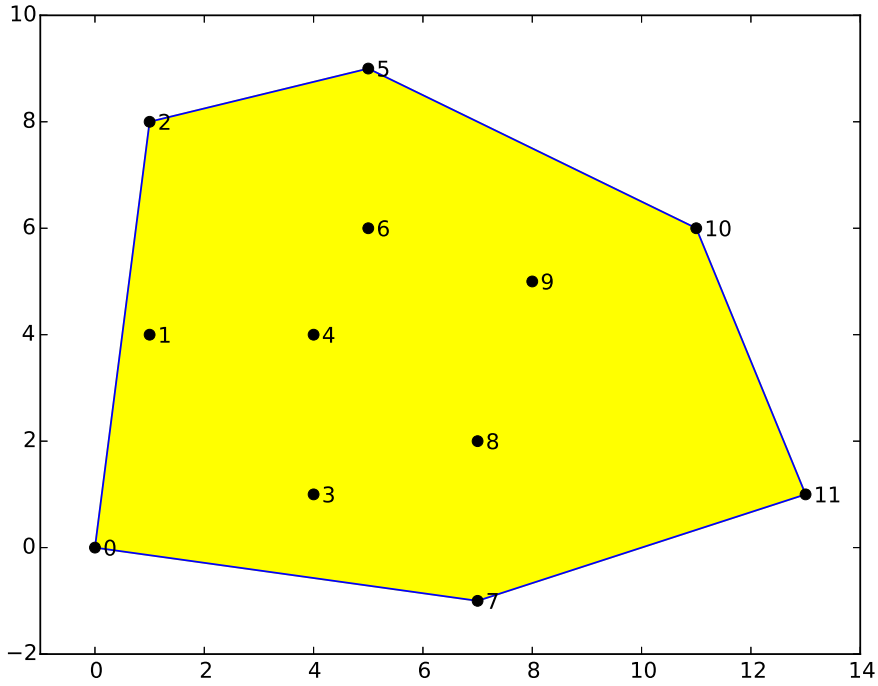
CORRIGÉ ÉPREUVE D'INFO CONCOURS BLANC

I. Algorithme de Jarvis

```python
# Corrigé Concours Blanc Info
# X-ENS MP-PC 2015 1ère partie

import numpy as np
import matplotlib.pyplot as plt

# Question 1

def plusBas(tab, n):
    jmin = 0
    ordonnee_min = tab[1][jmin]
    abscisse_min = tab[0][jmin]

    for j in range(1, n):
        if tab[1][j] < ordonnee_min or \
            (tab[1][j] == ordonnee_min and tab[0][j] < abscisse_min):
            jmin = j
            ordonnee_min = tab[1][jmin]
            abscisse_min = tab[0][jmin]
    return jmin

# Question 2
# Pour i=0, j=3 et k=4, le déterminant des vecteurs (pi pj, pi pk) vaut 12
# Ce triangle est direct
# Pour i=8, j=9, k=10, le déterminant vaut -8
# Le triangle est indirect

# Question 3

def orient(tab, i, j, k):
    # de façon très élémentaire, sans les fonctions numpy
    det = (tab[0][j] - tab[0][i]) * (tab[1][k] - tab[1][i])-\
            (tab[1][j] - tab[1][i]) * (tab[0][k] - tab[0][i])
    if det > 0:
        return 1
    elif det < 0:
        return -1
    else:
        return 0

# Question 5

def prochainPoint(tab, n, i):
    if i == 0:
        j = 1
    else:
        j = 0
    for k in range(n):
        if k != i and k != j and orient(tab, i, j, k)<0:
            # on a trouvé k tel que pj < pk
            j = k
    return j

# Question 6
# La fonction ci-dessus avec i=10  examine les points
```

```python
# pk à la recherche du plus grand élément du nuage privé de p10
# A chaque étape, l'indice j est celui du point réalisant ce maximum
# Pour k=1, le triangle (p10,p0,p1) est indirect donc j prend la valeur 1
# Pour k=2, le triangle (p10,p1,p2) est indirect donc j prend la valeur 2
# Pour k=3, le triangle (p10,p2,p3) est direct donc on garde j=2
# etc
# Pour k=5, le triangle (p10,p2,p5) est indirect donc j prend la valeur 5
# k ne change plus ensuite

# Question 7

def convJarvis(tab, n):
    premierPoint = plusBas(tab, n)
    L = [premierPoint]
    prochain = prochainPoint(tab, n, premierPoint)
    while prochain != premierPoint:
        L.append(prochain)
        prochain = prochainPoint(tab, n, prochain)
    return L

# Question 8
# La fonction plusBas est de complexité O(n) et n'est appelée qu'une fois
# La fonction prochainPoint est de complexité O(n) et est appelée m+1 fois
# La méthode append est un O(1) et est appelée m fois
# soit une complexité totale de O(n) + O(mn)+O(m) = O(mn)

# Exécution

tab = np.array([[0, 1, 1, 4, 4, 5, 5, 7, 7, 8, 11, 13],
                [0, 4, 8, 1, 4, 9, 6, -1, 2, 5, 6, 1]])
L = convJarvis(tab, tab.shape[1])
print(L)

# Tracé
L.append(L[0]) # pour fermer le polygone
plt.plot(tab[0, L], tab[1, L], color='blue')
plt.fill(tab[0, L], tab[1, L], color='yellow')
plt.plot(tab[0], tab[1], color='black', marker='o', linestyle = 'none')
for i in range(tab.shape[1]):
    plt.annotate(i, tab[:, i]+[0.15,-0.15])
    xmin = min(tab[0,:])
xmax = max(tab[0,:])
ymin = min(tab[1,:])
ymax = max(tab[1,:])
plt.xlim(xmin-1,xmax+1)
plt.ylim(ymin-1,ymax+1)
plt.show()

[7, 11, 10, 5, 2, 0]
```

