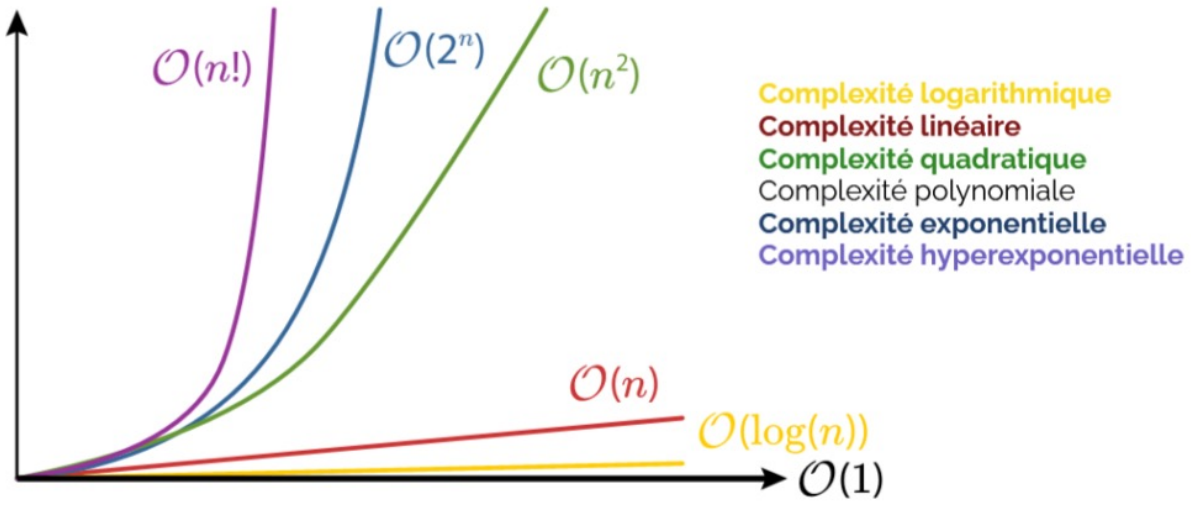
R1.01
DEV1

Algorithmes de tri et complexité



Au cours précédent

- Les tableaux (rappel)
- Les listes chaînées
- Les tableaux VS les listes chaînées
- Les piles et les files

Sommaire

- Pourquoi utiliser un tableau ?
- Pourquoi le trier ?
- Différentes méthodes de tri
- La complexité
- Algorithme de recherche

Pourquoi un tableau ?

- stocker un ensemble de données
  - même type (primitifs, enum, struct)
- traiter un ensemble de données
  - recherches
  - tris

Pourquoi trier un tableau ?

- Certaines applications nécessitent des données triées.
- Des algorithmes se basent souvent sur un tri.
- La recherche dans un tableau trié est plus rapide.

Trier un tableau

- Proposer un algorithme pour trier ce tableau par ordre croissant

0	1	2	3	4	5
14	78	54	23	5	66

0	1	2	3	4	5
5	14	23	54	66	78

Trier un tableau

- Opérations de bases
  - Comparaison des éléments
  - Permutation des éléments

- Algorithmes
  - Tri par sélection
  - Tri par insertion
  - Tri à bulles
  - Tri rapide
  - Tri …

Tri par sélection

- **Principe**

On recherche le plus petit élément du tableau et on le met en premier. Et ainsi de suite …

- **Algorithme** https://youtu.be/xWBP4lzkoyM

```
tri_selection(tableau t)
    n ← longueur(t)
    pour i de 0 à n - 1 - 1
        min ← i
        pour j de i + 1 à n - 1
            si t[j] < t[min]
                min ← j
        si min ≠ i
            échanger( t[i], t[min])
```

Tri par sélection

Tri par permutation (tri à bulles)

