

Algorytmy i Struktury Danych

Wykład 1

Wykład - nagranie + PDF z notatkami

UPeL

Cwiczenia - WebEx

kolokwia $\rightarrow 3 \times 3$ zadania 0/1/2

100% $\rightarrow 18$ pkt

plusy = 0.5 pkt

30 min $\rightarrow \leq 1$ plus / ≤ 2 plusy / ≤ 3 plusy

zadania obruzgające +/0/-

zadania offline - 0/0.5/1 pkt najwyżej 2 razy

Egzamin : 1 - czerwiec

2 } 3 w niesieniu

29 III
17 V
14 VI

Literatura

① T. Cormen, C. Leiserson, R. Rivest, C. Stein

Wprowadzenie do algorytmów, PWN 2012

② S. Dasgupta, C. Papadimitriou, U. Vazirani,
Algorytmy, PWN 2010

③ S. Skiena, The Algorithm Design Manual,
Springer 2008

kolokwia założeniowe

- w terminach egzaminów

O czym jest ten przedmiot?

- jakimi metodami można efektywnie

rozwiązywać problemy obliczeniowe?

- efektywna organizacja danych?

O czym nie jest?

- nlc jest to kurs programowania
-

Złożoność obliczeniowa

- złożoność czasowa algorytmu to funkcja opisująca liczbę elementarnych operacji, jakie algorytm wykonyje na danych określonego rozmiaru
- złożoność pamięciowa to liczba używanych kolejnych pamięci

Notacja asymptotyczna

$f(n)$ jest $\mathcal{O}(g(n))$

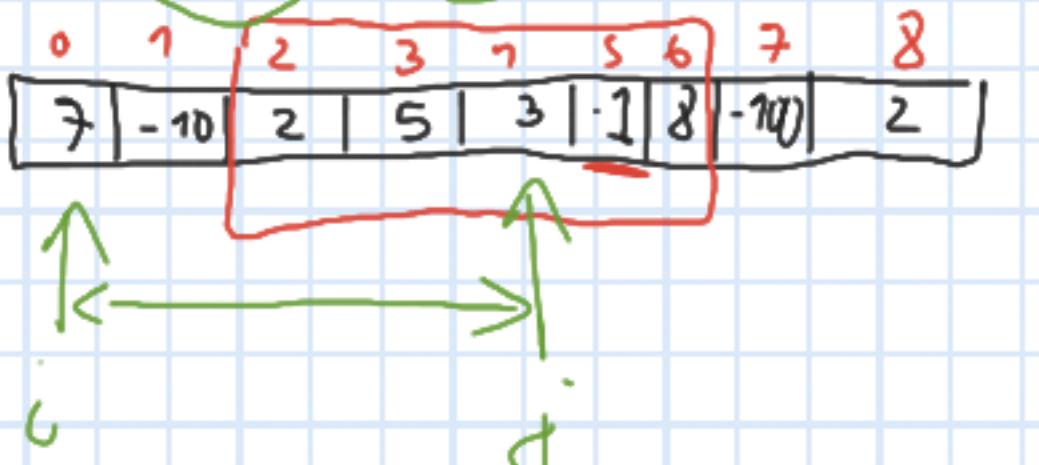
$f(n)$ jest $\mathcal{\Omega}(g(n))$

$f(n)$ jest $\Theta(g(n))$

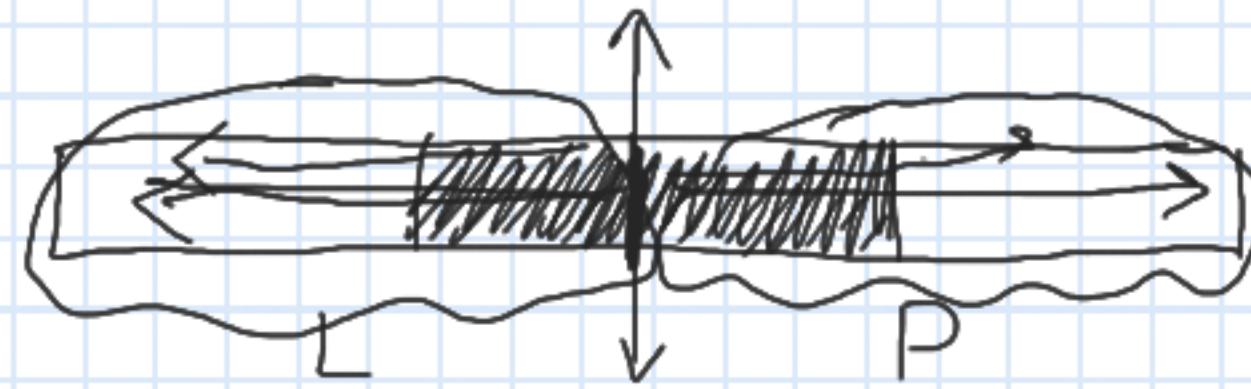
Problem: Suma Spojnego Podcięgu (SSP)

Dane : $A[0, \dots, n-1]$ - tablica liczb całkowitych

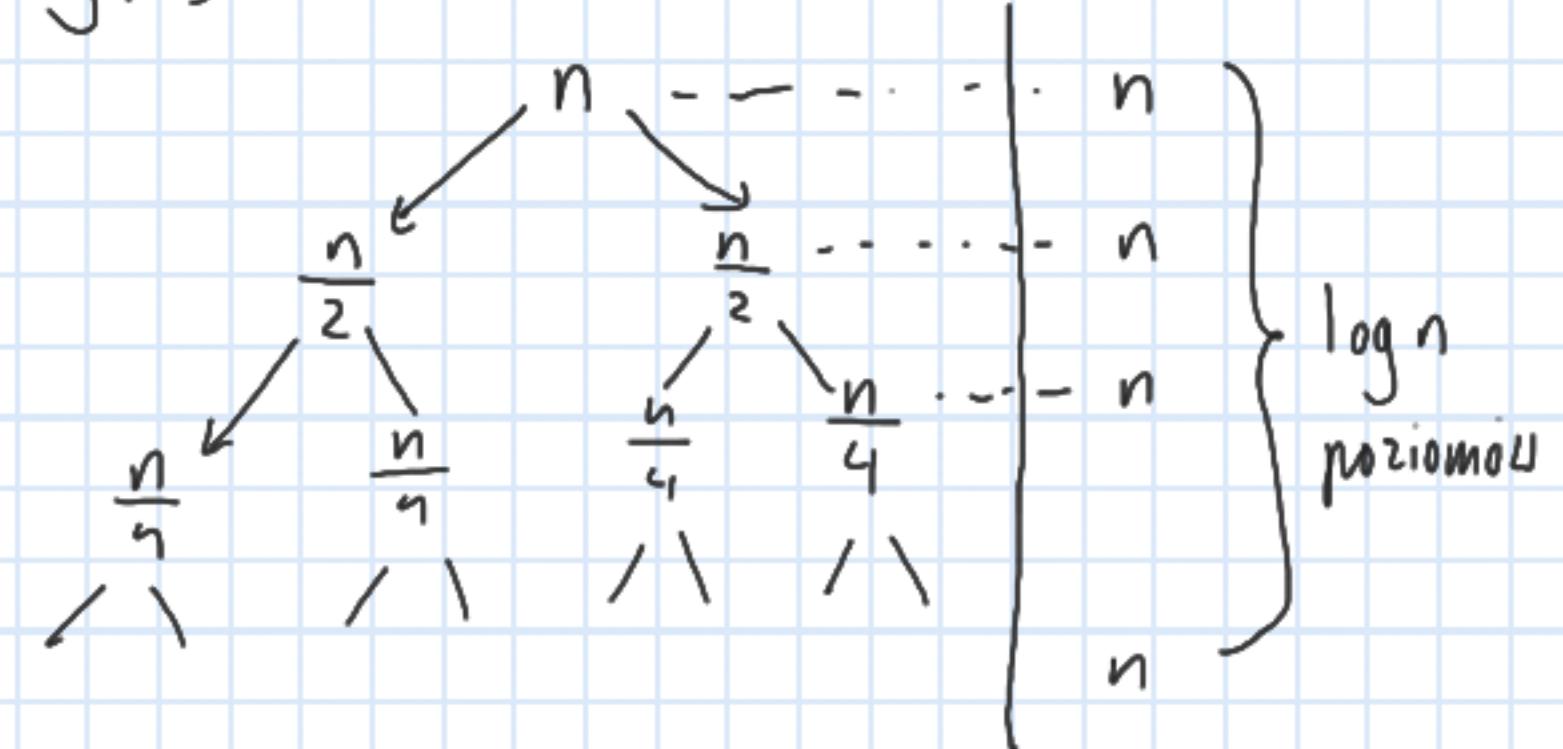
Wyzwic : $\max_{i,j} \left(\sum_{k=i}^j A[k] \right)$



