Zadanie 3. (1,5pkt)

Otoczką wypukłą zbioru P, punktów na płaszczyźnie, nazywamy najmniejszy wielokąt wypukły zawierający (w swoim wnętrzu lub na brzegu) wszystkie punkty z P. Naturalny, oparty na zasadzie dziel i zwyciężaj, algorytm znajdowania otoczki wypukłej dla zbioru P, dzieli P na dwa (prawie) równoliczne podzbiory (np. pionową prostą), znajduje rekurencyjnie otoczki wypukłe dla tych podzbiorów, a następnie scala te otoczki. Podaj algorytm wykonujący tę ostatnią fazę algorytmu, tj. algorytm scalania dwóch otoczek wypukłych.

```
// wylicza iloczyn wektorowy wektorów ab, bc
Product(a, b, c)
p1 = c - a
p2 = b – a
product = p1.x * p2.y – p1.y * p2.x
return product

// określa, czy wektor bc skręca w prawo względem wektora ab
IsTurnRight(a, b, c)
return Product(a, b, c) > 0

// określa, czy wektor bc skręca w lewo względem wektora ab
IsTurnLeft(a, b, c)
return Product(a, b, c) < 0

// łączy 2 otoczki wypukłe
MergeHulls(h1, h2)
// znajduje górne i dolne krawędzie łączące h1,h2
<LU, RU> = FindUpperEdges(h1, h2)
<LL, RL> = FindLowerEdges(h1, h2)
// usuwa te punkty, które wcześniej były częścią którejś z otoczek h1,h2, a teraz
// są wewnątrz otoczki h (czyli nie należą do niej)
h1 = RemoveInsideLeft(h1, LU, LL)
h2 = RemoveInsideRight(h2, RU, RL)
return h1 ∪ h2
```

```
FindUpperEdges(h1, h2) // szuka 2 wierzchołków, które od góry łączą otoczki
// wyznaczamy kandydatów na punkty łączące otoczki
LU = {p ∈ h1: p.x jest największe w h1 (p.y jest największe dla remisów}
RU = {p ∈ h2: p.x jest najmniejsze w h1 (p.y jest najw. dla remisów}
while true: // szukanie górnej krawędzi łączącej
        anyChange = false // czy zmienił się jakiś z kandydatów górnej krawędzi
        LU2 = nextH(LU) // „wyższy” z sąsiadów LU z lewej otoczki
        RU2 = nextH(RU) // „wyższy” z sąsiadów RU z prawej otoczki
        // „podnoszenie” prawego górnego kandydata
        if (IsTurnLeft(LU, RU, RU2)
                RU = RU2
                anyChange = true
        // „podnoszenie” lewego górnego kandydata
        if (IsTurnRight(RU, LU, LU2)
                LU = LU2
                anyChange = true
        // znaleźliśmy 2 punkty łączące otoczki z góry
        if (!anyChange) return <LU,RU>

FindLowerEdges(h1, h2) // szuka 2 wierzchołków, które od dołu łączą otoczki
// wyznaczamy kandydatów na punkty łączące otoczki
LL = {p ∈ h1: p.x jest największe w h1 (p.y jest najmniejsze dla remisów}
RL = {p ∈ h2: p.x jest najmniejsze w h1 (p.y jest najmn. dla remisów}
while true: // szukanie dolnej krawędzi łączącej
        anyChange = false // czy zmienił się jakiś z kandydatów dolnej krawędzi
        LL2 = nextL(LL) // „niższy” z sąsiadów LL z lewej otoczki
        RL2 = nextL(RL) // „niższy” z sąsiadów RL z prawej otoczki
        // „obniżanie” prawego dolnego kandydata
        if (IsTurnRight(LL, RL, RL2)
                RL = RL2
                anyChange = true
        // „obniżanie” lewego dolnego kandydata
        if (IsTurnLeft(RU, LU, LU2)
                LU = LU2
                anyChange = true
        // znaleźliśmy 2 punkty łączące otoczki z góry
        if (!anyChange) return <LL,RL>
```

```
// usuwa te punkty, które wcześniej były częścią otoczki h2, a teraz
// są wewnątrz otoczki h (czyli nie należą do niej)
RemoveInsideLeft(h2, RU, RL)
for-each p in h2:
        if (IsTurnLeft(RL, RU, p) // jest poza nową otoczką
                remove p from h2
return h2

// usuwa te punkty, które wcześniej były częścią otoczki h1, a teraz
// są wewnątrz otoczki h (czyli nie należą do niej)
RemoveInsideRight(h1, LU, LL)
for-each p in h1:
        if (IsTurnRight(RL, RU, p) // jest poza nową otoczką
                remove p from h1
return h1
```