

## Zad 1.

Napisz rekurencyjne funkcje, które dla danego drzewa binarnego T obliczają:

a) liczbę wierzchołków w T

b) maksymalną odległość między wierzchołkami w T

a) procedure count(node)

if (node == NULL)

return 0

return (1 + count(node.left) + count(node.right))

$O(n)$

Dla każdego wierzchołka liczymy ile jest wierzchołków „pod nim” i dodajemy 1 (on sam). Liście mają atrybuty .right i .left równe NULL, więc w takim wypadku procedura count zwróci 0.

b) procedure diameter(node, \*height)

LH, RH, LD, RD = 0

if (node == NULL)

\*height = 0

return 0

$O(n)$

LD = diameter(node.left, &LH)

RD = diameter(node.right, &RH)

\*height = max(LH, RH) + 1

return max(LH+RH+1, LD, RD)

Zauważamy że średnica drzewa jest równa średnicy lewego lub prawego poddrzewa lub wysokości prawego i lewego drzewa +1.

### Zad 3

#### Leksykograficzne sortowanie topologiczne

$L \leftarrow$  Empty List

$S \leftarrow$  Priority Queue containing all nodes with no incoming edge

while  $S$  is not empty:

remove node  $n$  from  $S$

$O(|V| + |E|)$

add  $n$  to  $L$

for each edge  $(n m)$

remove  $(n m)$  from graph,

if  $m$  has no incoming edges:

insert  $m$  into  $S$

if graph has edges

return "Graph has a cycle"

else

return  $L$

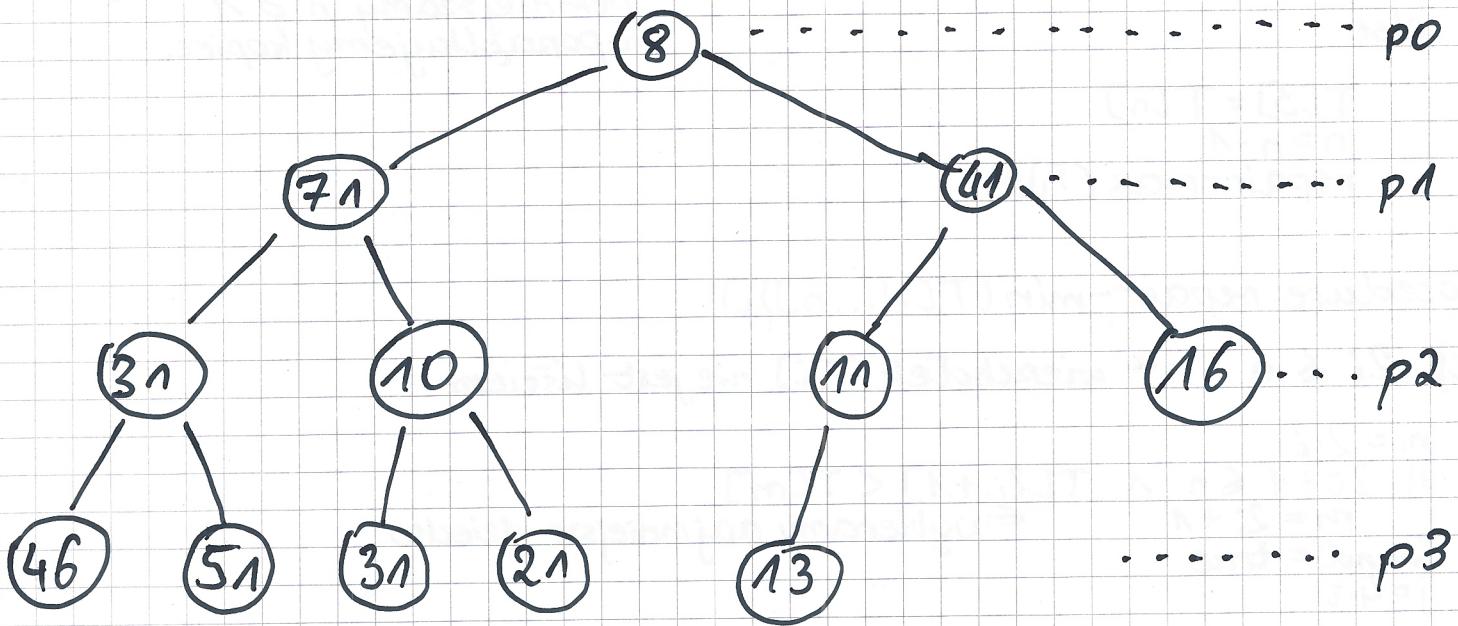
## Zad 2

Wapisz w pseudokodzie procedury

- przywracania porządku
- usuwania minimum
- usuwania maksimum

z kopca minimaksowego.