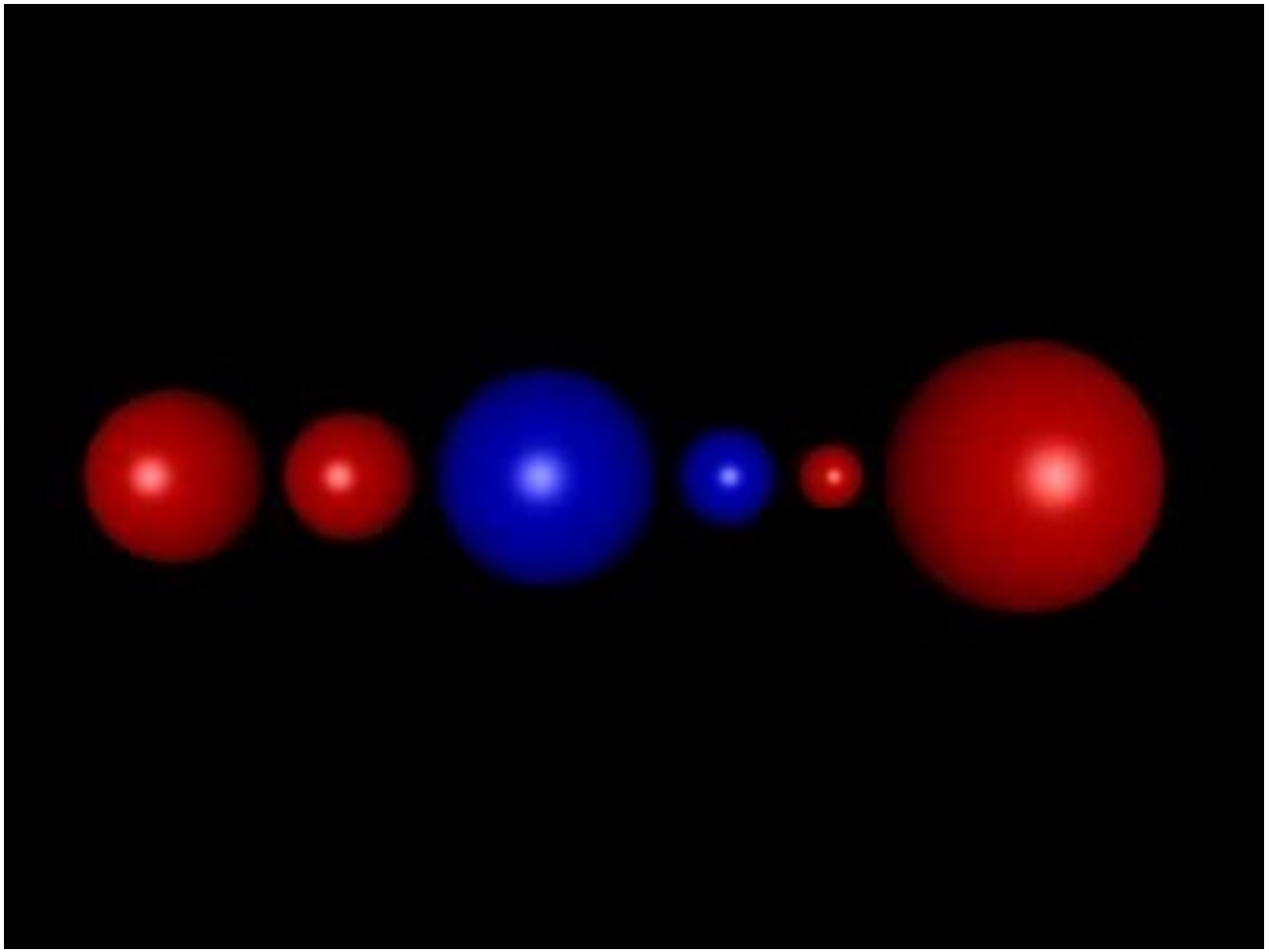
- **Principe**

On parcourt le tableau en comparant les éléments deux à deux et en les inversant si nécessaire. A la fin du premier parcours l'élément le plus grand est placé.

