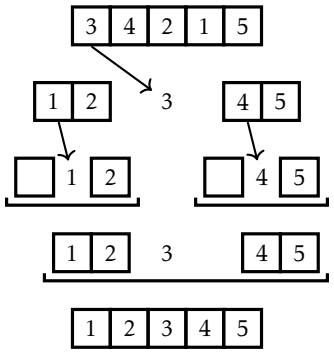
Algorithmes de tri



Objectifs

A la fin de la séquence d'enseignement les élèves doivent :

- pouvoir détailler les principales méthodes de tris
- pouvoir programmer les principales méthodes de tris

Table des matières

1 Trier ?

Le recours au tri est extrêmement fréquent. Pour le réaliser, il existe différentes méthodes qui n'ont pas toutes la même rapidité. Un moteur de recherche célèbre est capable de proposer, à partir de mots clés et en une fraction de seconde, un nombre hallucinant de pages triées suivant des critères qui lui sont propres.

Ce cours porte donc sur les principales méthodes de tris :

- tri par insertion
- tri par propagation (tri bulle)
- tri par sélection
- tri fusion
- tri rapide (Quick Sort)

Étant donné un ensemble d'éléments munis d'une relation d'ordre, l'objectif est de classer les éléments suivant un des sens de la relation d'ordre.

2 Stratégies par parcours des données

2.1 Tri par insertion

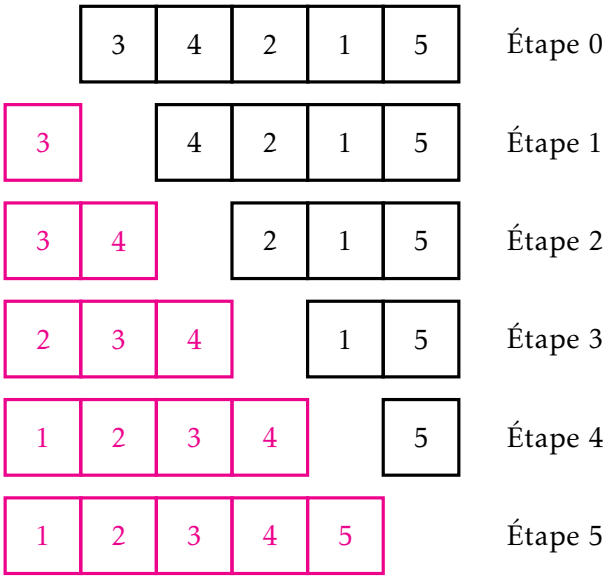
2.1.1 Principe

Principe :

Étant donné un ensemble d'éléments ordonnables, le tri par insertion :

- *sélectionne le premier élément de l'ensemble et le place dans une nouvelle structure.*
- *sélectionne le deuxième élément, le compare au premier élément sélectionné puis le place avant ou après la structure.*
- *le ième élément étant sélectionné, on parcourt la structure, triée, à (i-1) éléments, soit par ordre croissant, soit par ordre décroissant pour le ranger à sa place.*

2.1.2 Mise en images



2.1.3 Algorithme

Algorithm 1 Tri insertion

entrée: T un tableau de valeurs ordonnables
résultat: T trié par ordre croissant
tri_insertion(T):

```
1: pour i de 0 à taille(T) -1 faire
2:     x ← T[i]
3:     j ← i
4:     tant que j > 0 et T[j-1] > x faire
5:         T[j] ← T[j-1]
6:         j ← j - 1
7:     fin tant que
8:     T[j] ← x
9: fin pour
   renvoi: T
```

2.2 Tri par propagation (tri bulle)

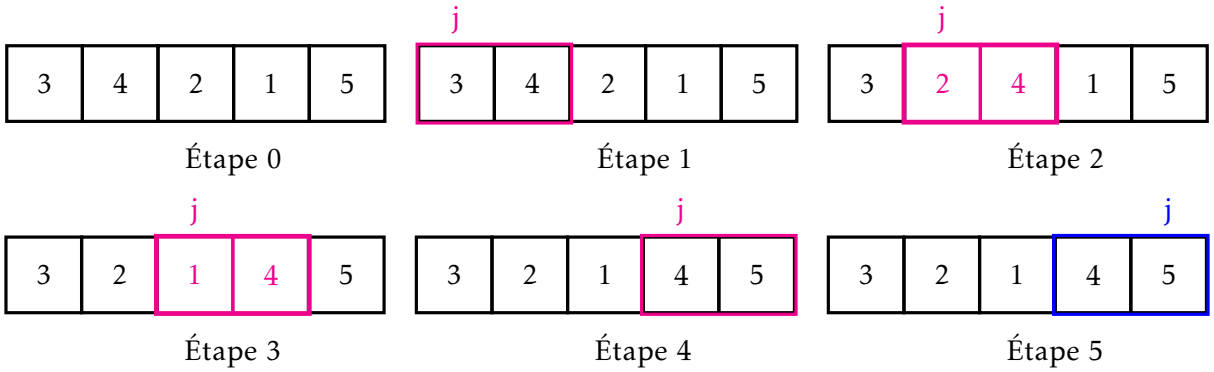
2.2.1 Principe

Principe :