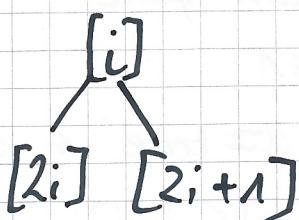
Kopiec minimax:



Każdy węzeł na parzystym poziomie jest mniejszy niż wszyscy jego potomkowie, a każdy węzeł na nieparzystym poziomie jest większy niż wszyscy jego potomkowie.

Kopiec jest reprezentowany jako tablica.

Element z indeksem  $i$  znajduje się na poziomie  $\lfloor \log_2(i) \rfloor$



procedure delete-min ( $T[1..n]$ )

$T[1] = T[n]$

$n = n - 1$

repair-min ( $T, 1$ )

procedure delete-max ( $T[1..n]$ )

if  $T[2] > T[3]$

$T[2] = T[n]$

$n = n - 1$

repair-max ( $T, 2$ )

else

$T[3] = T[n]$

$n = n - 1$

repair-max ( $T, 3$ )

procedure repair-min ( $T[1..n], i$ )

if  $2i \leq n$  ← wierzchołek  $T[i]$  nie jest liściem

$m = 2 \cdot i$

if  $2i + 1 \leq n \wedge T[2i + 1] < T[m]$

$m = 2i + 1$

← wybieramy najmniejsze dziecko

found = true

$j = 4 \cdot i$

while  $j < \min(4i + 4, n + 1)$  ← przeglądamy uniki

if  $T[j] < T[m]$

$m = j$

found = false

$j = j + 1$

if found = true ← jeśli jako najmniejsze znaleźliśmy dziecko

if  $T[m] < T[i]$ :

$tmp = T[i]$

$T[i] = T[m]$

$T[m] = tmp$

else ← jeśli jeden z uników był najmniejszy

if  $T[m] < T[i]$

$tmp = T[i]$

$T[i] = T[m]$

$T[m] = tmp$

if  $T[m] > T[\lfloor \frac{m}{2} \rfloor]$ :

W kopcu minimaksowym element minimalny znajduje się, zawsze na początku. Wystarczy wstać na jego miejsce ostatni element w tablicy i uporządkować kopiec, po ponowniszeniu  $n \leftarrow 1$ .

Element maksymalny znajduje się na drugim lub trzecim miejscu tablicy, na jego miejsce element na końcu tablicy, ponownie znamy  $n \leftarrow 1$  i porządkujemy kopiec.

$\begin{cases} \text{tmp} = T[m] \\ T[m] = T[\lfloor \frac{m}{2} \rfloor] \\ T[\lfloor \frac{m}{2} \rfloor] = tmp \end{cases}$

repair-min ( $T, m$ )

procedure repair-max( $T[1..n]$ ,  $i$ )

