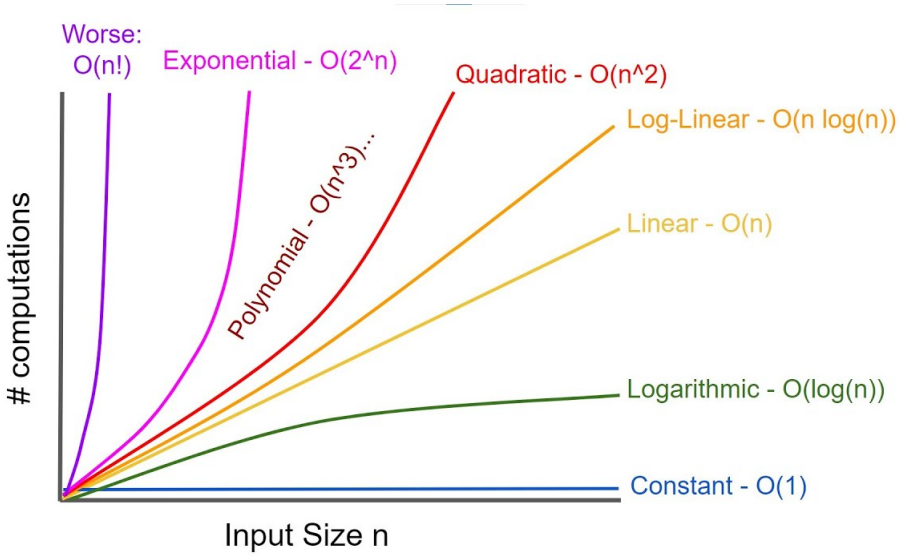
Corrigé – Diviser pour régner

Exercice 1



Complexité en temps :

- constant $O(1)$: calcul d'une ou plusieurs valeurs, **sans boucle** ;
- linéaire $O(n)$: parcours d'un tableau, **une boucle** ;
- quadratique $O(n^2)$: parcours d'un tableau à deux dimensions, **deux boucles imbriquées** ;
- logarithmique $O(log_2\ n)$: la recherche dichotomique ;
- quasi linéaire $O(n\ log_2\ n)$: le tri fusion.

Exercice 2

1. Ces algorithmes ont une complexité quadratique (en n^2).
2. Il est pertinent d'utiliser le tri fusion, car sa complexité est meilleure : la courbe la représentant se trouve en-dessous de la courbe de la fonction quadratique, pour une valeur de n donnée.

Exercice 3 – *(Les tris par insertion et par sélection : applications - révisions de 1ère -)*

1. Tri par sélection

   (a) `[9, 7, 1, 4, -2, 0, 8, 3]`

[9]	7	1	4	[-2]	0	8	3
-2	[7]	1	4	9	[0]	8	3
-2	0	[1]	4	9	7	8	3
-2	0	1	[4]	9	7	8	[3]
-2	0	1	3	[9]	7	8	[4]
-2	0	1	3	4	[7]	8	9
-2	0	1	3	4	7	[8]	9
-2	0	1	3	4	7	8	9

(b) `[2, 6, 7, -2, 0, -5, 1]`

[2]	6	7	-2	0	[-5]	1
-5	[6]	7	[-2]	0	2	1
-5	-2	[7]	6	[0]	2	1
-5	-2	0	[6]	7	2	[1]
-5	-2	0	1	[7]	[2]	6
-5	-2	0	1	2	[7]	[6]
-5	-2	0	1	2	6	7

2. Tri par insertion

(a) `[9, 7, 1, 4, -2, 0, 8, 3]`

9	[7]	1	4	-2	0	8	3
7	9	[1]	4	-2	0	8	3
1	7	9	[4]	-2	0	8	3
1	4	7	9	[-2]	0	8	3
-2	1	4	7	9	[0]	8	3
-2	0	1	4	7	9	[8]	3
-2	0	1	4	7	8	9	[3]
-2	0	1	3	4	7	8	9

(b) `[2, 6, 7, -2, 0, -5, 1]`

2	[6]	7	-2	0	-5	1
2	6	[7]	-2	0	-5	1
2	6	7	[-2]	0	-5	1
-2	2	6	7	[0]	-5	1
-2	0	2	6	7	[-5]	1
-5	-2	0	2	6	7	[1]
-5	-2	0	1	2	6	7

Exercice 4

1. Ce qui se **répète** pour le tri par sélection :
   - Parcourir un tableau pour prendre le minimum ;
   - Échanger ce minimum avec le premier élément.

   Ce qui **change** pour le tri par sélection :
   - L'indice du premier élément du sous-tableau qu'on considère (sous-tableau droit de plus en plus petit) ;
   - L'ordre des éléments du tableau.

   Ce qui se **répète** pour le tri par insertion :
   - Récupérer l'élément courant à partir du deuxième élément (ce qui laisse un « trou » pour mettre potentiellement un autre élément) ;
   - Comparer l'élément courant au précédent ;
   - Si plus petit que le précédent, on décale le précédent d'un cran vers la droite (dans le trou laissé, ce qui génère un autre trou) ;
   - Si plus grand, on insère le courant dans le trou auquel on en est.

Ce qui **change** pour le tri par insertion :

- L'indice de l'élément courant considéré ;
- L'ordre des éléments du tableau.