$O(n^3)$ - najbrudzige metoda

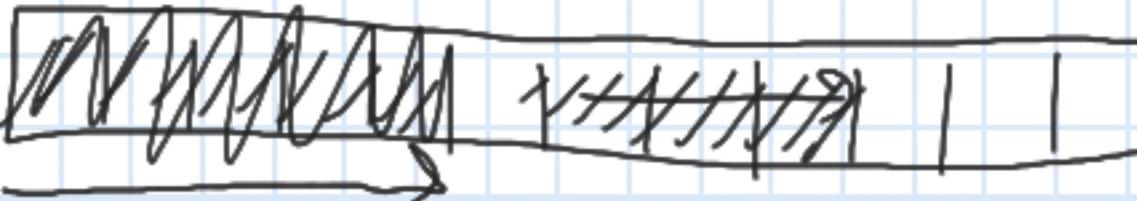


$O(n \log n)$



$O(n^2)$

$$f(i) = \max(f(i-1) + A[i], 0)$$



< 0

$f(i) = \text{maks. wartość rozciągająca końca tego}$

Σn^2

Algorytmy i Struktury Danych

Wykład 2

Zadania offline - można wykonać dopiero
nie określono duch zadani danej osoby

Problem: Sortowanie

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

Zadanie: Znaleźć ciąg a'_1, \dots, a'_n
taką permutacją wejściowego

ciągu, taki że

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

$$a_i = a_j$$

$$a_i \leq a_j \text{ i } a_j \leq a_i$$

Varianty

reprezentacja danych

↳ tablica

↳ lista

↳ plik

1 - kier.
2 - kier.

dzielenie "w miejscu" - alg. sort.
dziela w miejscu, jeli. poza pamięcią
potwierdza na dane wejściowe uzyska
 $O(1)$ pamięci

stabilność - algorytm sortowania jest stabilny, jeśli

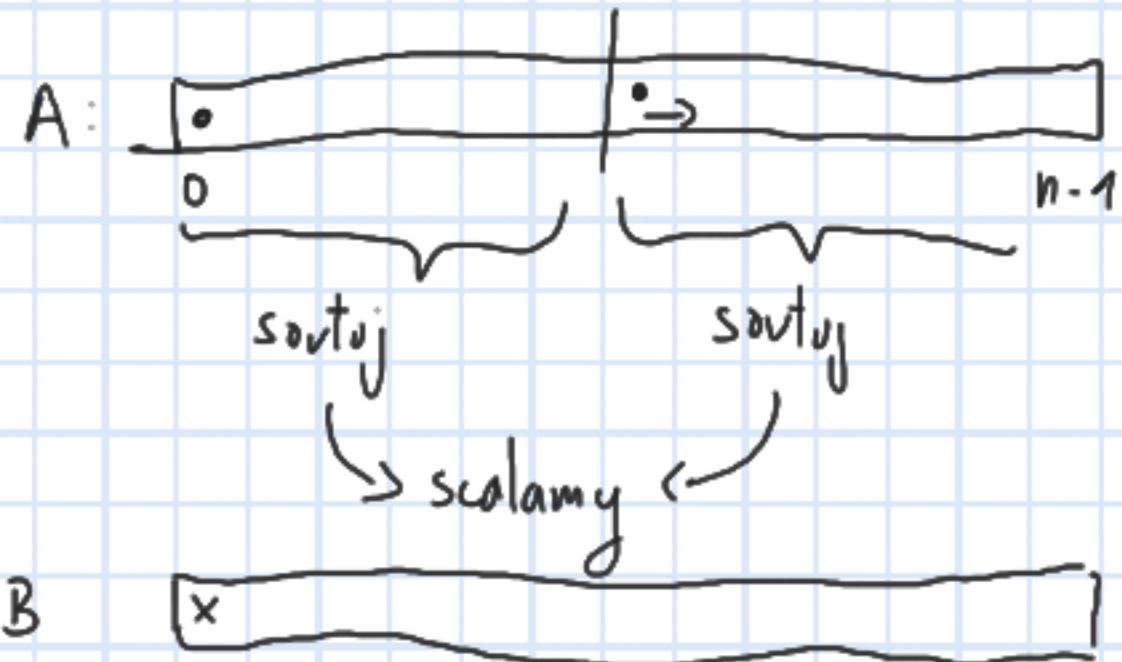
ma własność, że dla kaidy dwóch

$$a_i, a_j \text{ t.z. } a_i = a_j, i < j,$$

↳ posortowanych danych a_i występuje przed a_j

Algorytm Merge Sort - sortowanie przez scalanie

- wykorzystuje technikę dziel i zbywają



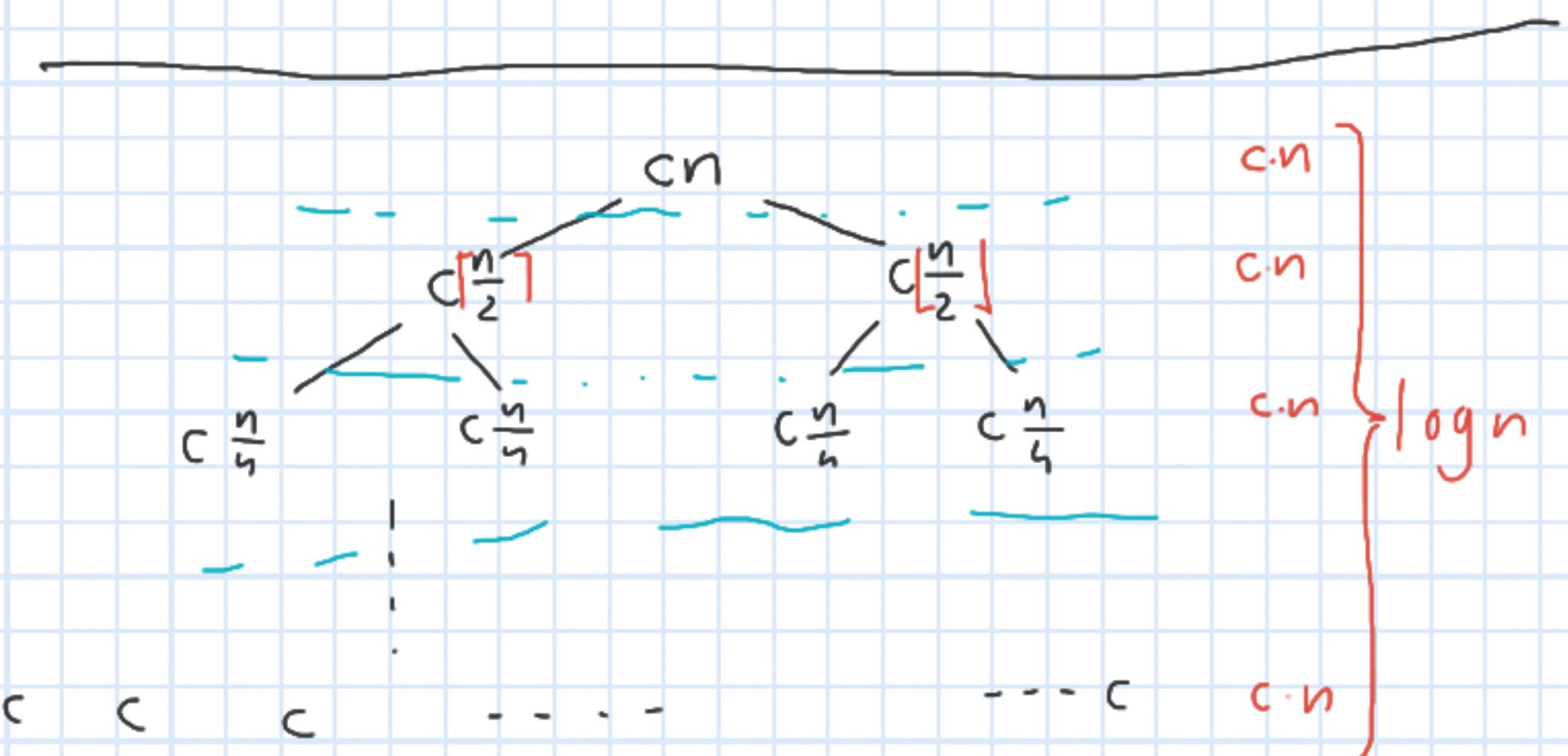
Chcemy oszacować złożoność obliczeniową
MergeSorta

$$T(n) = \begin{cases} c & , \text{ dla } n \leq 1 \\ T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + cn, & \text{upr.} \end{cases}$$

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

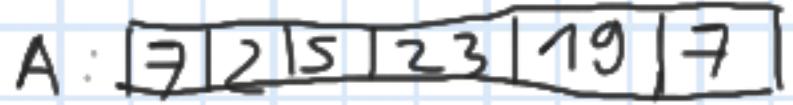
Forsidēm
ī na Tatvīzus

$$\begin{aligned}
 T(n) &= c \cdot n \\
 &+ T\left(\lceil \frac{n}{2} \rceil\right) + T\left(\lfloor \frac{n}{2} \rfloor\right) = \\
 &+ T\left(\frac{n}{3}\right) + T\left(\frac{n}{3}\right) + T\left(\frac{n}{3}\right) + T\left(\frac{n}{3}\right) = \\
 T\left(\frac{n}{3}\right) &\quad T\left(\frac{n}{3}\right) \quad T\left(\frac{n}{3}\right) \quad T\left(\frac{n}{3}\right)
 \end{aligned}$$