if  $2i \leq n$

$m = 2i$

if  $2i+1 \leq n \wedge T[2i+1] > T[m]$

$m = 2i+1$

found = true

$j = 4i$

while  $j < \min(4i+4, n+1)$

if  $T[j] > T[m]$

$m = j$

found = false

$j = j + 1$

if found == true

if  $T[m] > T[i]$

$tmp = T[i]$

$T[i] = T[m]$

$T[m] = tmp$

else if  $T[m] > T[i]$

$tmp = T[i]$

$T[i] = T[m]$

$T[m] = tmp$

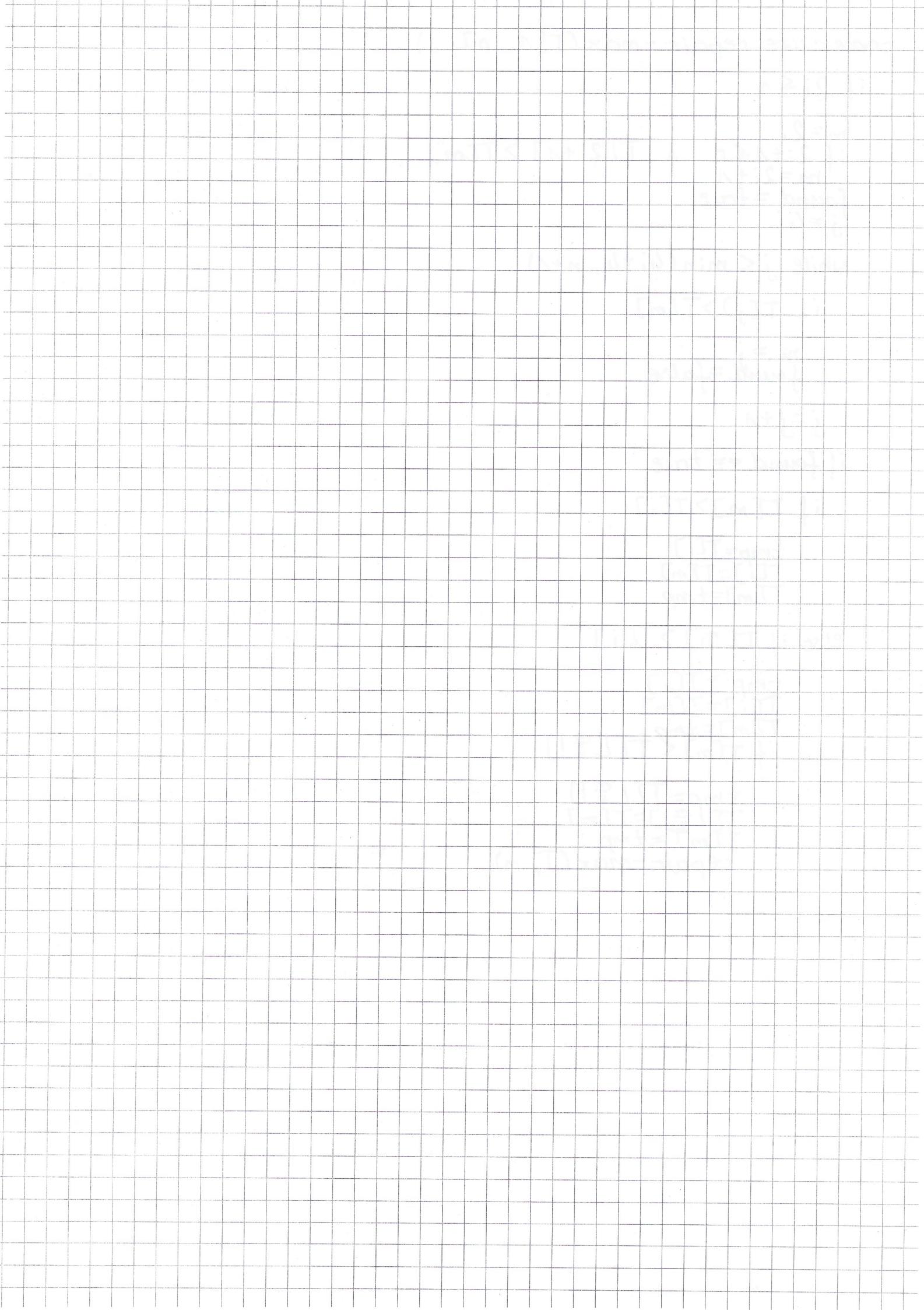
if  $T[m] < T[\lfloor \frac{m}{2} \rfloor]$

$tmp = T[\lfloor \frac{m}{2} \rfloor]$

$T[\lfloor \frac{m}{2} \rfloor] = T[m]$

$T[m] = tmp$

repair-max( $T, m$ )



## Zad 4

za pomocą algorytmu Dijkstry wyznaczymy odległość każdego wierzchołka od  $v$ . Następnie zaczynając od u będziemy przemieszczać się po sąsiednich wierzchołkach. Te 2 nich, które są bliżej  $v$  niż  $u$  leżą na sensorych ścieżkach. Wtedy rekurencyjnie wyznaczymy dla nich ilość sensorych ścieżek mnoiąc wynik przez ilość sensorych ścieżek w poprzednim wierzchołku na ścieżce.

function ścieżki

for  $i$  in  $0 \dots |V|-1$

odwiedzony [ $i$ ] = false

ścieżki [ $i$ ] = 0

ścieżki [ $v$ ] = 1

odwiedzony [ $v$ ] = true

odległości [ $0 \dots |V|-1$ ] = dijkstra ( $G, v$ )

wyznacz ( $u$ )

return ścieżki

function wyznacz ( $w$ )

