# Testons nos connaissances

- Calculer la complexité de la fonction suivante :

```
def f(n):
    return n+1
```

- Calculer la complexité de la fonction suivante :

```
def f(n):
    for i in range(n):
        print(i)
```

- Calculer la complexité des fonctions suivantes :

```python
def f(n):
    for i in range(n):
        print(i)
    print("**********")
    for j in range(n,0,-1):
        print(j)
```

```python
def f(n):
    for i in range(n):
        print(i,":")
        for j in range(n,0,-1):
            print(j,end=" ")
        print("")
```

```python
def f(n):
    for i in range(n):
        print(i,":")
        for j in range(i,0,-1):
            print(j,end=" ")
        print("")
```

```python
def recherche_dichotomique(liste, element,debut,fin):
    nbIteration=0
    while debut <= fin:
        nbIteration=nbIteration+1
        milieu = (debut + fin) // 2
        valeur_milieu = liste[milieu]

        if valeur_milieu == element:
            return milieu  # L'élément a été trouvé, renvoie son index.
        elif valeur_milieu < element:
            debut = milieu + 1
        else:
            fin = milieu - 1
    print("Nombre d'itération",nbIteration)
    return -1  # L'élément n'est pas dans la liste.

# Exemple d'utilisation :
ma_liste = [1, 2, 3, 4,5,6,7,8,9,10,11,12,13,14,15,16]
element_recherche = -6

resultat = recherche_dichotomique(ma_liste, element_recherche,0,15)

if resultat != -1:
    print(f"L'élément {element_recherche} a été trouvé à l'index {resultat}.")
else:
    print(f"L'élément {element_recherche} n'a pas été trouvé dans la liste.")
```

1. Calculer le nombre d'itérations pour la boucle while, en considérant d'abord une liste de 4 éléments, puis, 16 éléments, et enfin 2^k élements.

2. Quelle serait la complexité de l'agorithme si on avait que des listes dont la longueur est une puissance de 2 ?

2. Généraliser.

- Calculer la complexité la complexité de la fonction suivante :

```
1 nbAppel=0
2 def fact(n):
3     global nbAppel
4     nbAppel=nbAppel+1
5     if (n==0):
6         return 1
7     return n*fact(n-1)
8 print(fact(10))
9 print(nbAppel)
```

- Donner la cascade des appels de la fonction Hanoi. En déduire la complexité de Hanoi :

```
def hanoi(de, vers, via, n) :
    if n == 1 :
        print(de, "vers", vers)
    else :
        hanoi(de, via, vers, n − 1)
        print(de, "vers", vers)
        hanoi(via, vers, de, n − 1)
```