Exercise 2. Danych jest n prostych $l_1, \ldots, l_n$ na płaszczyźnie ($l_i = a_i x + b_i$) takich, że żadne trzy proste nie przecinają się w jednym punkcie. Mówimy, że prosta l_i jest widoczna z punktu p, jeśli istenieje punkt q na prostej l_i taki, że odcinek $\overline{pq}$ nie ma wspólnych punktów z żadną inną prostą l_j poza punktami p i q.

Ułóż algorytm znajdujący wszystkie proste widoczne z punktu $(0, +\infty)$.

```
A[] <- lista gradientow
B[] <- lista wyrazow wolnych

sort(A, B) <- sortowanie po A i przestawiamy od razu B

RET[] <- [0, 1]

for i in range(2, n):
    RET[] <- i
    j <- i
    while ptk_przec(j, j-1) < ptk_przec(j-1, j-2):
        RET.remove(j-1)
        j <- j-1
```

Lemat: Mamy prostą L, jeśli protsa l_0 jest widoczna, to każda inna prosta l_i o mniejszych gradiencie też jest widoczna.

Exercise 4. Dane jest drzewo binarne (możesz założyć dla prostoty, że jest to pełne drzewo binarne), którego każdy wierzchołek v_i skrywa pewną liczbę rzeczywistą x_i. Zakladamy, że wartości skrywane w wierzchołkach są różne. Mówimy, że wierzchołek v jest minimum lokalnym, jeśli wartość skrywana w nim jest mniejsza od wartości skrywanych w jego sąsiadach.

Ułóż algorytm znajdujący lokalne minimum odkrywając jak najmniej skrywanych wartości.

```
V[] <- lista wartosci
n <- ilosc wierzcholkow

def find_min (v, ojciec):
    # ojciec trzyma info czy ojciec byl wiekszy
    if ocjciec:
        if 2*v >= n:
            return v
        if V[2*v] > V[v] and V[2*v+1] > V[v]:
            return v
    else:
        if 2*v >= n:
            return -1 # bo doszlismy juz do lisci i nie znalezlismy po drodze

        pot = find_min(2*v)

        if V[2*v] < V[v]:
            if pot == -1:
                return find_min(2*v+1)

        if V[2*v+1] < V[v]:
            pot = find_min(2*v+1)
            if pot == -1:
                return find_min(2*v)

        if pot == -1:
            return find_min(2*v+1)
```

Exercise 5. Dane jest nieukorzenione drzewo z naturalnymi wagami na krawędziach oraz liczba naturalna C.

(a) Ułóż algorytm obliczający, ile jest par wierzchołków odległych od siebie o C.

(b) (Z) Jak w pinkcie (a), ale algorytm ma działać w czasie $O(n \log n)$.

Exercise 6. Macierz A rozmiaru $n \times n$ nazywamy macierzą Toreplitza, jeśli jej elementy spełniają równanie $A[i, j] = A[i - 1, j - 1]$ dla $2 \leq i, j \leq n$.

(a) Podaj reprezentację macierzy Toeplitza, pozwalającą dodawać dwie takie macierze w czasie $O(n)$

(b) Podaj algorytm, oparty na metodzie dziel i zwyciężaj, mnożenia macierzy Toeplitza przez wektor. Ile operacji arytmetycznych wymaga takie mnożenie?

($\alpha*$) Lista, gdzie pierwsze n to pierwszy wiersz, a kolejne $n - 1$ to pierwsze kolumna, dodajemy po prostu po kolei

($\alpha*$) Dla n podzielnego przez 2 zauważmy, że możemy macierz T podzielić na KURWA NIE CHCE