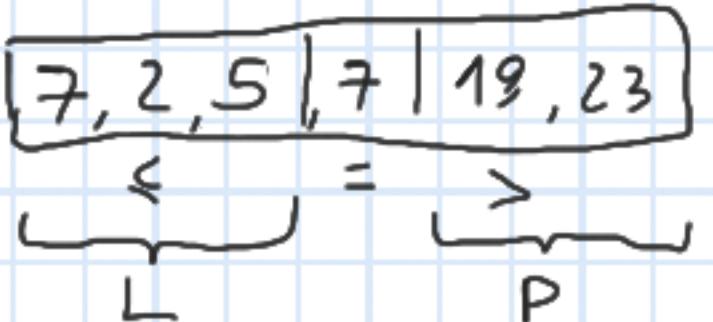
Algorytm QuickSort

Takie działa na zasadzie "dziel i zryciżaj"

A: 

① wybierz pivot, np $x = 7$

② podziel tablicę na elementy \leq od pivotu
 oraz $>$ od pivotu



③ posortuj rekurencyjnie L i P

Gdyby partition zawsze dzielił tablicę na

półowy, to QuickSort działałby w czasie:

$$T(n) = \begin{cases} c, & n \leq 1 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + cn, & \text{w.p.} \end{cases}$$

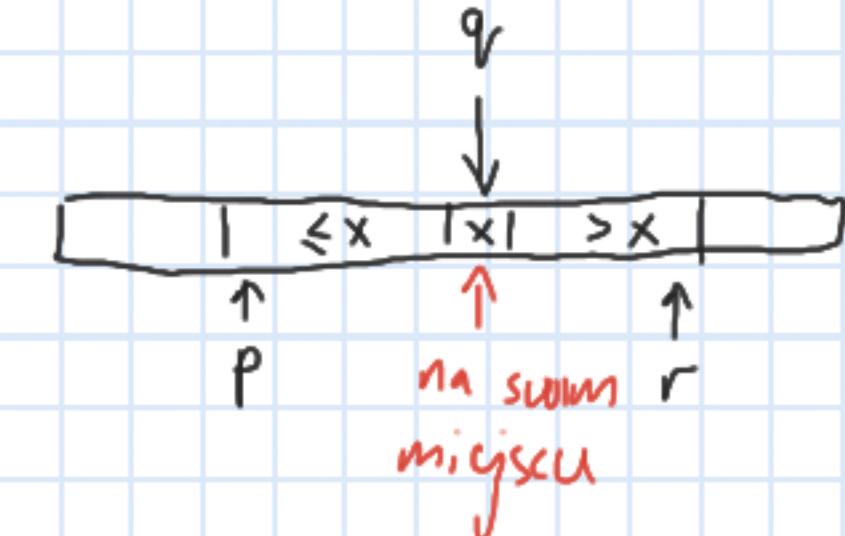
angi: w wariancie $O(n \log n)$

def quicksort(A, p, r):
if $p < r$:

$q = \text{partition}(A, p, r)$

quicksort(A, p, q - 1)

quicksort(A, q + 1, r)



def partition(A, p, r):

$x = A[r]$

$i = p - 1$

for j in range(p, r):

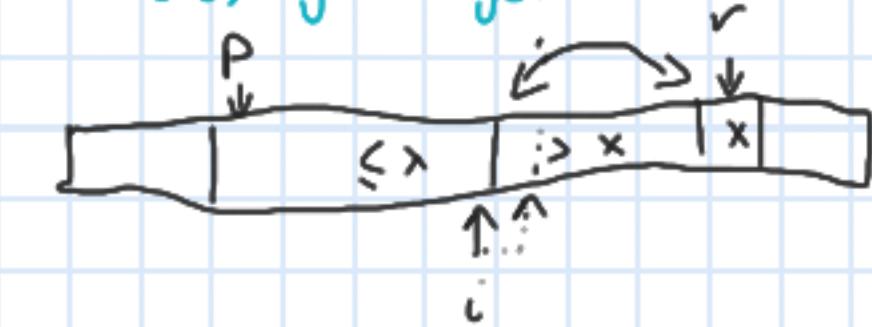
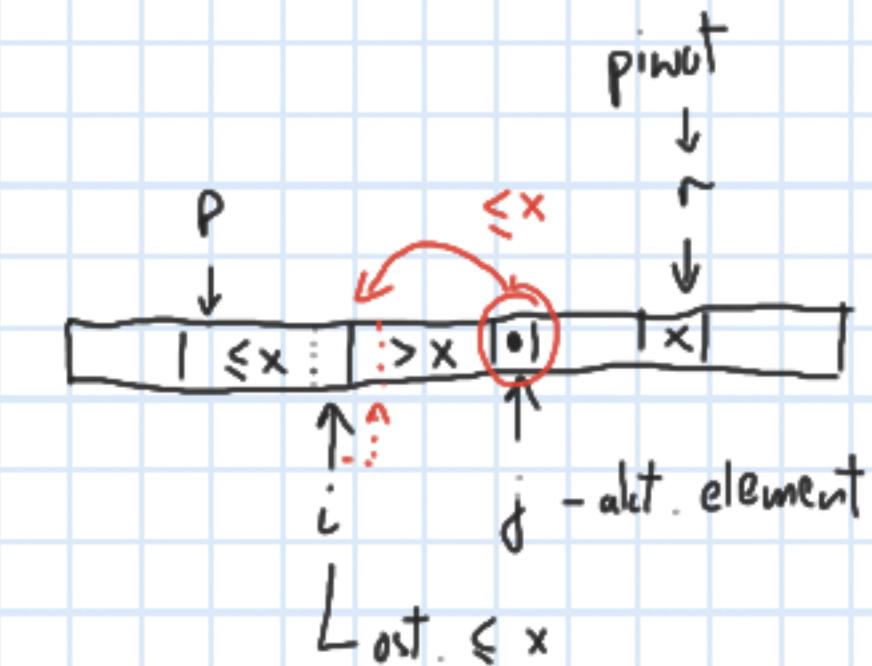
if $A[j] \leq x$:

$i += 1$

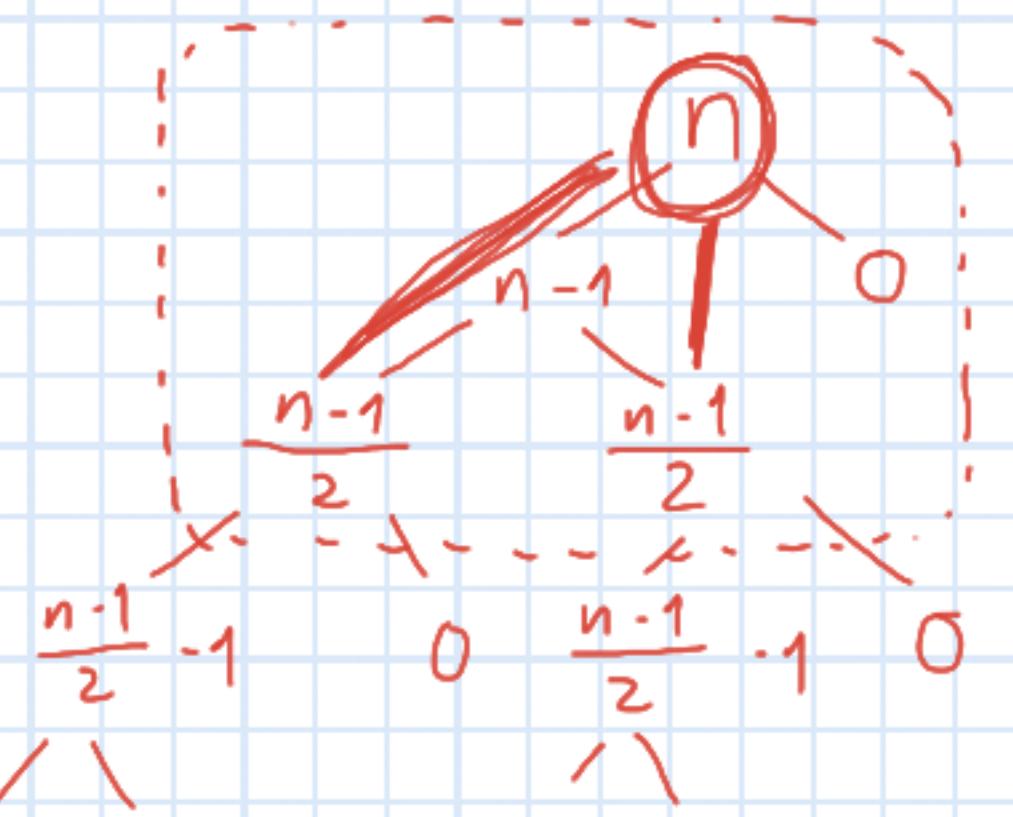
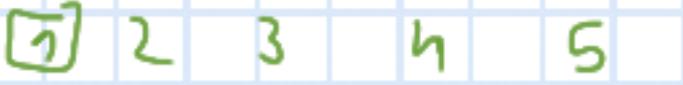
swap($A[i], A[j]$) $\leftarrow A[i], A[j] = A[j], A[i]$

