

Exercices Diviser pour régner

Christophe Viroulaud

Terminale - NSI

Algo 02

1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

5. Exercice 5

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercise 1

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

```
1 def inserer(tab: list, j: int) -> None:
2     if j >= 0 and tab[j] > tab[j+1]:
3         tab[j], tab[j+1] = tab[j+1], tab[j]
4         inserer(tab, j-1)
5
6
7 def tri_insertion_rec(tab: list) -> None:
8     for i in range(len(tab)):
9         inserer(tab, i-1)
```

1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

5. Exercice 5

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 2

La fonction **insérer** fait *redescendre* l'élément de rang j en le comparant avec celui de rang $j-1$. Cette propagation **de proche en proche** s'arrête quand les deux éléments sont égaux. L'ordre relatif est préservé.

```
1 t = [(1, 5), (3, 4), (1, 1), (2, 9), (1, 2)]
2 t = [(1, 5), (1, 1), (3, 4), (2, 9), (1, 2)]
3 t = [(1, 5), (1, 1), (2, 9), (3, 4), (1, 2)]
4 t = [(1, 5), (1, 1), (1, 2), (2, 9), (3, 4)]
```

Code 1 – Exemple d'exécution du tri insertion

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Dans le tri fusion, les éléments sont triés en se déplaçant également de proche en proche. L'ordre relatif est encore préservé.

```
1 def tri_selection(tab: list) -> None:
2     for i in range(len(tab)):
3         i_mini = i
4         for j in range(i+1, len(tab)):
5             if tab[j] < tab[i_mini]:
6                 i_mini = j
7         tab[i], tab[i_mini] = tab[i_mini], tab[i]
```

Code 2 – tri par sélection

Dans un tableau de taille n , le tri par sélection échange l'élément de rang j par le plus petit dans l'intervalle $]j; n[$. Certains éléments peuvent ainsi *sauter* par-dessus certains éléments égaux.

```
1 t = [(1, 5), (2, 4), (1, 1), (2, 9), (1, 2)]  
2 t = [(1, 5), (1, 1), (2, 4), (2, 9), (1, 2)]  
3 t = [(1, 5), (1, 1), (1, 2), (2, 9), (2, 4)]
```

Code 3 – Exemple de la non stabilité du tri par sélection

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

5. Exercice 5

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

```
1 def dichotomie_imp(tab: int, x: int) -> int:
2     debut, fin = 0, len(tab) - 1
3     while debut <= fin:
4         milieu = (debut + fin)//2
5         if tab[milieu] == x:
6             return milieu
7         elif tab[milieu] < x:
8             debut = milieu+1
9         else:
10             fin = milieu-1
11     return -1
```

Code 4 – Version impérative

```
1 def dichotomie_rec(tab: list, x: int, debut:
   int, fin: int) -> int:
2     if debut <= fin:
3         milieu = (debut + fin)//2
4         if tab[milieu] == x:
5             return milieu
6         elif tab[milieu] < x:
7             return dichotomie_rec(tab, x,
milieu + 1, fin)
8         else:
9             return dichotomie_rec(tab, x,
debut, milieu - 1)
10    else:
11        return -1
```

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Code 5 – Version récursive

1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

5. Exercice 5

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 4

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

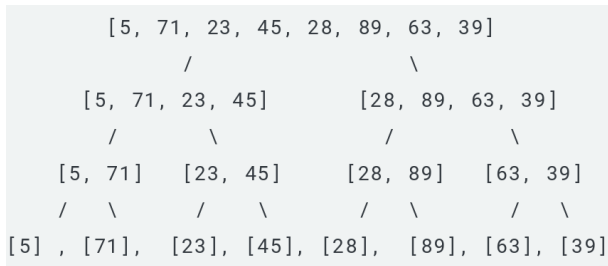


FIGURE 1 – Séparation

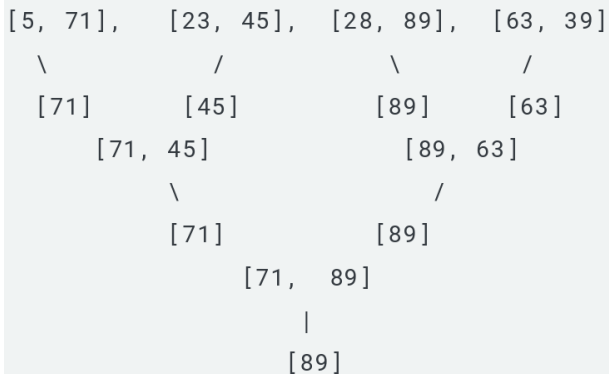


FIGURE 2 – Recombinaison

Cette fonction renvoie le maximum de la liste.

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

5. Exercice 5

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 5

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

```
1 def tri_rapide(tab: list) -> list:
2     if not tab:
3         return []
4     else:
5         pivot = tab[0]
6         petit = [x for x in tab if x < pivot]
7         grand = [x for x in tab[1:] if x >= pivot
8     ]
9         return tri_rapide(petit) + [pivot] +
10        tri_rapide(grand)
```


Exercice 5

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

```
1 def partitionner(tab: list, deb: int, fin: int)
  -> int:
2     pivot = tab[deb]
3     pos = deb
4     for i in range(deb+1, fin):
5         if tab[i] < pivot:
6             pos += 1
7             tab[i], tab[pos] = tab[pos], tab[i]
8     # place le pivot
9     tab[deb], tab[pos] = tab[pos], tab[deb]
10    return pos
```

Exercice 5

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

```
1 def tri_rapide(tab: list, deb: int, fin: int) ->  
  None:  
2     """  
3     Args:  
4         tab (list): tableau d'entiers  
5         deb (int): indice de début (inclus)  
6         fin (int): indice de fin (exclus)  
7     """  
8     if deb < fin:  
9         pivot = partitionner(tab, deb, fin)  
10        tri_rapide(tab, deb, pivot)  
11        tri_rapide(tab, pivot+1, fin)
```