Voici la figure obtenue :



II. Algorithme de Graham

```python
# Corrigé Concours Blanc Info
# X-ENS MP-PC 2015 2ème partie

import numpy as np
import matplotlib.pyplot as plt

def orient(tab, i, j, k):
    # de façon très élémentaire, sans les fonctions numpy
    det = (tab[0][j] - tab[0][i]) * (tab[1][k] - tab[1][i])-\
            (tab[1][j] - tab[1][i]) * (tab[0][k] - tab[0][i])
    if det > 0:
        return 1
    elif det < 0:
        return -1
    else:
        return 0

# Question 8bis
# Primitives sur les piles
# Les piles sont simulées à l'aide de listes

def newStack():
    return []

def isEmpty(s):
    return len(s) == 0

def push(i, s):
    s.append(i)

def top(s):
    return s[-1]
```

```
33
34  def pop(s):
35      return s.pop()
36
37  # Question 9
38  # On utilise le tri fusion vu en cours, adapté
39
40  def fusionner(a, b):
41      la = a.shape[1]
42      lb = b.shape[1]
43      if a[0][0] < b[0][0]:
44          c = a[:,0:1]
45          i,j = 1,0
46      else:
47          c = b[:,0:1]
48          i,j = 0,1
49      while i < la and j < lb:
50          if a[0][i] < b[0][j] :
51              c = np.concatenate((c,a[:, i:(i+1)]),axis=1)
52              i += 1
53          else :
54              c =np.concatenate((c,b[: ,j:(j+1)]),axis=1)
55              j += 1
56      # à ce stade, un (au moins) des deux tableau est fini, et on ajoute les éléments de l'autre à
57      if i == la :
58          c = np.concatenate((c,b[:, j:]),axis=1)
59      else :
60          c =np.concatenate((c,a[:, i:]),axis=1)
61      return c
62
63  def trifusion(a) :
64      n = a.shape[1]
65      if n == 1:
66          return a
67      m = n // 2
68      return fusionner( trifusion(a[:, :m]) , trifusion(a[:, m:]) )
69
70  # Question 10
71
72  def majES(tab, es, i):
73      p1 = pop(es)
74      if isEmpty(es):    # on n'a qu'un élément dans la pile
75          push(p1, es)   # on remet p1!
76          push(i, es)
77          return  # on s'arrête là
78
79      p2 = top(es)
80      while not isEmpty(es) and orient(tab, i, p1, p2) == -1:
81          p1 = pop(es)   # on dépile p1
82          if not isEmpty(es):
83              p2 = top(es)  # on met à jour p2
84
85      push(p1, es)  # que es soit vide ou que l'orientation soit bonne
86                    # il faut remettre p1
87      push(i, es)
88
89  # Question 11
90
91  def majEI(tab, ei, i):
92      # c'est le même principe mais on s'arrête lorsque l'orientation est négative
93      p1 = pop(ei)
```

```
    if isEmpty(ei):
        push(p1, ei)
        push(i, ei)
        return

    p2 = top(ei)
    while not isEmpty(ei) and orient(tab, i, p1, p2) == 1:
        p1 = pop(ei)
        if not isEmpty(ei):
            p2 = top(ei)

    push(p1, ei)
    push(i, ei)

# Question 12

def convGraham(tab, n):
    # on balaye le nuage et on met à jour les enveloppes convexes
    # sup et inf en même temps
    es = newStack()
    ei = newStack()
    push(0, es)
    push(0, ei)
    for i in range(1, n):
        majES(tab, es, i)
        majEI(tab, ei, i)

    pop(es)   # point en double
    while not isEmpty(es):
        push(pop(es), ei)
    pop(ei) # point en double aussi
    return ei

# Question 13
# Le tri est de complexité O(nlog(n))
#
# Dans ConvGraham, on fait n itérations et dans chaque fonction de mise à jour
# des piles on ne rajoute qu'une seule fois chaque  nouveau point
# et les points enlevés ne reviennent plus donc on a une complexité en O(n).
#
# soit au total une complexité O(nlog(n)) + O(n) = O(nlog(n)).


tab = np.array([[0, 1, 1, 4, 4, 5, 5, 7, 7, 8, 11, 13, 6, 10, 2, 0.5, 14, 8, 4.5, 14, -0.5],
                [0, 4, 8, 1, 4, 9, 6, -1, 2, 5, 6, 1, 10, -1.5, 0, -2, 0, 9, -3, 4, 6]])

tab=trifusion(tab)

L=convGraham(tab, tab.shape[1])

# Tracé
L.append(L[0])
plt.plot(tab[0, L], tab[1, L], color='blue')
plt.fill(tab[0, L], tab[1, L], color='yellow')
plt.plot(tab[0], tab[1], color='black', marker='o', linestyle = 'none')
for i in range(tab.shape[1]):
    plt.annotate(i, tab[:, i]+[0.15,-0.15])
xmin = min(tab[0,:])
xmax = max(tab[0,:])
ymin = min(tab[1,:])
ymax = max(tab[1,:])
```

```
155  plt.xlim(xmin-1,xmax+1)
156  plt.ylim(ymin-1,ymax+1)
157  plt.show()
```

Voici la figure obtenue :