- **Algorithme** https://youtu.be/gWkvvsJHbwY

```
tri_à_bulles(tableau T)
    pour i allant de (taille de T)-1 à 1
        pour j allant de 0 à i-1
            si T[j+1] < T[j]
                échanger(T[j+1], T[j])
```

Tri par permutation (tri à bulles)



Tri par insertion

- **Principe**

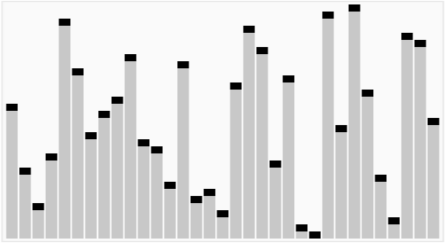
On divise le tableau en deux parties, une triée et une non triée. Ensuite, comme pour trier des cartes, on insère dans la partie triée un élément au bon endroit.

- **Algorithme** https://youtu.be/ROalU379l3U

```
tri_insertion(tableau T)
    n ← taille(T)
    pour i de 1 à n - 1
        // mémoriser T[i] dans x
        x ← T[i]
        // décaler vers la droite les éléments de T[0]..T[i-1]
        // qui sont plus grands que x en partant de T[i-1]
        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
```

Tri par insertion

Tri rapide (quick sort)



- **Principe**

On place un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite.

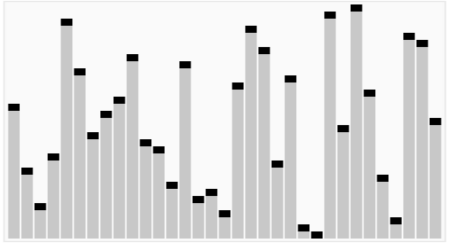
Cette opération s'appelle le partitionnement. Pour chacun des sous-tableaux, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.

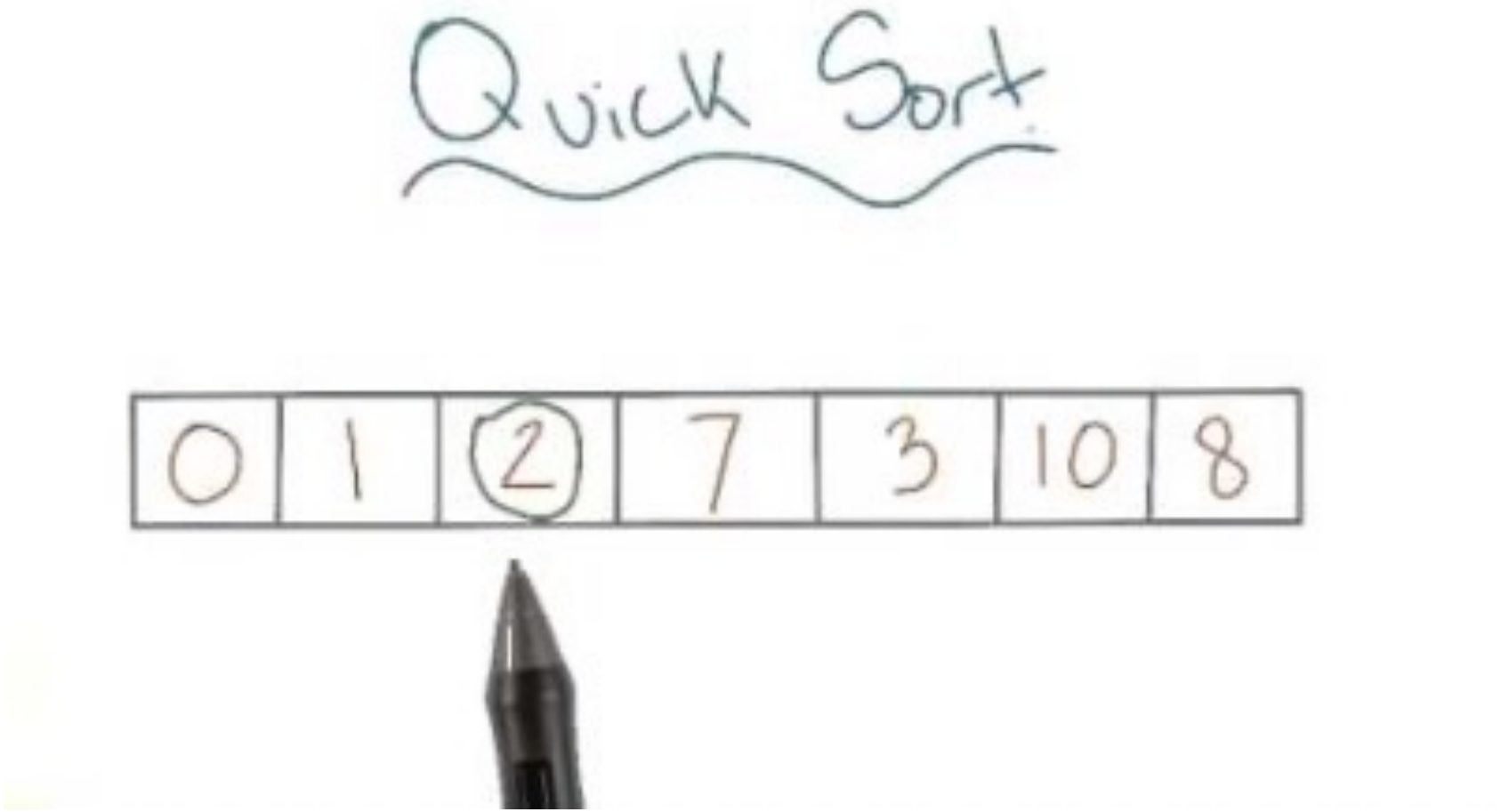
Concrètement, pour partitionner un sous-tableau :

