► **Q1.**

```
def croissante(l):
    for i in range(len(l)-1):
        if l[i] > l[i+1]:
            return False
    return True
```

► **Q2.**

```
def maxi(l):
    ind = [0]
    M = l[0]
    for i in range(1, len(l)):
        if l[i] == M:
            ind.append(i)
        elif l[i] > M:
            M = l[i]
            ind = [i]
    return ind
```

► **Q3.**

```
def distance(l, i, j):
    if i > j:
        i, j = i, j
    d = 0
    for k in range(i, j):
        d += sqrt(1 + (l[k+1]-l[k])**2)
    return d
```

► **Q4.**

```
def pic(l, i):
    if i == 0:
        return l[i] > l[i+1]
    if i == len(l)-1:
        return l[i] > l[i-1]
    return l[i] > l[i-1] and l[i] > l[i+1]
```

► **Q5.**

• Une liste strictement croissante est un exemple où il n'y a aucun pic intérieur donc il n'y a pas nécessairement de pic intérieur.

• Tout d'abord, s'il y a un pic en $i \in [\![0, n-2]\!]$ alors la valeur suivante est strictement inférieure. Donc le nombre de pics est majoré par $\frac{n}{2}$ pour n pair et $\frac{n+1}{2}$ pour n impair.
En considérant une liste d'éléments $1, 0, 2, -1, 3, -2, \ldots$ on peut construire une suite pour laquelle ces valeurs sont réalisées.
Donc le nombre maximal de pics est $\frac{n}{2}$ pour n pair et $\frac{n+1}{2}$ pour n impair.

► **Q6.**

```
def pic_interieur(l):
    for i in range(1, len(l)-1):
        if pic(l, i):
            return True
    return False
```

► **Q7.**

```
def pics(l):
    p = []
    for i in range(len(l)):
        if pic(l,i):
            p.append(i)
    return p
```

► **Q8.**

```
def plus_grande_vallee(l):
    p = pics(l)
    # On ajoute maintenant à p les extrémités en tant que bornes
    # des vallées, si elles ne sont pas déjà des pics
    if p[0] != 0:
        p = [0] + p
    if p[len(p) - 1] != len(l) - 1:
        p.append(len(l) - 1)
    # Liste des tailles des vallées
    t = [p[i + 1] - p[i] + 1 for i in range(len(p) - 1)]

    imax = maxi(t)
    return t[imax]
```

► Q9.

```
1  def placementGloutonEnAvant(l, long, delta):
2      n = len(l)
3      poteaux = [0]       # premier poteau placé en x = 0
4      i, j = 0, 1
5      while i != n-1:
6          while j <= n-1 and estDeltaAuDessusDuSol(l,i,j,long,delta):
7              j += 1
8          i = j - 1
9          poteaux.append(i)
10     return poteaux
```

► Q10.

Supposons que les poteaux soient placés aux indices $p_0, p_1, \dots, p_k$ (donc $p_0 = 0$ et $p_k = n$).
La boucle liée au `while` de la ligne 5 a autant d'itérations que le nombre $k+1$ de poteaux placés.
Pour i fixé, compte tenu de la complexité de la fonction `estDeltaAuDessusDuSol`, la complexité de la boucle liée au `while` de la ligne 6 est :

$$\sum_{j=p_i}^{p_{i+1}} \mathrm{O}(j - p_i)$$

ce que l'on peut majorer par $\mathrm{O}((p_{i+1} - p_i)^2)$.
Donc la complexité de `placementGloutonEnAvant` est majorée par :

$$\sum_{i=0}^{k-1} \mathrm{O}((p_{i+1} - p_i)^2) \text{ donc } \mathrm{O}\Big(\sum_{i=0}^{k-1} (p_{i+1} - p_i)^2\Big)$$

ce que l'on peut ecore majorer par :

$$\mathrm{O}\Big(\Big(\sum_{i=0}^{k-1} (p_{i+1} - p_i)\Big)^2\Big)$$

c'est-à-dire :

$$\mathrm{O}\Big(\big(\underbrace{p_k - p_0}_{=n}\big)^2\Big) \text{ } i.e. \text{ } \mathrm{O}(n^2).$$

Donc la complexité est quadratique.

Autre explication (plus concise) –
Notons $m+1$ le nombre de poteaux. Au total j est itéré n fois, i est itéré m fois et le `append` apparaît m fois donc toutes ces opérations ont une complexité en $\mathrm{O}(n)$.
D'autre part, la fonction `estDeltaAuDessusDuSol` est appelée au plus $n+m$ fois donc la complexité totale des appels est en $\mathrm{O}(n^2)$.
Donc globalement, la complexité est en $\mathrm{O}(n^2)$.

► Q11.

```
def placementGloutonAuPlusLoin(l, long, delta)::
    n = len(l)
    poteaux = [0]
    i = 0  # abscisse du dernier poteau placé
    while i < n:
        j = n-1
        while not estDeltaAuDessusDuSol(l,i,j,long,delta):
            j -= 1
        i = j
        poteaux.append(i)
    return poteaux
```

Barème sur 27 : 2+3+2+3+2+2+2+3+3+3+2 (et 22/27 correspond à 18/20).
Moyenne et écart-type de la classe : 13,1 et 3,8
(aux CCINP2024 : 12,5 et 2,8 en informatique ; 13,8 et 2,5 en modélisation).