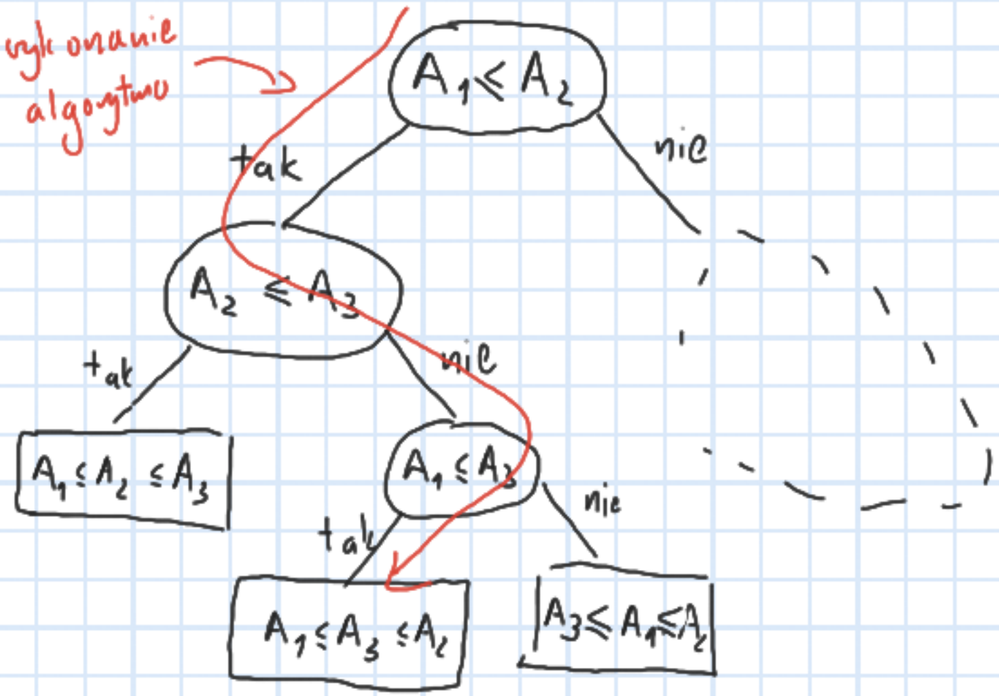
ASD - Wykład 4

Dolne ograniczenie na złożoność czasową sortowania

A: $\boxed{A_1 | A_2 | A_3}$ ← tablica n elementowa



wysokość drzewa

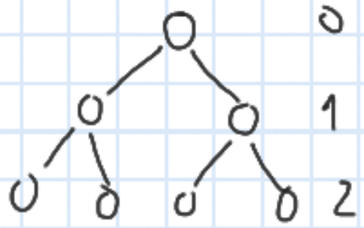
$h \geq \log(n!)$

$\frac{n}{2}(\log n - 1) = \log\left(\frac{n}{2}\right)^{\frac{n}{2}} \leq \log n! \leq \log n^n = n \log n$

$\Theta(n \log n)$

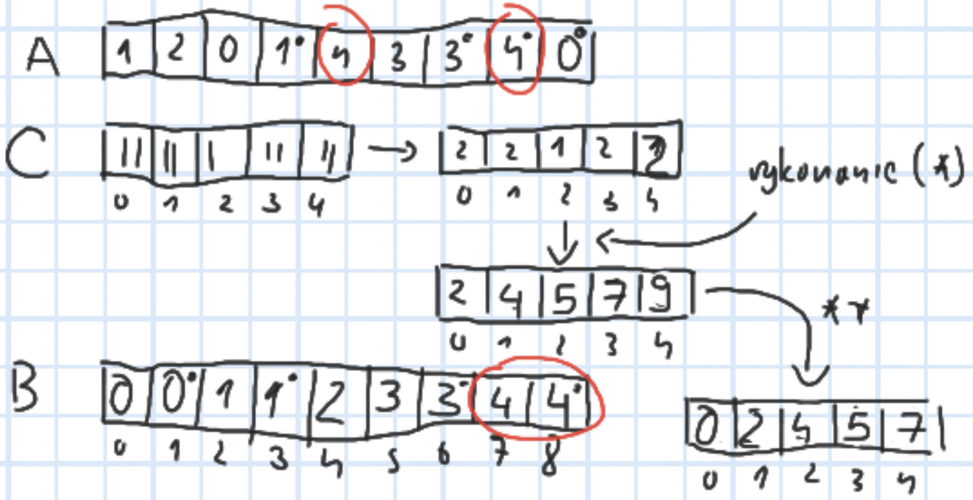
Mamy drzewo binarne, które ma co najmniej $n!$ liści

Drzewo binarne o wysokości h ma $\leq 2^h$



Sortowanie w czasie liniowym

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



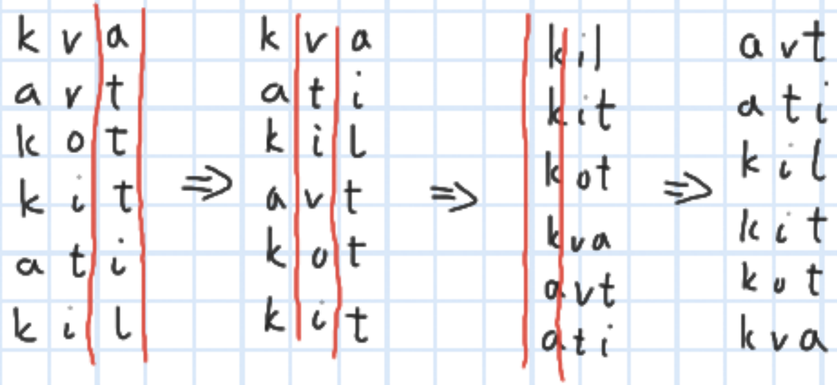

```
def counting_sort(A, k):
    n = len(A)
    C = [0] * k
    B = [0] * n
    for x in A: C[x] += 1
```