- le pivot est placé à la fin (arbitrairement), en l'échangeant avec le dernier élément du sous-tableau,
- tous les éléments inférieurs au pivot sont placés en début du sous-tableau,
- le pivot est déplacé à la fin des éléments déplacés.

- **Algorithme** https://youtu.be/kUon6854jol

Tri rapide (quick sort)





Complexité des algorithmes

- performances d'un programme
  - temps d'exécution
  - place en mémoire

- complexité

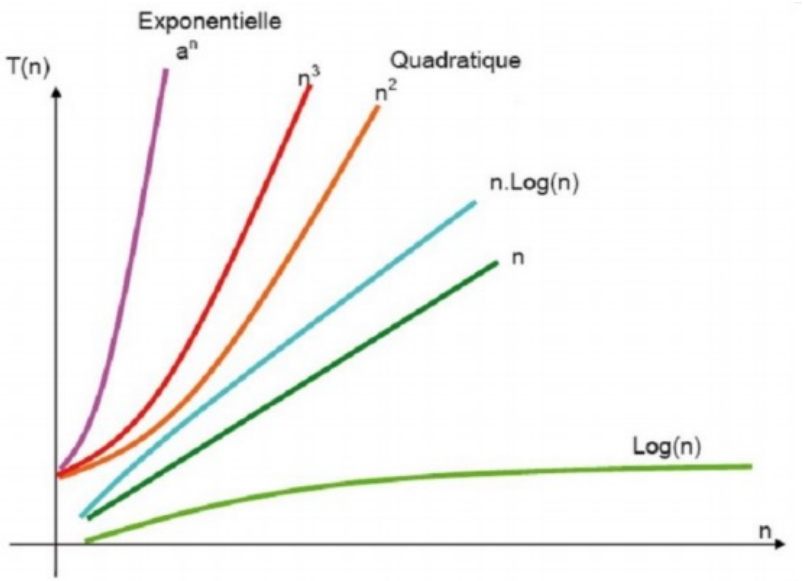
  évolution des performances en fonction d'éléments caractéristiques de l'algorithme

- efficacité

Complexité des algorithmes

- comparaison des algorithmes
  - nombre d'éléments à trier (n)
  - temps d'exécution (t)