foreach  $x$  in  $w$ .neighbourhood

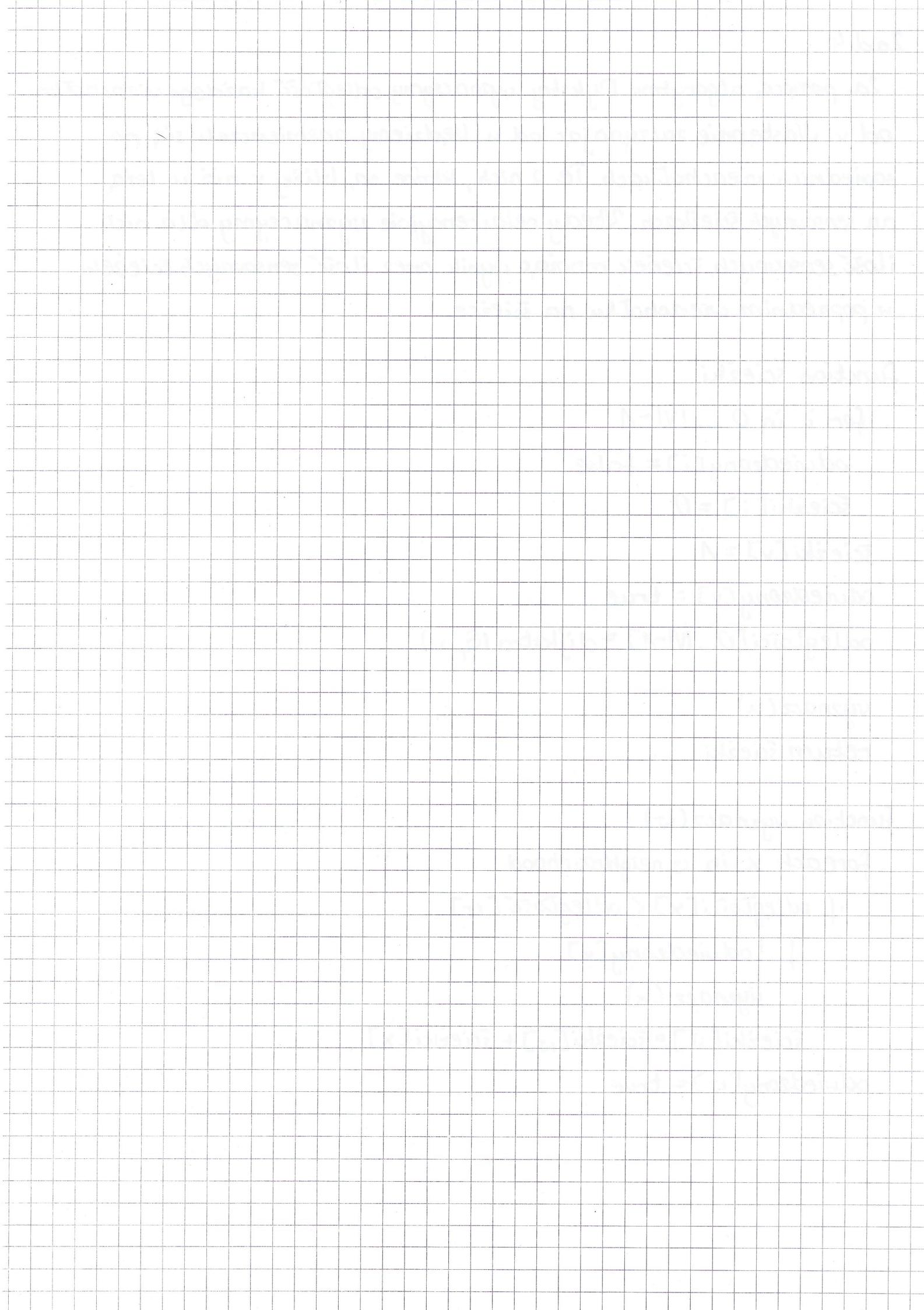
if odległość [ $x$ ] < odległość [ $w$ ]

if !odwiedzony [ $x$ ]

wyznacz ( $x$ )

ścieżki [ $w$ ] = ścieżki [ $w$ ] + ścieżki [ $x$ ]

odwiedzony [ $w$ ] = true



## Zad 6

Pomysł: podziel tablicę na dwie równe (lub różniące się o 1) części:

$L - \lceil \frac{n}{2} \rceil$  najmniejszych elementów

$R - \lfloor \frac{n}{2} \rfloor$  największych elementów.

Będziemy porównywać  $i$ -ty element z  $L$  z  $j$ -tym elementem z  $R$ , tak, że  $L[i] : R[j] \Rightarrow L[i] \leq R[j]$  i  $i$  oraz  $j$  są możliwe najmniejsze.

Po usunięciu takiej pary zwiększymy wynik o 2. Powtarzamy, aż dojdziemy do ostatniego elementu z  $L$  lub  $R$ .

procedure pary ( $T[1..n]$ )

$$L = T[1.. \lceil \frac{n}{2} \rceil]$$

$$R = T[\lceil \frac{n}{2} \rceil + 1..n]$$

$O(n)$

$$L = 1, r = \lceil \frac{n}{2} \rceil + 1$$

$$\text{wynik} = 0$$

while  $r \leq n$  and  $L \leq \lceil \frac{n}{2} \rceil$

    while  $R[r] \leq L[L]$  and  $L \leq \lceil \frac{n}{2} \rceil$

$L = L + 1$

        if  $L \leq \lceil \frac{n}{2} \rceil$

$$\text{wynik} = \text{wynik} + 2$$

$$r = r + 1$$

$$L = L + 1$$

return wynik