(*) `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]
```

$O(n + k)$

Radix Sort – sortowanie pozycyjne

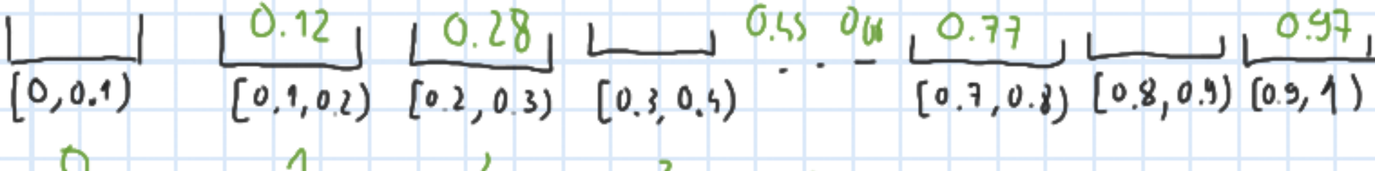


Sortowanie kubełkowe – algorytm stosowany wtedy, gdy wiemy, że dane pochodzą z rozkładu jednostajnego na pewnym przedziale

Rozważmy n elementową tablicę A, której klucze pochodzą z rozkładu jednostajnego na $[0,1)$

$n = 10$

0.12 , 0.77 , 0.45 , 0.66 , 0.28 , 0.97 , ...

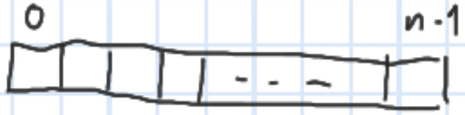


Tworzymy n kubełków

Abstrakcyjne struktury danych

- coś co oferuje pewien "kontrakt" opisujący
  - zbiór operacji
  - w jaki sposób będzie działać
- fizyczna realizacja

Tablica

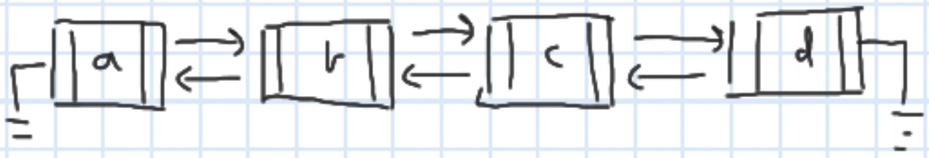


"coś, do niego można się odwoływać po numerach komórek"

Lista jedno/dwukierunkowa

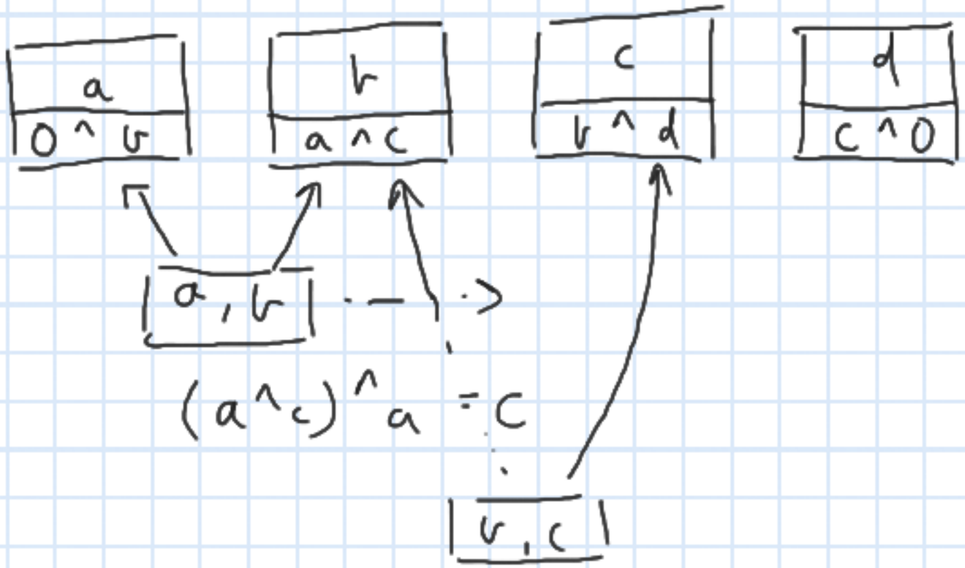
"Coś, co pozwala mierzyć się od początku do końca i wpinać/usuwać elementy"

Lista dwukierunkowa z jednym wskaźnikiem




```
  10110  A:     11010  (A XOR B)
XOR 01100  B: XOR 01100  B
    11010      A: 10110
```

XOR ≡ ^



Stos

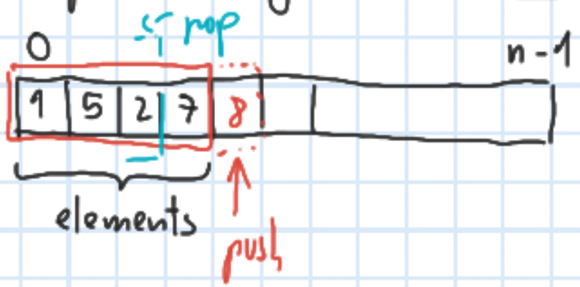
"Coś, co pozwala odkładać elementy na szczyt i z niego zdejmować"

Operacje

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

Implementacja listowa

Implementacja tablicowa



Dwa stosy?