    → **définition de l'ordre O**



1	constante
$\log_2 n$	logarithmique
n	linéaire
n^2	quadratique
n^3	cubique
a^n $\forall\ c > 1$	exponentielle

Algorithmes de tri

nb d'éléments	algorithme	en ordre	ordre aléatoire	ordre inverse
256	insertion	0.02	0.82	1.64
	bulle	1.26	2.04	2.80
	tri rapide	0.08	0.12	0.10
2048	insertion	0.22	50.74	103.80
	bulle	80.18	128.84	178.66
	tri rapide	0.72	1.22	0.76

https://www.toptal.com/developers/sorting-algorithms

algorithme	ordre
insertion -	$O(n^2)$
insertion =	**$O(n^2)$**
insertion +	$O(n)$
bulle -	$O(n^2)$
bulle =	**$O(n^2)$**
bulle +	$O(n)$
quick sort -	$O(n^2)$
quick sort =	**$O(n\log_2 n)$**
quick sort +	$O(n\log_2 n)$

Algorithmes de recherche

- Recherche linéaire
  - tableau trié
  - tableau non trié

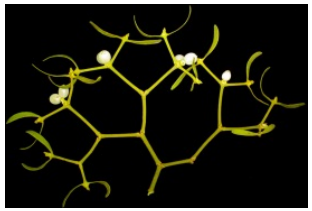
$O(n/2)$

- Recherche dichotomique
  - tableau trié

$O(\log_2 n)$

Ordre ?

Recherche dichotomique



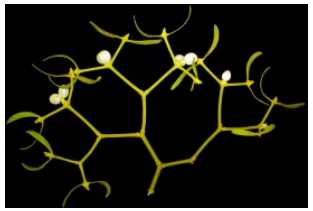
- **Principe**

On recherche dans un tableau trié. On commence par comparer l'élément avec la valeur de la case au milieu du tableau. Si les valeurs sont égales, la tâche est accomplie, sinon on recommence dans la moitié du tableau pertinente.

- **Algorithme** https://youtu.be/P3YID7liBug

```
1  debut = 0
2   fin = taille de tab
3   trouve = faux
4   tant que trouve == faux et debut <= fin
5       milieu = (debut + fin)/2
6       si (tab[milieu] == nombre)
7       alors trouve = vrai
8       sinon
9           si nombre > tab[milieu]
10          alors debut = milieu + 1
11          sinon
12              fin = milieu -1
13          fin si
14      fin si
15  retourner milieu
```

Recherche dichotomique



DS2 ex1

Ex1. Chiffrement de César et pas de salade ! **(10 pts/20min)**

Le chiffrement de César est une méthode de chiffrement par décalage très simple utilisée par Jules César dans ses correspondances secrètes. Le texte chiffré s'obtient en remplaçant chaque lettre du texte original par une lettre à distance fixe, toujours du même côté, dans l'ordre de l'alphabet.

- Exemples
  « ABC » codé avec un décalage de 3 donne : « DEF »
  « `Et 1 salade cezar, une !` » codé avec un décalage de 2 donne :
  « `Gv 1 ucncfg egbct, wpg !` »

1. Écrire la fonction *`offset`* qui prend en paramètre un caractère *`c`* et un entier *`n`*, renvoyant le caractère codé comme indiqué ci-dessus. Si le caractère n'est pas dans l'alphabet (par exemple l'espace ou les lettres accentuées), le caractère retourné sera celui passé en paramètre (voir les exemples ci-dessus).

2. Écrire une fonction *`decode`* qui prend en paramètre un entier *`n`* et une chaîne de caractères *`sentence`*, renvoyant la chaîne décodée avec un décalage de n lettres (on décale de n lettres dans l'autre sens).

3. Écrire la fonction *`main`* qui lit une phrase codée dans le fichier « `text2Decode.txt` », décode la phrase avec un décalage de 3 et écrit la phrase décodée dans le fichier « `textDecoded.txt` ».

SAE 1.02 The Game