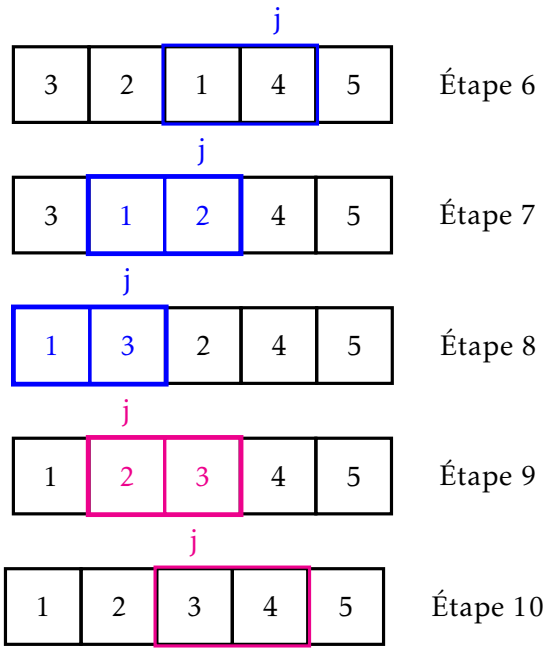
Étant donné un tableau `T` de valeurs ordonnables, le tri bulle consiste :

- *à parcourir le tableau dans le sens des indices croissants et pour chaque indice j, si la valeur d'indice $j+1$ est plus petite que la valeur d'indice j, les deux valeurs sont permutées*
- *à parcourir le tableau dans le sens des indices décroissants et pour chaque indice j, si la valeur d'indice $j-1$ est plus grande que la valeur d'indice j, les deux valeurs sont permutées*
- *à limiter l'étude du tableau au tableau `T[i-1:-i]` pour le ième parcours.*
- *à poursuivre jusqu'au statu quo ou quand la taille du tableau à étudier est inférieure à 2*

2.2.2 Mise en images



2.2.3 Algorithme



Algorithm 2 Tri par propagation (tri bulle)

```
    entrée: T un tableau de valeurs ordonnables
    résultat: T trié par ordre croissant
    tri_bulle(T):
 1: imin, imax ← 0, taille(T) - 1
 2: tant que imin < imax faire
 3:     pour i de imin à imax-1 faire
 4:         si T[i] > T[i+1] alors
 5:             T[i], T[i+1] ← T[i+1], T[i]
 6:         fin si
 7:     fin pour
 8:     imax ← imax -1
 9:     pour i de imax à imin + 1 avec un pas de -1 faire
10:         si T[i] < T[i-1] alors
11:             T[i], T[i-1] ← T[i-1], T[i]
12:         fin si
13:     fin pour
14:     imin ← imin + 1
15: fin tant que
16: renvoi: T
```

2.3 Tri par sélection

2.3.1 Principe

Principe :

Étant donné un tableau `T` de valeurs ordonnables, le tri par sélection consiste :

- *à déterminer l'élément le plus petit du tableau et l'échanger de place avec le premier élément du tableau*
- *à recommencer à l'itération `i+1` en omettant les `i` premières valeurs du tableau*

2.3.2 Mise en images

3	4	2	1	5	Étape 0
1	4	2	3	5	Étape 1
1	2	4	3	5	Étape 2
1	2	3	4	5	Étape 3

2.3.3 Algorithme

Algorithm 3 Tri sélection

entrée: `T` un tableau de valeurs ordonnables
résultat: `T` trié par ordre croissant
tri_selection(T):

```
1:  deb, n ← 0, taille(T)
2:  tant que deb < n-1 faire
3:      imin, vmin← deb, T[deb]
4:      pour j de deb+1 à n-1 faire
5:          si vmin > T[j] alors
6:              imin, vmin ← j, T[j]
7:          fin si
8:      fin pour
9:      T[imin], T[deb] ← T[deb], T[imin]
10:     deb ← deb+1
11: fin tant que
12: renvoi: T
```

3 Stratégies « diviser pour mieux régner »

3.1 Tri rapide (Quick Sort)

3.1.1 Principe

Principe :

