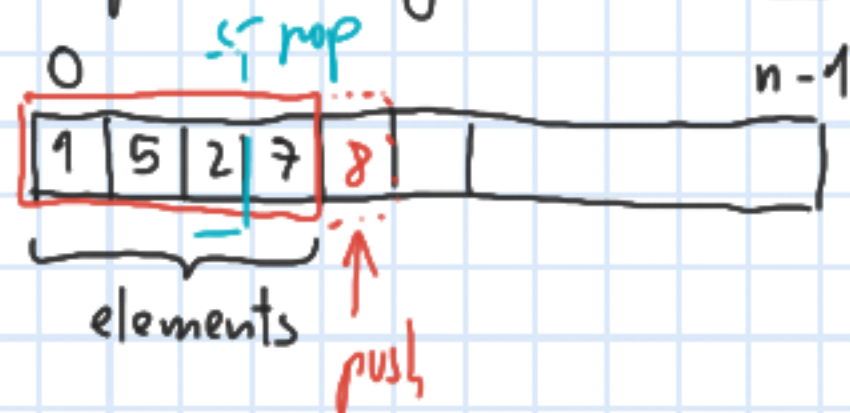
Operacje

- push(x)
- pop()
- is_empty()

Implementacja listowa



Implementacja tablicowa



Dwa stosy?

