## I. Fonction impérative.

```
def h(L, t):
    return (L[t[0]] - L[t[1]])**2
def f(L):
    res = (0, 1)
    for i in range(len(L)):
        for j in range(i+1, len(L)):
            if h(L, res) > h(L, (i, j)):
                res = (i, j)
    return res
```

1. Que retourne
   - `f([10, 1000, 10000])`
   - `f([1, 3, 5, 7, 0, 13])`
2. Que fait la fonction?
3. Donnez en justifiant sa complexité au pire cas.

## II. Fonction récursive.

```
def f(u):
    if len(u) == 1:
        return u
    return f(u[:-1]) + f(u[1:])
```

1. Que retourne
   - `f("abc")`
   - `f("abcd")`
2. Donnez en justifiant sa complexité au pire cas.

```
def _g(L):
    if len(L) == 2:
        return (min(L), max(L))
    m1, m2 = _g(L[1:])
    if L[0] < m1:
        return (L[0], m2)
    if L[0] < m2:
        return (m1, L[0])
    return (m1, m2)

g = lambda L: _g(L)[1]
```

3. Que retourne
   - `g([42, 12, 23, 222])`
   - `g([22, 11, 443, 11, 12])`
4. Que fait la fonction?
5. Donnez en justifiant sa complexité au pire cas.

## IV. Programmation fonctionnelle.

En supposant que `count` est la fonction du module `itertools`.

```
def f(L, g):
    return min(zip(map(g, L), count()))[1]
```

1. Que retourne
   - `f([2, 3, 5], lambda x:-x)`
   - `f(["", "aa", "aaa", "v"], len)`
2. Que fait la fonction.

```
def g(L):
    if not type(L) == list:
        return type(L) == int
    return all(map(g, L))
```

3. Que retourne
   - `g([1, 2, 3, [], [3]])`
   - `g([['a', 1, 2], 3])`
4. Que se passe-t-il dans le bout de code suivant:

```
L = [1, 2, [4, [6, 7], 5], 3, 4]
L[2][1].append(L)
g(L)
```

5. Que se passe-t-il dans le bout de code suivant:

```
L = [1, 2]
K = [L, L]
g(K)
```

## V. Structure de données.

```
def f(L):
    S = set()
    K = []
    for x in L:
        if x not in S:
            K.append(x)
            S.add(x)
    return K
```

1. Que retourne:
   - `f(["chou", "garnie", "chou"])`
   - `f([12, 2, 12, 23, 4, 5, 4, 2])`
2. Que fait la fonction?
3. Donnez en justifiant sa complexité au pire cas.
4. Donnez en justifiant sa complexité en moyenne amortie.