Exercice 1 :

1. impératif

```
def dichotomie_imp(tab: int, x: int)->int:
    """
    version impérative
    Retourne la position de x ou -1 s'il n'est pas dans la
       liste
    """
    debut, fin = 0, len(tab) - 1
    while debut <= fin:
        milieu = (debut + fin)//2
        if tab[milieu] == x:
            return milieu
        elif tab[milieu] < x:
            debut = milieu+1
        else:
            fin = milieu-1
    return -1
```

2. récursif

```
def dichotomie_rec(tab: list, x: int, debut: int = 0, fin:
   int = -1)->int:
    """
    version récursive
    Retourne la position de x ou -1 s'il n'est pas dans la
       liste
    """
    #initialise 'fin'
    if fin == -1:
        fin = len(tab) - 1
    if debut <= fin:
        milieu = (debut + fin)//2
        if tab[milieu] == x:
            return milieu
        elif tab[milieu] < x:
            return dichotomie_rec(tab, x, milieu + 1, fin)
        else:
            return dichotomie_rec(tab, x, debut, milieu - 1)
    else:
        return -1
```

3. liste

```
l = [i for i in range(50)]
```

4. test

```
print(dichotomie_imp(l, 10))
print(dichotomie_rec(l, 10))
print(dichotomie_imp(l, 52))
print(dichotomie_rec(l, 52))
```


Christophe Viroulaud

5. L'algorithme de dichotomie est de complexité $O(log_2(n))$.

Exercice 2 :

```python
from random import randint

def tri_rapide(tab: list)->list:
    if not tab:
        return []
    else:
        pivot = tab[0]
        petit = [x for x in tab if x < pivot]
        grand = [x for x in tab[1:] if x >= pivot]
        return tri_rapide(petit) + [pivot] + tri_rapide(grand)

l = [randint(0,100) for _ in range(20)]
print(tri_rapide(l))
```

Ce tri n'est pas stable.

Exercice 3 :

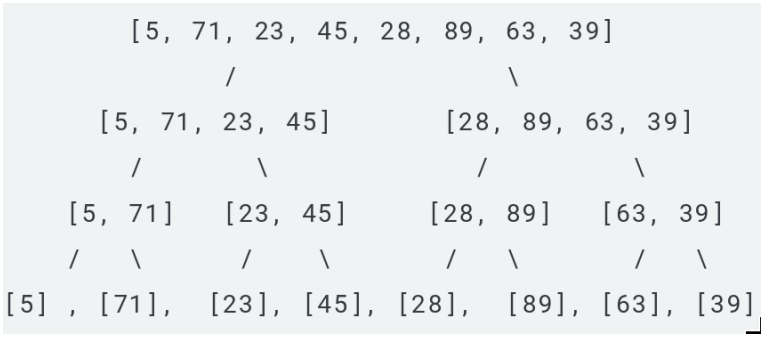

FIGURE 1 – Séparation

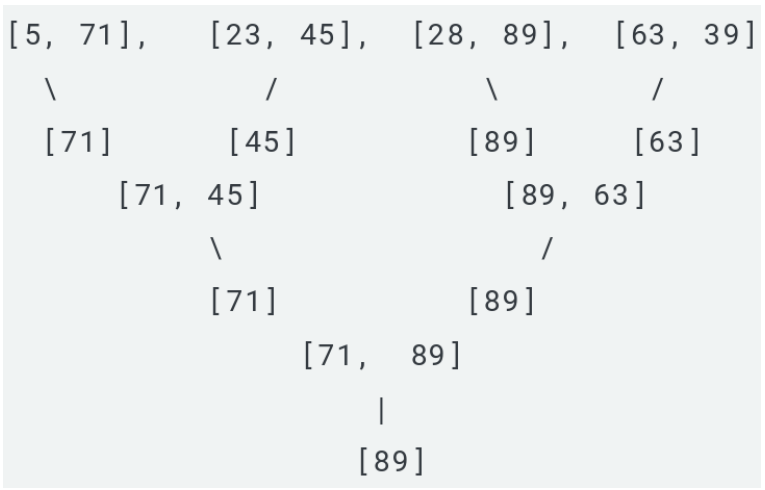

FIGURE 2 – Recombinaison

Cette fonction renvoie le maximum de la liste.

Christophe Viroulaud