Étant donné un tableau `T` de valeurs ordonnables, le tri rapide consiste :

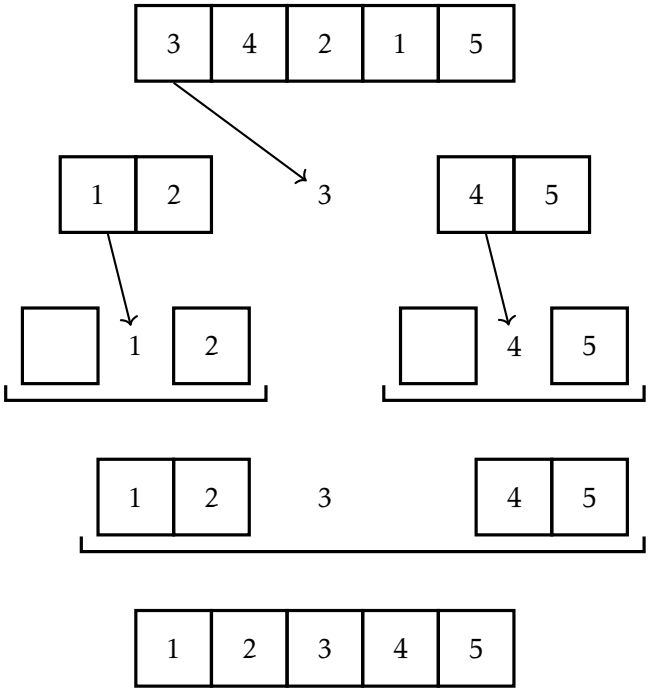
- *à prendre au hasard un élément du tableau qu'on appelle pivot*
- *à créer une liste avec toutes les valeurs de `T` plus petites que le pivot et une autre liste avec toutes les valeurs de `T` plus grande que le pivot*
- *à assembler à gauche les valeurs plus petites que le pivot triées par tri rapide, le pivot au centre et les valeurs plus grandes que le pivot triées par tri rapide à droite*
- *à retourner le tableau si celui-ci est vide ou n'a qu'un élément*

C'est l'algorithme de tri le plus utilisé et peut être même l'algorithme le plus utilisé dans le monde.

3.1.2 Mise en images

Par défaut, on choisit comme pivot le premier élément du tableau. Le premier pivot est donc 3.

3.1.3 Algorithme



Algorithm 4 Tri rapide

entrée: T un tableau de valeurs ordonnables
résultat: T trié par ordre croissant
tri_rapide(T):

```
si taille(T) < 2 alors
    renvoi: T
fin si
petits, grands ← [], []
pivot ← T.pop()
pour i de 1 à taille(T) faire
    x ← T.pop()
    si x ≤ pivot alors
        petits.append(x)
    sinon
        grands.append(x)
    fin si
fin pour
tri_rapide(petits)
tri_rapide(grands)
renvoi: petits + [pivot] + grands
```