2. Pseudo-code (langage naturel) du tri par sélection (attention aux indices, on n'est pas en Python) :

```
fonction tri_selection(tableau t)
    n ← longueur(t)
    pour i variant de 0 à n - 2
        min ← i
        pour j de i + 1 à n - 1
            si t[j] < t[min], alors min ← j
        fin pour
        si min != i, alors échanger t[i] et t[min]
    fin pour
fin fonction
```

Pseudo-code (langage naturel) du tri par insertion :

```
 fonction tri_insertion(tableau T)
    n ← longueur(t)
    pour i de 1 à n - 1
        # mémoriser T[i] dans x
        x ← T[i]

        # décaler les éléments T[0]..T[i-1] qui sont plus grands que x, en partant de T
        j ← i
        tant que j > 0 et T[j - 1] > x
                    T[j] ← T[j - 1]
                    j ← j - 1

        # placer x dans le "trou" laissé par le décalage
        T[j] ← x
    fin pour
fin fonction
```

3. Ce qui donne en Python :

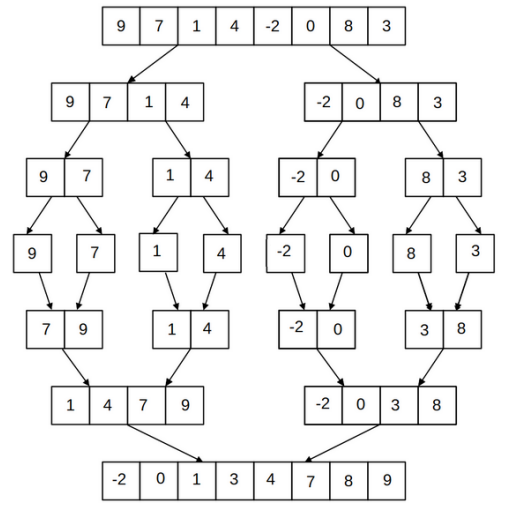
```python
def tri_par_selection(lst):
    for i in range(len(lst)):
        i_min = i
        for j in range(i+1, len(lst)):
            if lst[j] < lst[i_min]:
                i_min = j
        lst[i], lst[i_min] = lst[i_min], lst[i]

def tri_par_insertion(lst):
    for i in range(1,len(lst)):
        el_courant = lst[i]
        j = i
        while j > 0  and lst[j-1] > el_courant:
            lst[j] = lst[j-1]
            j = j-1
        lst[j] = el_courant
```

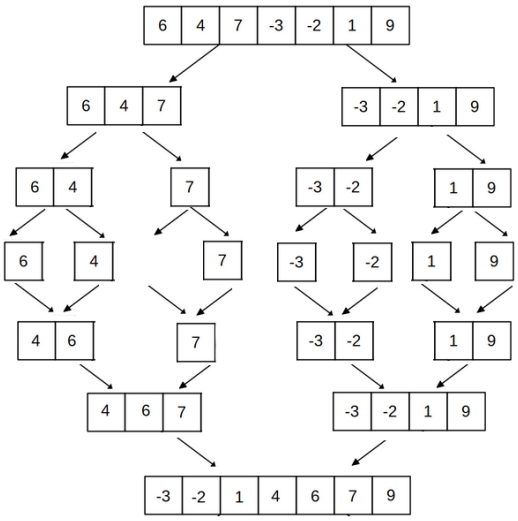
4. Pour tester le bon fonctionnement de notre programme précédent, il ne faut pas oublier que les modifications sont faites en place et que les fonctions renvoient `None` :

```
lst = [9, 7, 1, 4, -2, 0, 8, 3]
tri_par_selection(lst)
assert lst == [-2, 0, 1, 3, 4, 7, 8, 9]

lst = [2, 6, 7, -2, 0, -5, 1]
tri_par_selection(lst)
assert lst == [-5, -2, 0, 1, 2, 6, 7]

lst = [9, 7, 1, 4, -2, 0, 8, 3]
tri_par_insertion(lst)
assert lst == [-2, 0, 1, 3, 4, 7, 8, 9]

lst = [2, 6, 7, -2, 0, -5, 1]
tri_par_insertion(lst)
assert lst == [-5, -2, 0, 1, 2, 6, 7]
```

Exercice 5

1.





2. (a) La fusion :

```
def fusion_it(lst1, lst2):
    taille1 = len(lst1)
    taille2 = len(lst2)
    lst_trie = []
    i1 = 0
    i2 = 0
    while (i1 < taille1) and (i2 < taille2):
        if lst1[i1] < lst2[i2]:
            lst_trie.append(lst1[i1])
            i1 = i1 + 1
        else:
            lst_trie.append(lst2[i2])
            i2 = i2 + 1
    while i1 < taille1:
        lst_trie.append(lst1[i1])
        i1 = i1 + 1
    while i2 < taille2:
        lst_trie.append(lst2[i2])
        i2 = i2 + 1
    return lst_trie

def fusion_rec(lst1,lst2):
    # premier cas de base, pour une valeur triviale de lst1
    if len(lst1) == 0:
        return lst2
    # deuxième cas de base, pour une valeur triviale de lst2
    elif len(lst2) == 0:
        return lst1
    elif lst1[0] < lst2[0]:
        # on place le bon élément en premier
        return [lst1[0]] + fusion_rec(lst1[1:], lst2)
    else:
        return [lst2[0]] + fusion_rec(lst1, lst2[1:])
```

(b) Le tri :

```
def tri_fusion(lst):
    if len(lst) <= 1:    # cas de base
        return lst
    else:
        # on coupe le tableau en deux sous-tableaux
        lst1 = [lst[i] for i in range(len(lst)//2)]
        lst2 = [lst[i] for i in range(len(lst)//2, len(lst))]

    return fusion_rec(tri_fusion(lst1), tri_fusion(lst2))
```