Algorithme des demi-plans : calcul de l'enveloppe convexe d'un ensemble de sommets
Données : un tableau V de n sommets notés v_i avec i ∈ [0, n-1]
Hypothèse : V ne contient pas plus de 2 points alignés
Résultat : le résultat est un tableau qui contient, pour chaque sommet de l'enveloppe convexe, une paire de deux nombres: son indice (dans le tableau V) et l'indice (dans le tableau V) de son successeur sur l'enveloppe convexe dans le sens direct. Lorsqu'au moins trois points alignés sont rencontrés, l'algorithme s'arrête et retourne le tableau des indices des trois premiers points alignés rencontrés par l'algorithme.

Fonctions auxiliaires à écrire : les fonctions `tour` et `findFirst` sont des prérequis de cet algorithme.

La fonction `tour` prend, comme arguments, trois points a, b et c et donne comme résultat :
+1 lorsque a, b, c forment un tour gauche,
-1 lorsque a, b, c forment un tour droit et
0 lorsque a, b, c sont alignés.

La fonction `findFirst` ne sert qu'à rendre l'algo plus lisible (dans la boucle `faire…tant que`, ligne 12 et suivantes) et doit retourner l'indice du premier point de V différent de V[i] et V[j].

```
 1  //initialisations
 2        aop result = ∅                                    //aop pour a-rray o-f p-airs
 3        int n = taille(V)
 4        int current, previous, i, j, k, f
 5        bool mc                                            // mc pour m-ême c-ôté
 6
 7  //début
 8  pour les couples (i, j) de 0 à n – 1 avec i < j
 9        mc = true
10        f = findFirst(V, i, j)
11        previous = tour (V[i],V[j],V[f])
12        k = f + 1
13        faire
14              si (k ≠ i && k ≠ j)
15                    current = tour(V[i], V[j], V[k])
16                    si (current == 0)                     // exception points alignés
17                          exception ( "alignement", [V[i], V[j], V[k]])
18                    sinon si (current ≠ previous)
19                          mc = false
20                    fin si
21              fin si
22              previous = current
23        tant que (++k < n && mc)
24
25        si (k == n && previous == current)
26              si (current > 0) result.add(i, j)
27              si (current < 0) result.add(j, i)
28              si (current == 0) exception ( "alignement", [V[i], V[j], V[k]])
29        fin si
30  fin pour
31
32  //résultat
33  return result
```

Algorithme de Jarvis: calcul de l'enveloppe convexe d'un ensemble de sommets
Données : un ensemble V de n sommets notés v_i avec i ∈ [0, n-1]
Hypothèse : V ne contient pas plus de 2 points alignés
Résultat : Le résultat est le tableau des sommets de V qui forment son enveloppe convexe.
Lorsque trois points alignés sont rencontrés, l'algorithme s'arrête et retourne le tableau des indices de ces trois points.

Fonctions auxiliaires à écrire : les fonctions `minY`, `findNext` et `tour`
La fonction `minY` ne sert qu'à rendre l'algo plus lisible, elle retourne comme résultat le (ou l'un des) point(s) d'ordonnée (coordonnée y) minimum pour l'ensemble des sommets de V.

La fonction `findNext` permet de trouver, pour un point P donné, le point qui se situe le plus à droite de P. Cette fonction utilise la fonction `tour` pour comparer les candidats.

La fonction `tour` (commune aux trois algorithmes) prend, comme arguments, trois points a, b et c et donne comme résultat :
+1 lorsque a, b, c forment un tour gauche,
-1 lorsque a, b, c forment un tour droit et
0 lorsque a, b, c sont alignés.

```
1   //initialisations
2       min = minY(V)
3       result = [min]
4       current = undefined
5       previous = min
6
7   //début
8       faire
9           current = findNext(previous)
10          result.add(current)
11          previous = current
12      tant que (current ≠ min)
13
14  //résultat
15  retourner result
```

Algorithme de Graham: calcul de l'enveloppe convexe d'un ensemble de sommets
Données : un ensemble V de n sommets notés v_i avec i ∈ [0, n-1]
Hypothèse : V ne contient pas plus de 2 points alignés
Résultat : Le résultat est la pile des sommets de V qui forment son enveloppe convexe.
Lorsque trois points alignés sont rencontrés, l'algorithme s'arrête et retourne le tableau des indices de ces points.

Fonctions auxiliaires à écrire: les fonctions `minY` et `tri`
Le `tri` de cet algorithme donne comme résultat l'ordre angulaire des points de V par rapport à un centre donné. Cette fonction de `tri` sera abordée comme exercice de td.

La fonction `minY` ne sert qu'à rendre l'algo plus lisible, elle retourne comme résultat le (ou l'un des) point(s) d'ordonnée (coordonnée y) minimum de l'ensemble des sommets de V.

La fonction `tour` (commune aux trois algorithmes) prend, comme arguments, trois points a, b et c et donne comme résultat :
+1 lorsque a, b, c forment un tour gauche,
-1 lorsque a, b, c forment un tour droit et
0 lorsque a, b, c sont alignés.

```
1   //initialisations
2       min = minY(V)
3       candidates = [tri(V–{min},min),min]   // on trie sans le min, et on ajoute min en fin de liste
4       result = [min, candidates[0]]         // result est un tableau avec des fonctions de pile
5       n = taille(candidates)
6       m = taille(result)
7
8   //début de l'algo
9       pour k de 1 à n - 1
10          tant que ((t = tour(result[m-2], result[m-1], candidates[k]) <= 0 && m >= 2)
11              si (t == 0)
12                  exception ("alignement", [result[m-2], result[m-1], candidates[k]])
13              fin si
14              result.pop()
15              m = taille(result)
16          fin tant que
17          result.push(candidates[k])
18          m = taille(result)
19      fin pour
20
21  //résultat
22      retourner result                      // l'enveloppe convexe de V qui commence par min
23                                            // et termine par min
```