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

return $i + 1$



Dla niekonstnych danych Quicksort może działać w czasie $O(n^2)$

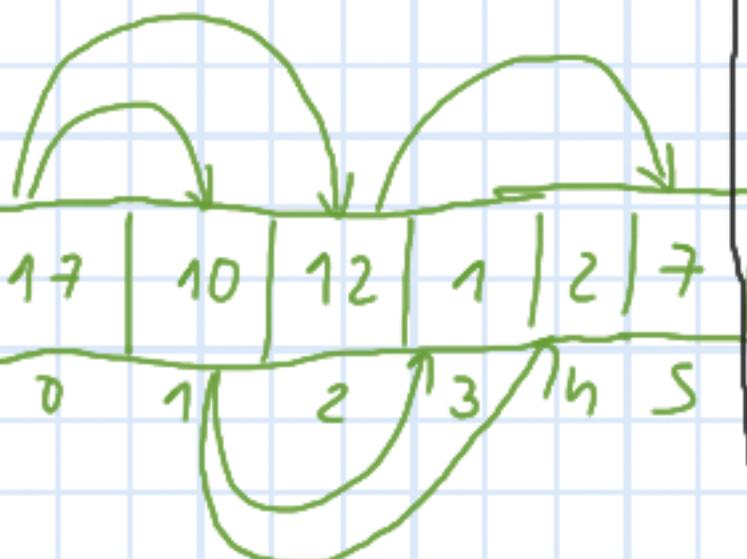
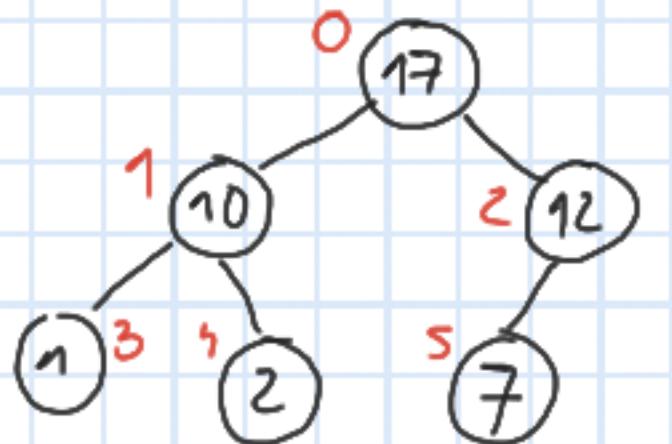


Zadanie obowiązkowe: proszę zaimplementować Quicksort tak, aby używać najwyżej $O(\log n)$ dodatkowej pamięci

```
def quicksort(A, p, r):
    while p < r:
        q = partition(A, p, r)
        Quicksort(A, p, q-1)
        p = q + 1
```

HeapSort - sortowanie przez kopcowanie

Kopiec - drzewo binarne, gdzie każdy węzeł ma wartość \geq niż jego dzieci



$$\text{left}(i) = 2i + 1$$

$$\text{right}(i) = 2i + 2$$

$$\text{parent}(i) = \lfloor (i-1)/2 \rfloor$$

Zadanie obowiązkowe: Proszę zaimplementować usuwanie danego elementu do kopca

def heapify(A, n, i):

l = left(i)

r = right(i)

m = i

if l < n and A[l] > A[m]: m = l

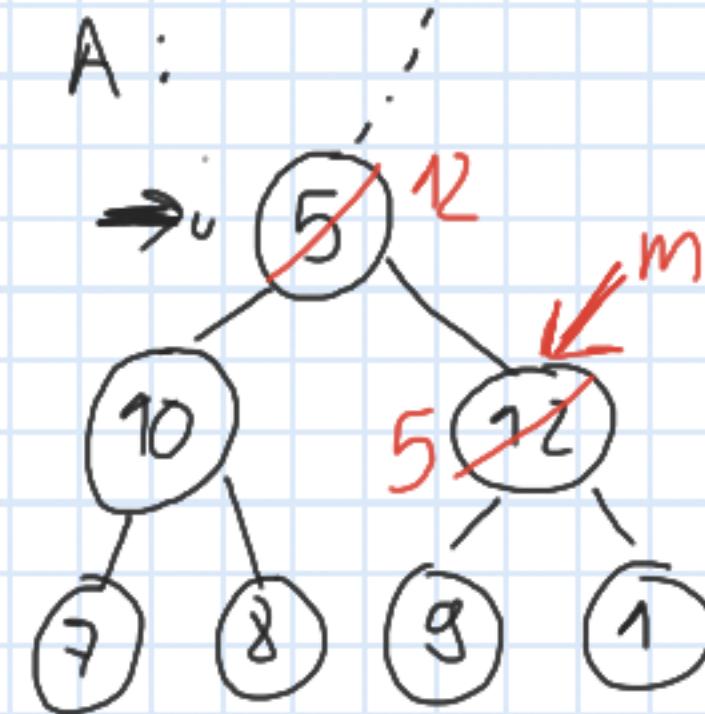
if r < n and A[r] > A[m]: m = r

if m ≠ i:

swap(A[i], A[m])

heapify(A, n, m)

A:



$O(\log n)$

def buildheap(A):

n = len(A)

for i = parent(n-1) to 0

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

heapify(A, n, i)

$O(n \log n)$ / $O(n)$

```
def heapSort(A):
```

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

```
    buildHeap(A)
```

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

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

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

$O(n \log n)$

Czas działania buildheap



$\lceil \frac{n}{2^{h+1}} \rceil$ - w kopcu o n elementach tyle jest "podkopciów" o wysokości h

$$\sum_{h=0}^{\lfloor \log n \rfloor} \lceil \frac{n}{2^{h+1}} \rceil \cdot h = O\left(\sum_{h=0}^{\lfloor \log n \rfloor} \frac{n \cdot h}{2^{h+1}}\right) = O(n)$$

$$n \cdot \sum_{h=0}^{\infty} \frac{h}{2^{h+1}}$$

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

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

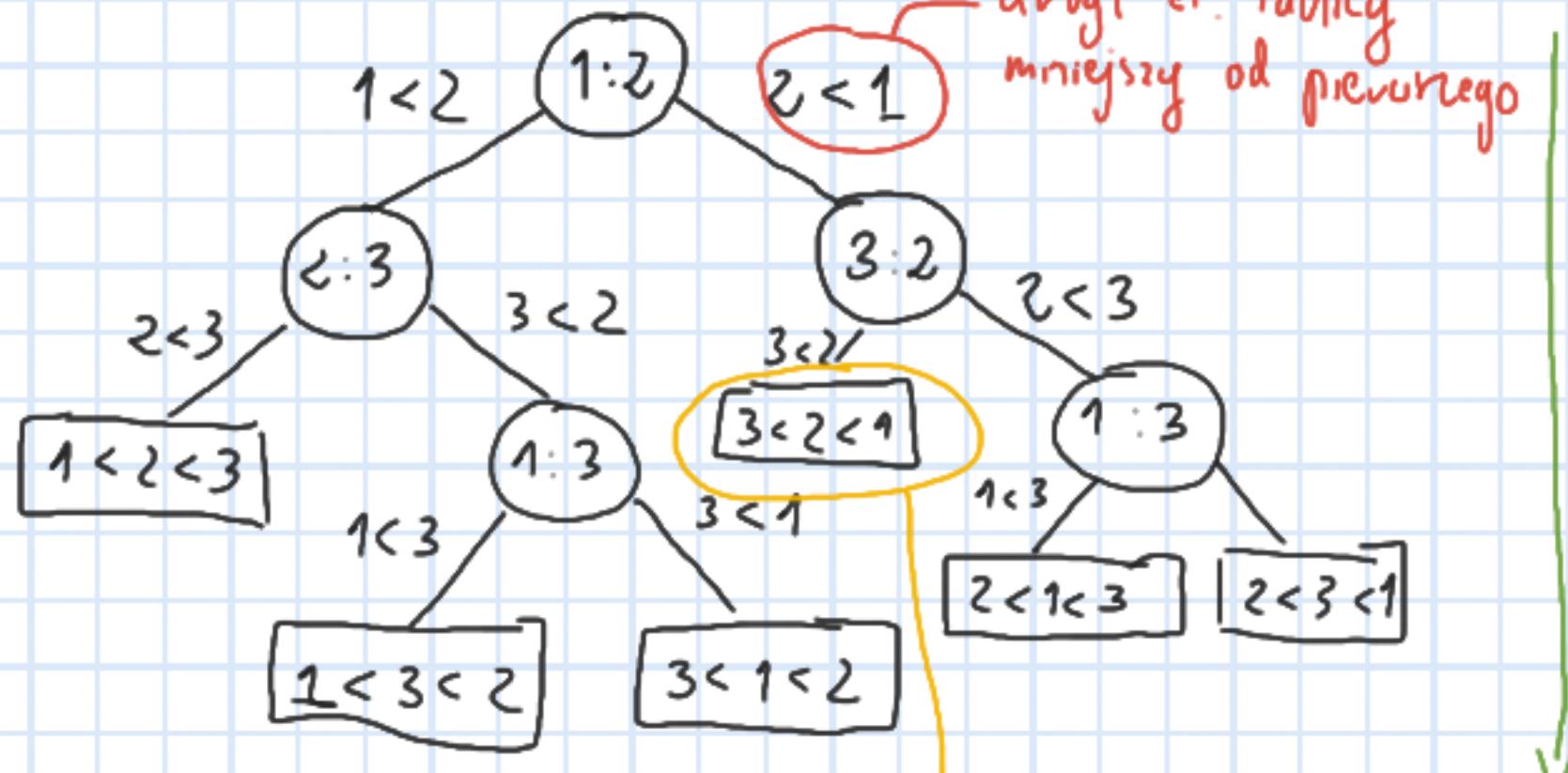
$$x f'(x) = x + 2x^2 + 3x^3 + \dots = \frac{x}{(1-x)^2} = 2$$

