

Exercice 1 :

1. impératif

```
1 def dichotomie_imp(tab: int, x: int)->int:
2     """
3     version impérative
4     Retourne la position de x ou -1 s'il n'est pas dans la
       liste
5     """
6     debut, fin = 0, len(tab) - 1
7     while debut <= fin:
8         milieu = (debut + fin)//2
9         if tab[milieu] == x:
10            return milieu
11        elif tab[milieu] < x:
12            debut = milieu+1
13        else:
14            fin = milieu-1
15    return -1
```

2. récursif

```
1 def dichotomie_rec(tab: list, x: int, debut: int = 0, fin:
   int = -1)->int:
2     """
3     version récursive
4     Retourne la position de x ou -1 s'il n'est pas dans la
       liste
5     """
6     #initialise 'fin'
7     if fin == -1:
8         fin = len(tab) - 1
9     if debut <= fin:
10        milieu = (debut + fin)//2
11        if tab[milieu] == x:
12            return milieu
13        elif tab[milieu] < x:
14            return dichotomie_rec(tab, x, milieu + 1, fin)
15        else:
16            return dichotomie_rec(tab, x, debut, milieu - 1)
17    else:
18        return -1
```

3. liste

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

4. test

```
1 print(dichotomie_imp(l, 10))
2 print(dichotomie_rec(l, 10))
3 print(dichotomie_imp(l, 52))
4 print(dichotomie_rec(l, 52))
```

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

Exercice 2 :

```

1 from random import randint
2
3 def tri_rapide(tab: list)->list:
4     if not tab:
5         return []
6     else:
7         pivot = tab[0]
8         petit = [x for x in tab if x < pivot]
9         grand = [x for x in tab[1:] if x >= pivot]
10        return tri_rapide(petit) + [pivot] + tri_rapide(grand)
11
12 l = [randint(0,100) for _ in range(20)]
13 print(tri_rapide(l))

```

Ce tri n'est pas stable.

Exercice 3 :

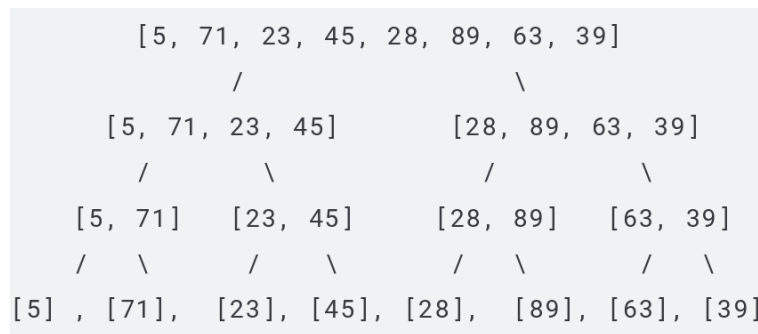


FIGURE 1 – Séparation

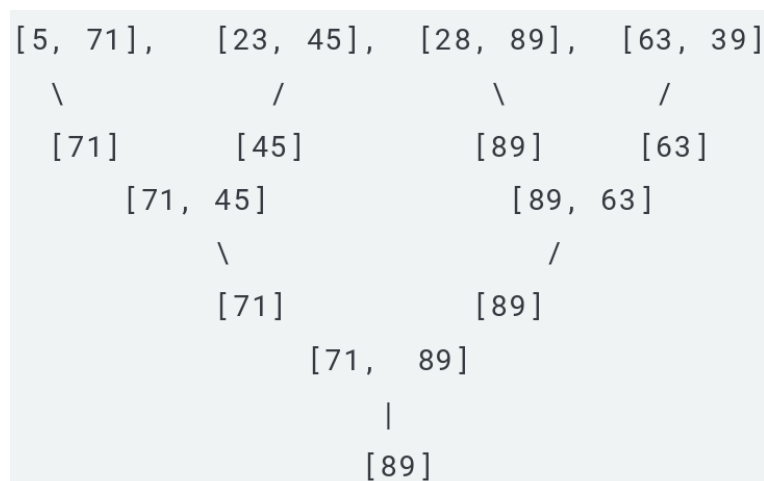


FIGURE 2 – Recombinaison

Cette fonction renvoie le maximum de la liste.