3.2 Tri fusion

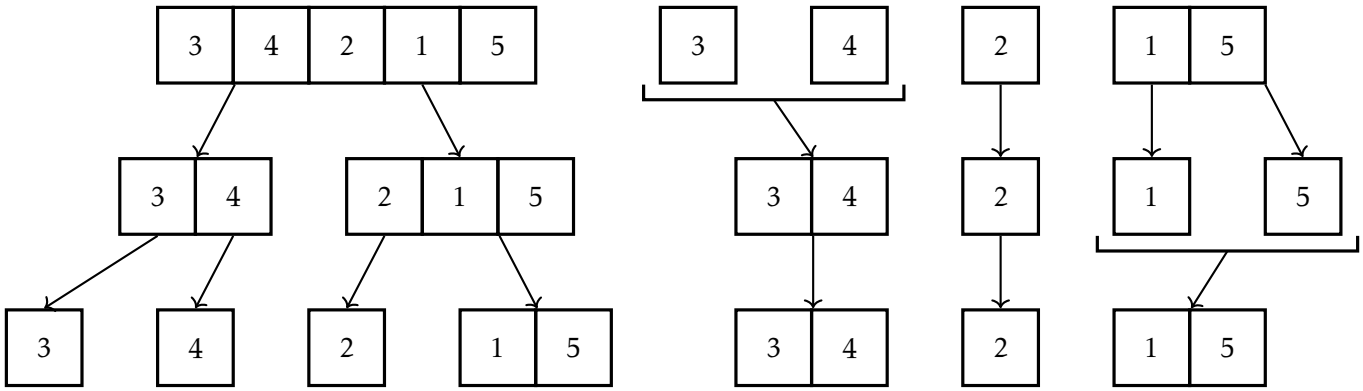
3.2.1 Principe

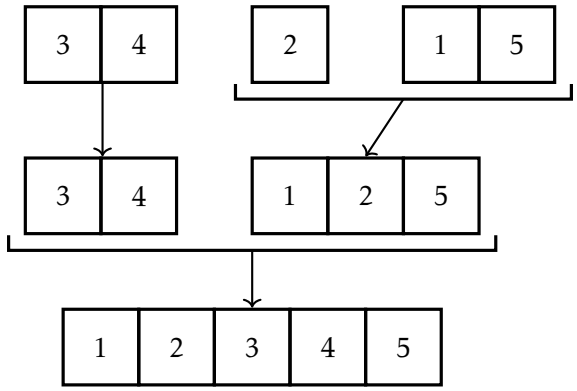
Principe :

Étant donné un tableau T de valeurs ordonnables, le tri fusion consiste :

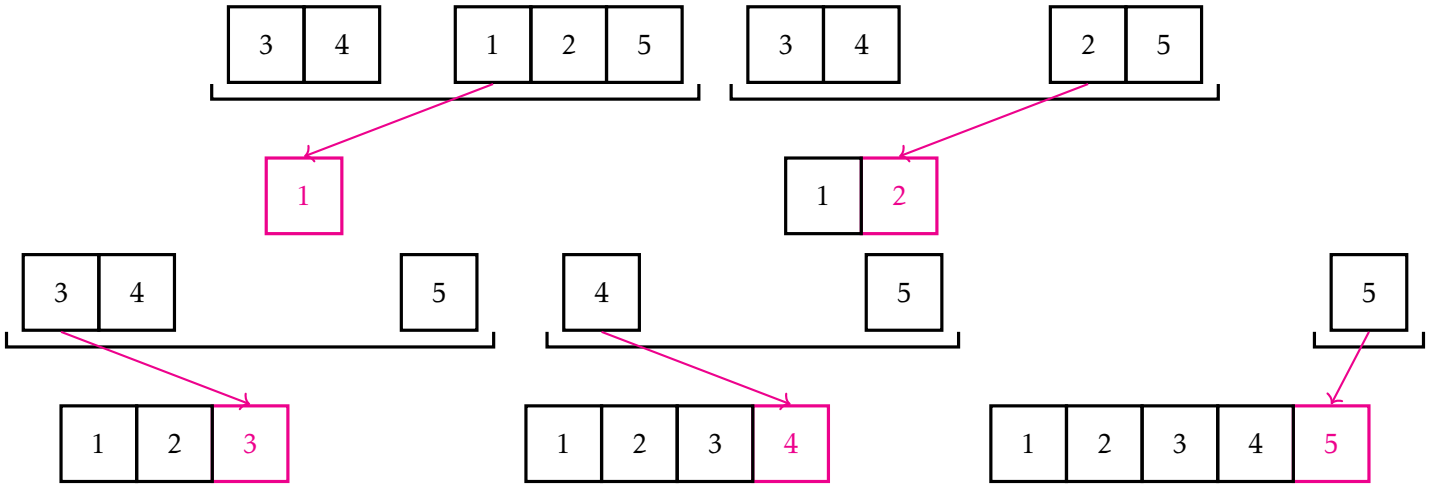
- *à couper le tableau en deux parts égales*
- *à trier par tri fusion chacune des deux parties*
- *à fusionner les deux parties*
- *à retourner directement la partie si elle ne comprend qu'un élément*

3.2.2 Mise en images globale





3.2.3 Mise en images de la dernière fusion



3.2.4 Algorithmes

Algorithm 5 Tri fusion

entrée: T un tableau de valeurs ordonnables
résultat: T trié par ordre croissant
tri_fusion(T):

```
si taille(T) == 1 alors
    renvoi: T
fin si
mil ← taille(T) // 2
gauche ← tri_fusion(T[:mil])
droite ← tri_fusion(T[mil:])
renvoi: fusion(gauche, droite)
```

Algorithm 6 Fusion

entrée: P et Q deux tableaux valeurs triés
résultat: T la fusion triée des tableaux P et Q
fusion(P, Q):

```
T, pi, qi ← [], 0, 0
np, nq ← taille(P), taille(Q)
tant que np > pi et nq > qi faire
    si P[pi] < Q[qi] alors
        T.append(P[pi])
        pi ← pi + 1
    sinon
        T.append(Q[qi])
        qi ← qi + 1
    fin si
fin tant que
si np > pi alors
    T ← T + P[pi:]
sinon
    T ← T + Q[qi:]
fin si
renvoi: T
```