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

Algorytmy i Struktury Danych

Wykład 3

Dolne ograniczenie na szybkość sortowania



Każda permutacja musi się pojawić jako liść

Tаблицa n elementowa $\Rightarrow n!$ permutacji

Dzewo binarne wysokości h ma najwyższą 2^h liczbą liści

Jesli nasze dzewo jest wysokością h , to $n! \leq 2^h \Rightarrow h \geq \log(n!) \Rightarrow h \text{ jest \geq } n \log n$

$$1 \cdot 2 \cdots \frac{n}{2} \underbrace{\cdots}_{\frac{n}{2}} \frac{n}{2} \cdots \frac{n}{2}$$

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

$$\frac{n}{2}(\log n - 1)$$

$\Theta(n \log n)$

Wykazanie tego
dowód to liczba
permutacji, którą
algorytm wykonyje
w najgorszym przypadku

$n \log n$

Sortowanie przez zliczanie

Chcemy posortować pewną tablicę rozmiaru n , zaczynającą się linią od 0 do $k-1$
naturalne

```
def countsort(A, k):
```

```
C = [0] * k
```

```
B = [0] * len(A)
```

```
for i in range(len(A)):
```

```
C[A[i]] += 1
```

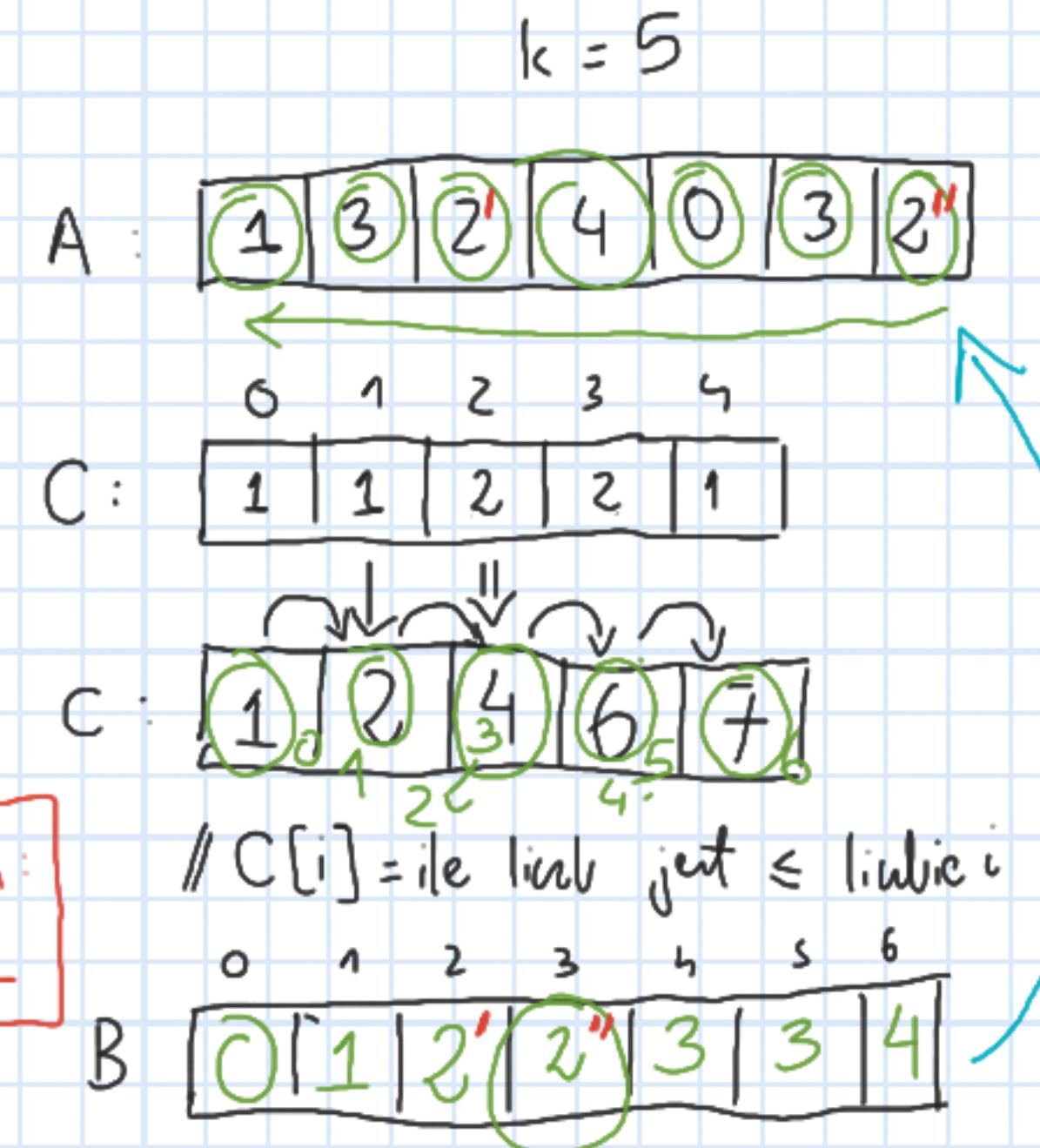
```
for x in A:  
    C[x] += 1
```

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

```
for i in range(len(A) - 1, -1, -1):  
    C[A[i]] -= 1
```

```
B[C[A[i]]] = A[i]
```

$O(n+k)$



```
for i in range(len(A)):  
    A[i] = B[i]
```

Zadanie obowiązkowe

Proszę zaproponować jak-najszybszy algorytm sortujący n elementową tablicę zaczynającą się linią ze zbioru $\{0, 1, 2, \dots, n^2 - 1\}$.

Sortowanie kubatkowe

Sortujemy tablice n liczb powodznych
z rozkładu jednostajnego nad $[0, 1)$

0.42, 0.93, 0.07, 0.21, 0.91, 0.13, 0.37

tuowymy n kubatków // list jednokierunkowych

$[0, 0.15)$ $[0.15, 0.30)$ $[0.30, 0.45)$ $[0.45, 0.6)$ $[0.6, 0.75)$ $[0.75, 0.9)$ $[0.9, 1)$

0.13

0.07

0.13

0.21

0.42

0.37

0.07

0.13

0.13

0.07, 0.13, 0.13, 0.21, 0.37, 0.42, 0.91

$O(n)$ gdy spłnione
założenia

$x \in [0, 1)$

x leży w kubatku

$\lfloor x \cdot n \rfloor$

\downarrow

0.91

Radix Sort - sortowanie pozycyjne

kra
avr
kot
kit
ati
kil

kra
ati
kil
avr
kot
kit

\Rightarrow

kil
ati
avr
kot
art
ati

avr
ati
kil
kot
kra
art

avr
ati
kil
kot
kra
art

Zadanie obowiązkowe

Tablica T jest długosci n , ale zawiera
tylko $\lceil \log n \rceil$ różnych wartości. Proszę
zaproponować jaknajszyszybszy algorytm sortujący
taką tablicę.

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

$$\begin{aligned} T(n) &= cn + c \frac{n}{2} + c \frac{n}{4} + c \frac{n}{8} + \dots + c \frac{n}{2^{\lceil \log n \rceil}} \\ &= cn \left(1 + \frac{1}{2} + \frac{1}{4} + \dots\right) \leq 2cn \end{aligned}$$

Statystyki pozycyjne

Zadanie: Wyznaczyć element, który po posortowaniu
tablicy znajdzie się na pozycji k -ej

Przykłady elementarne:

$k=0 \rightarrow \text{minimum}$

$k=n-1 \rightarrow \text{maksimum}$

def select(A, p, r, k):

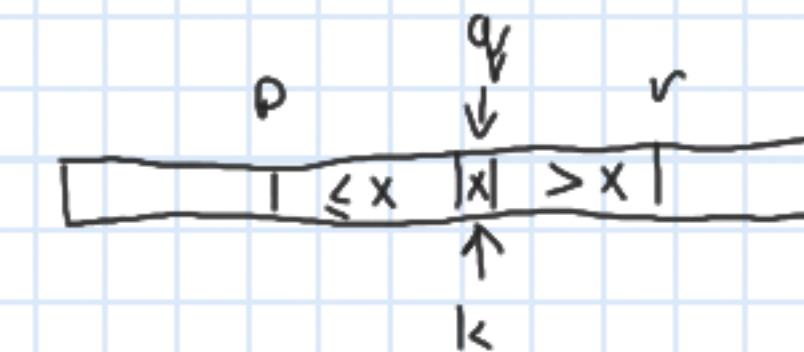
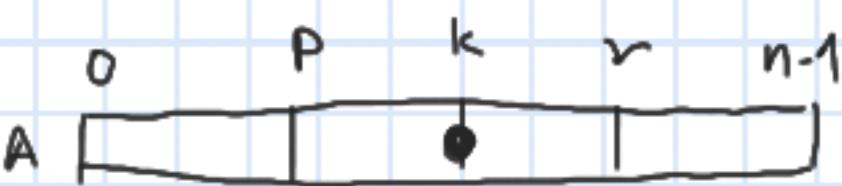
if $p == r$:
 return $A[p]$

$q = \text{partition}(A, p, r)$

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

elif $k < q$:
 return select($A, p, q-1, k$)

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



Algorytm: Magiczne Piątki

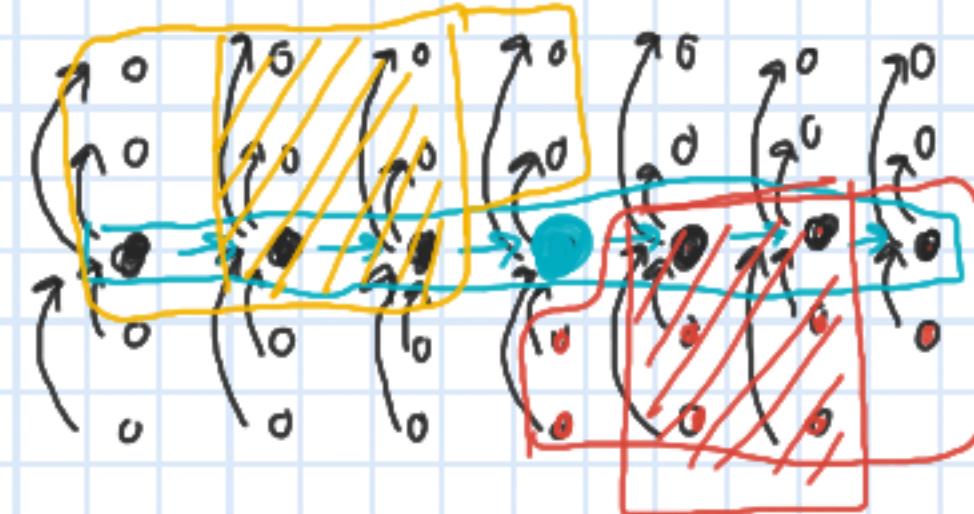
A - wejściowa n-elementowa tablica

① Podziel wejściową tablicę na $\lceil \frac{n}{5} \rceil$ grup po 5 elementów

② W każdej grupie wyznacz medianę

③ Rekurencyjnie wyznacz jako medianę medianę

④ Kontynuujemy jak w algorytmie select, traktując jak pivot pmy podziały



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

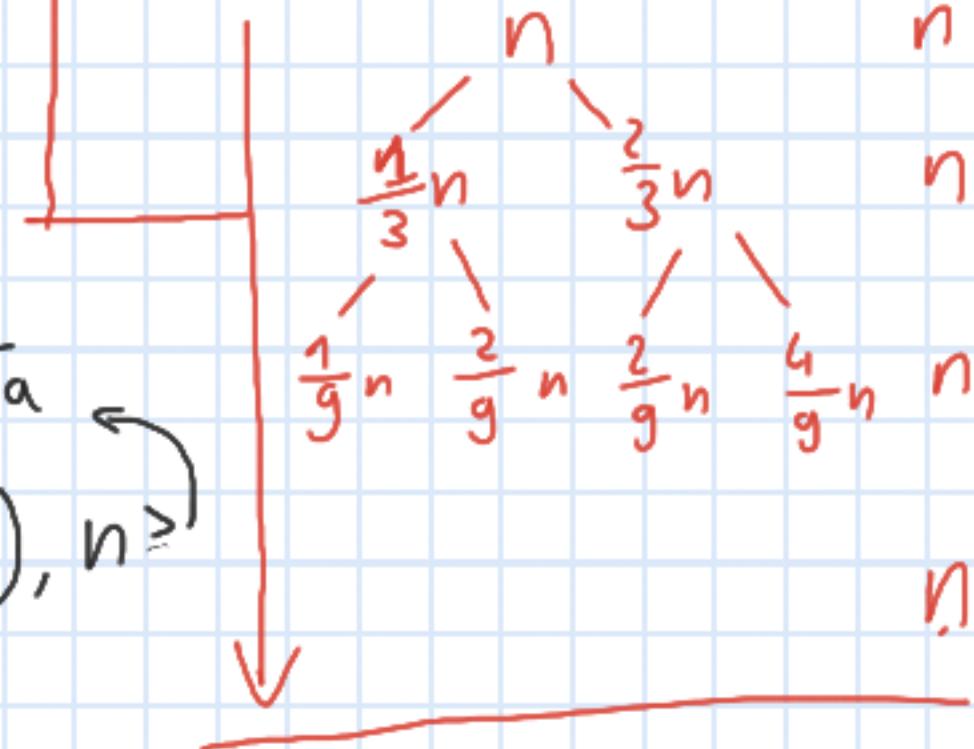
Złożoność czasowa

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

$\downarrow \frac{3}{10}n + \frac{7}{10}n = \frac{9}{10}n$

$$\begin{aligned} T(n) &= T\left(\lceil \frac{n}{3} \rceil\right) + T\left(\frac{2}{3}n\right) \\ &\Downarrow \frac{n}{3} + \frac{2}{3}n \end{aligned}$$

$$3 \cdot \left(\lceil \frac{1}{2} \lceil \frac{n}{3} \rceil \rceil - 2 \right) \approx \frac{1}{3}n$$



Twierdzimy, że $T(n) \leq cn$

Dowód przez indukcję:

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

Algorytmy i Struktury Danych

Wykład 4

Absyktacyjne struktury danych

↳ "kontekst" co do działania

↳ zestaw operacji

↳ "fizyczna" realizacja

↳ tablica

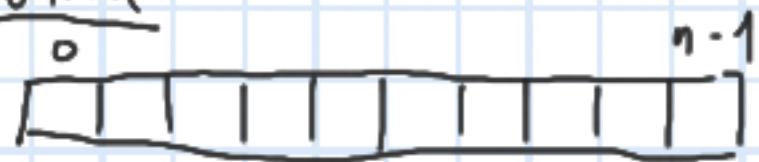
↳ lista

↳ drees

↳ graf

↳ ...

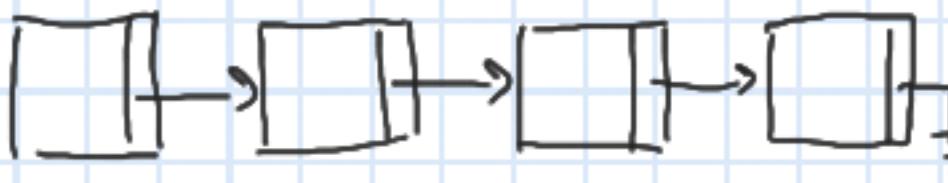
Tablica



"Coś, co pozwala odróżnić się do pól po ich numerach"

Priegład		
0	1	2
0	1	2
1	3	4
2	6	7

Lista jedno/dwukierunkowa

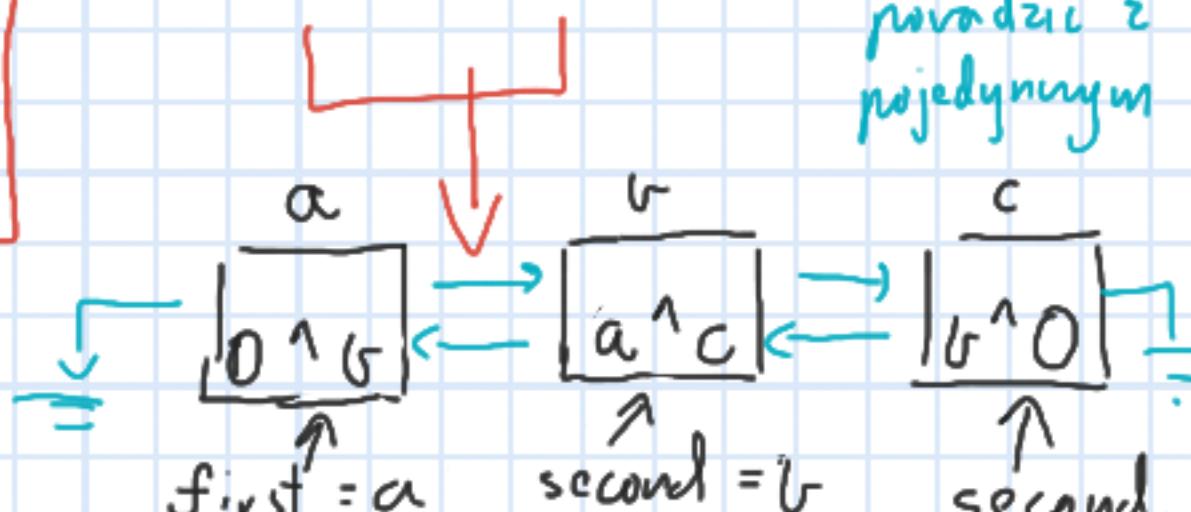


"Coś, co pozwala znaleźć punktek ciągu elementów, przenosząc się po jednym elemencie do przodu, wstawiając i usuwając elementy"



Lista dwukierunkowa

można służyć pozwodzić z pojedynczym uszczepieniem



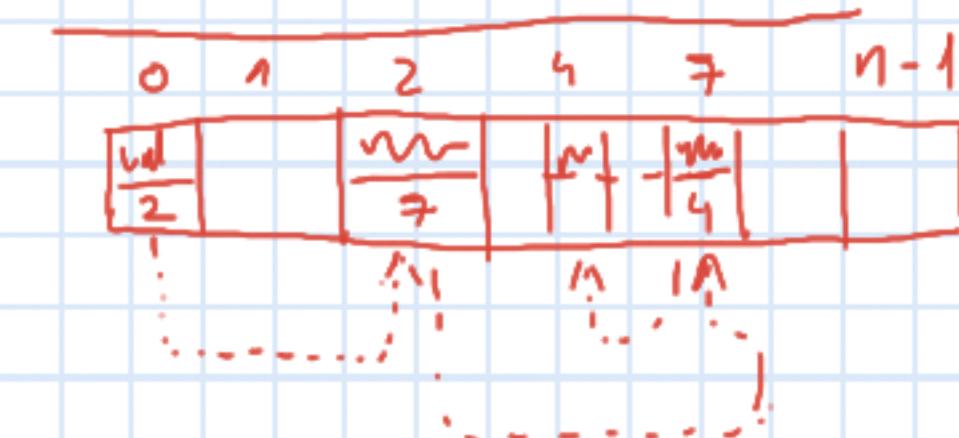
$$(a \wedge c) \wedge a = c$$

class Node:

def __init__(self):

self.next = None

self.value = None



class DNode:

def __init__(self):

{ self.next = None

self.prev = None }

$$0 \wedge 0 = 0$$

$$0 \wedge 1 = 1$$

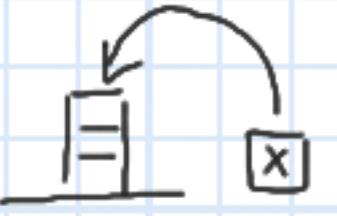
$$1 \wedge 0 = 1$$

$$1 \wedge 1 = 0$$

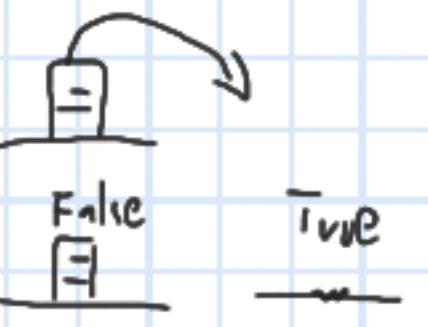
Stos

"Coś co pozwala odkładać elementy na sznur i zdejmować je w odwrotnej kolejności"

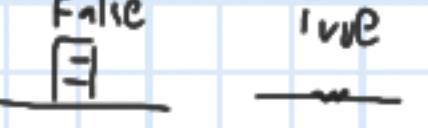
`push(s, x)`



`pop(s)`



`is-empty(s)`



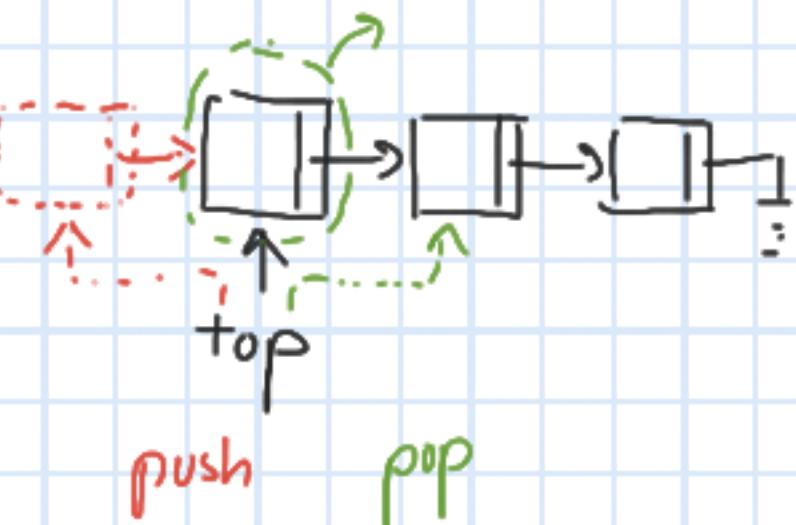
organizacja

pamięci w komp.

IBN PC XT/AT

LIFO - last in / first out

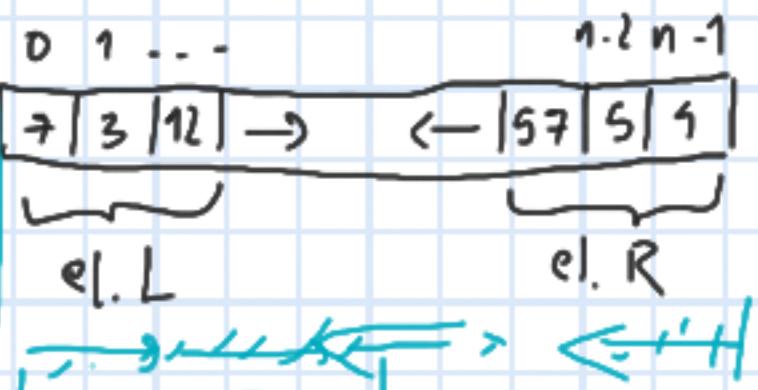
Implementacja listowa



Implementacja tablicowa

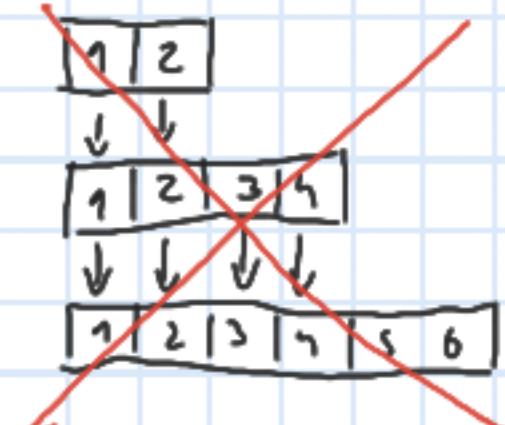


Dwa stosy



append u pythonic

Powiększanie stosu (analiza zamontowanej)



Jesli push powodowałby

wyjście poza zakres

tablicy, to alokujemy **2x** wiekowy stos

Uprowadzamy cennik

`push` → 3 zT

`pop` → 1 zT

2x

3x

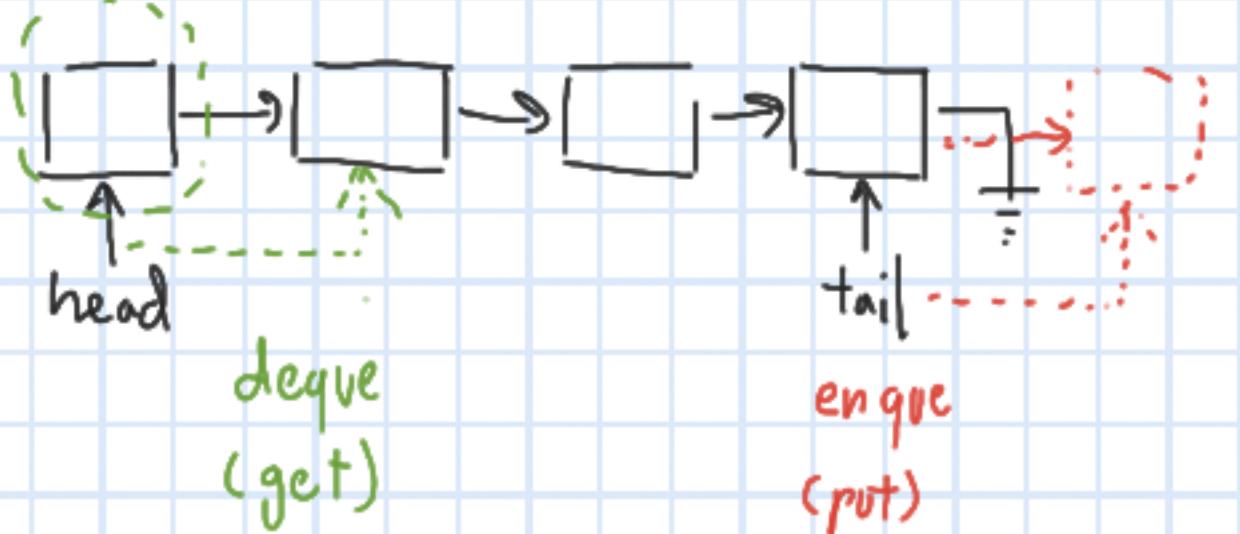
1.2

Jakie uprowadzić cennik i jak realizować funkcję `pop`, jeśli chcemy żeby stos mógł mieć

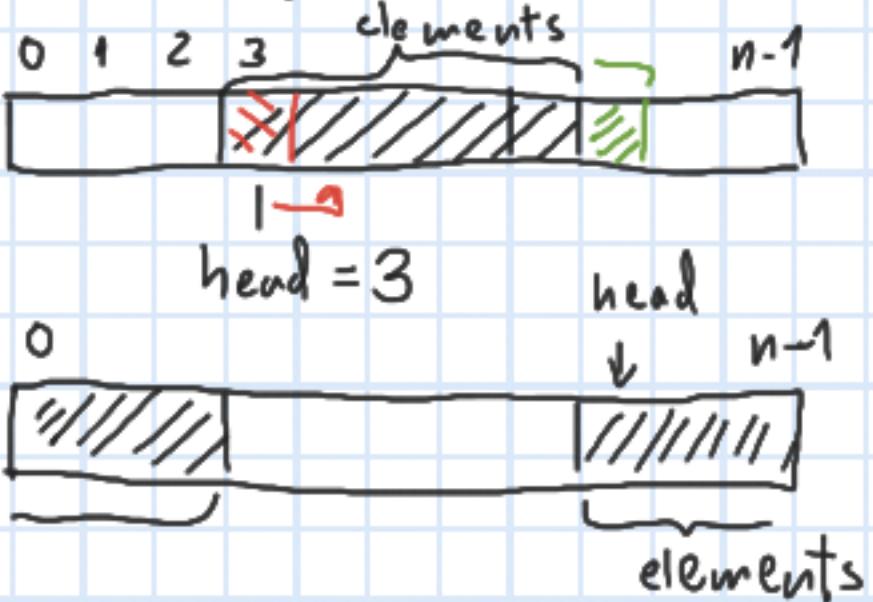
Kolejka (FIFO - first in / first out)

"Cos co pozuala ustawiaje na koniec i odnosi sie z punktu"

Implementacja listowa



Implementacja tablicowa



Kolejka priorytetowa

"Cos co pozuala gromadzic elementy razem z priorytetem i, ujmowac el. o najwyzszyim priorytecie"

Implementacja

- kopiec binarny
- tablica (posortowana lub nie)
- lista (posortowana lub nie)

- insert(el, x)
- extract-max()
- { - increase/decrease key

Jak zaimplementowac

kolejke majac dva stosy? (tak, zely dziala)

w zamontowanym czesciu statym

Tablica asocacyjna

Cos, co zachowuje sie jak tablica indeksowana dwuwartym typem

$$T["\text{Stone}"] = 27$$

Metody konstruowania algorytmów

- metoda dziel : zuygaj
podzieli problem na kilka (niezależnych) części, rozwiąż je i połącz wyniki
 - metody zachłanne

- metody zaktanne
 - w każdym kroku algorytmu wie to co ugdaje się najlepsze "w tej chwili"
 - programowanie dynamiczne
 - dzielimy problem na (zadodatkując na sicie) podproblemy, rozwiązyujemy je zapamiętując wyniki wątkowe i tyczymy rozwiązanie w całości

metoda zamiany wykładowego algorytmu
rekurencyjnego na (iteracyjny) algorytm
wielomianowy

Prykład prog. dynamicznego

$$F_1 = 1$$

$$F_2 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$

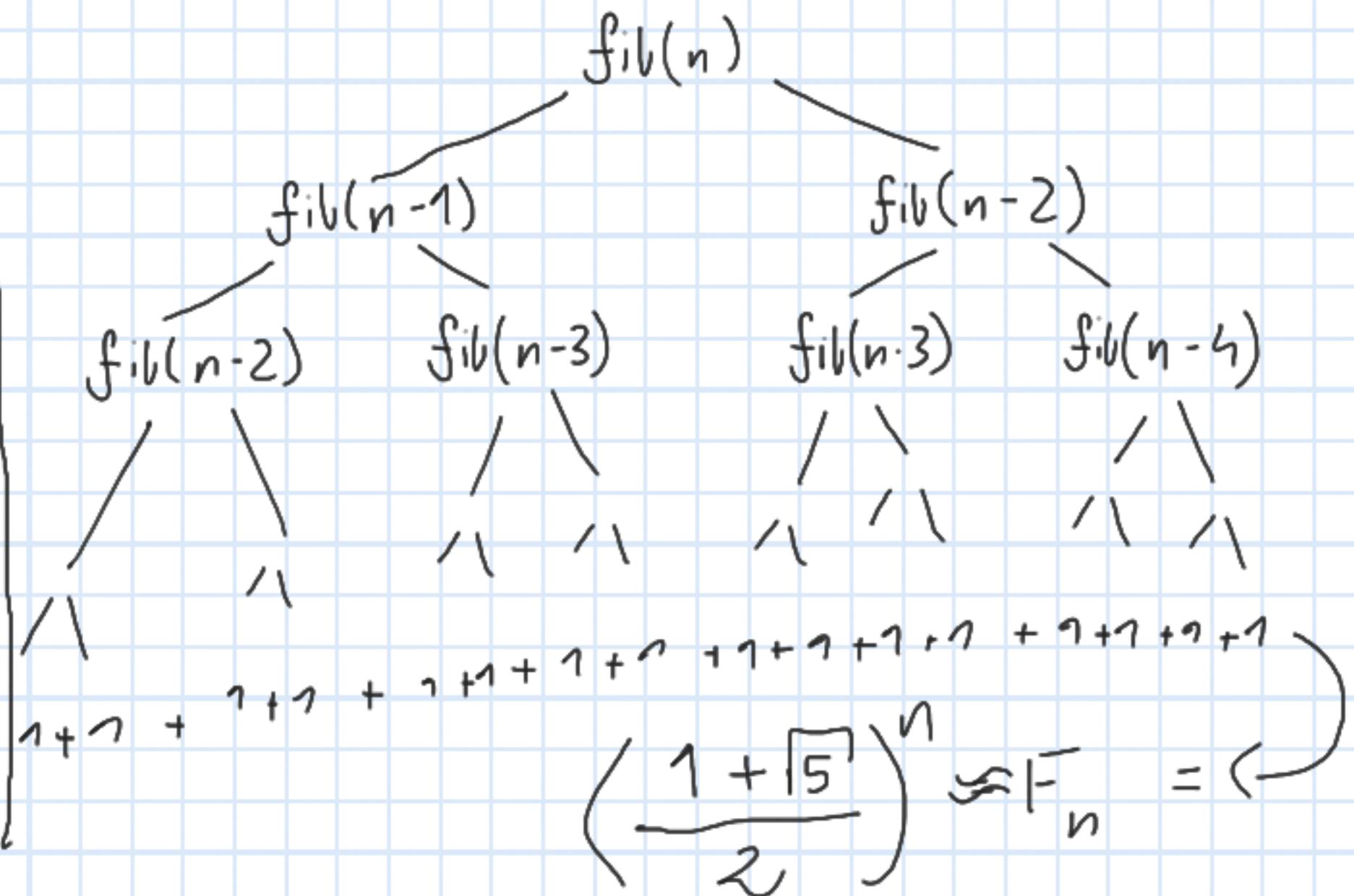
def fib(n):

if $n \leq 2$:

return 1

else:

return fib(n-1) + fib(n-2)



def dpfib(n):

$$F = [1] \times (n + 1)$$

for i in range(3, n+1):

$$F[i] = F[i-1] + F[i-2]$$

return F[n]