Algorithmes de Tris

1. **Tri par comptage**
2. **Tri par insertion**
3. **Tri par sélection**
4. **Tri à bulles**
5. **Tri rapide**

Pourquoi trier ?

- La recherche d'une donnée dans un ensemble trié est plus rapide
- Remplir un tableau en maintenant l'ordre des éléments n'est pas toujours facile
- Le nombre de données à trier peut être très important
  - nécessité d'avoir un algorithme de tri efficace

Tri par comptage

- Quand on a des nombres de 1 à p on peut les trier facilement en comptant le nombre d'occurrences de chaque nombre.

  (1) On crée un tableau de p valeurs

  (2) On met toutes ces valeurs à 0

  (3) On traverse T et on compte le nombre de fois ou T[i] est pris en incrémentant P[T[i]]

  (4) Ensuite on balaie le tableau P et on copie autant de fois une valeur qu'elle apparait dans P

Tri par comptage : exemple

Avant triage

T	1	27	3	1	3

Tableau de comptage

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
P	0	2	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Après triage

T	1	1	3	3	27

Tri par comptage

Procédure Tri-Comptage (T : tableau [1..n] d'entiers, n : entier)

```
début
  Entier i, j, k, min, max ;
  min ← T[1]; max ← T[1]
  pour i de 2 à n {
        si (T[i] < min) {
            min ← T[i]
        }
        si (T[i] > max) {
              Max ← T[i]
        }
  }
  pour k de 1 à (max-min+1) { /* Initialisation du tableau de comptage */
        P[k] ← 0
  }
  pour i de 1 à n {  /* Création du tableau de comptage */
        P[T[i]-min+1]  ← P[T[i]-min+1] + 1
  }
  i ← 1
  pour k de 1 à (max-min+1) { /* Création du tableau trié */
        pour j de 1 à P[k] {
              T[i] ← k + min -1
              i ← i +  1
        }
  }
Fin
```

Tri par comptage : estimation du coût

- **Coût de recherche du min et du max** :
  - On recherche le minimum et le maximum parmi n éléments : ***au plus 2*n comparaisons*** ;
- **Coût de l'initialisation du tableau de comptage** :
  - On parcourt un tableau de p éléments : ***au plus p initialisations*** ;
- **Coût de création du tableau de comptage** : ***au plus n opérations*** ;
- **Coût des copies des valeurs dans T** : ***au plus n opérations*** ;
- **Au final, la complexité de ce tri est en O(n+p).**

Tri par sélection du minimum

- Principe :
  - L'algorithme parcourt le tableau pour rechercher l'indice de la case qui contient l'élément minimum.
  - L'algorithme échange le contenu de cette case avec celui de la première case du tableau (si elle ne contient pas déjà le minimum).
  - L'algorithme répète ces opérations en éliminant à chaque fois du parcours la case qui a été examinée lors du parcours précédent.
  - L'algorithme s'arrête lorsque la partie de tableau sur laquelle il travaille ne contient plus qu'une seule case.

Tri par sélection : 1ère itération

Avant triage

6	3	4	2	3	1

min = 3 ; i_min = 2

6	3	4	2	3	1

min = 2 ; i_min = 4

6	3	4	2	3	1

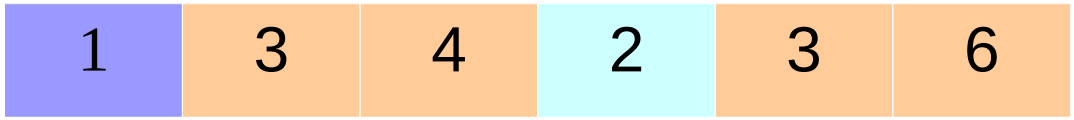
min = 1 ; i_min = 6

1	3	4	2	3	6

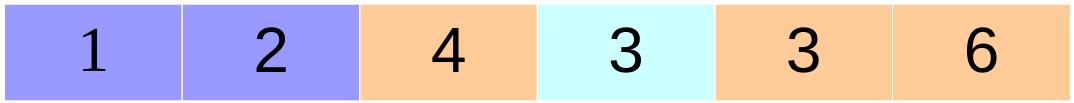
Échange entre la case d'indice 1 et la case d'indice i_min

Tri par sélection : exécution complète

1ère itération



min = 2 ; i_min = 4 → échange entre la case d'indice 2 et la case d'indice i_min



min = 3 ; i_min = 4 → échange entre la case d'indice 3 et la case d'indice i_min

min = 3 ; i_min = 5 → échange entre la case d'indice 4 et la case d'indice i_min

min = 4 ; i_min = 5 → pas d'échange

Tri par sélection

```
Procédure Tri-Selection (T:tableau [1..n] d'entiers, n:entier)
début
  Entier i, j, k, i_min, temp ;
  pour i de 1 à n {
     i_min ← i
     pour j de i+1 à n {
     /* recherche du minimum parmi les éléments d'indices i+1 à n */
              Si (T[j] < T[i_min]) {
                  i_min ← j
              }
      }
      Si (i <> i_min) { /* échange */
          temp ← T[i]
          T[i] ← T[i_min]
          T[i_min] ← temp
      }
  }
Fin
```

Tri par sélection : estimation du coût

- **Combien de comparaisons ?**

au cours d'une itération, toutes les cases sont comparées au minimum courant

  - ***(n-1) comparaisons*** à la première itération : on recherche le minimum parmi (n) éléments.
  - ***(n-2) comparaisons*** à la seconde itération : on recherche le minimum parmi (n -1) éléments.
  - …
  - Au total : n x (n-1)/2 comparaisons.

- **Combien d'affectations ?**

Le placement définitif d'un élément s'effectue avec un seul échange

  - au plus (n-1) échanges, 3 fois plus d'affectations

Pour n = 10 000 éléments, 45 000 000 comparaisons et 10 000 échanges.

Peut-on diminuer le nombre de parcours ?

- Pourquoi placer un seul élément à chaque parcours ?

- Idée d'amélioration : au fur et à mesure du parcours, on trie les éléments parcourus par ordre croissant
  - On tri d'abord les deux premiers éléments
  - On place le 3ème élément dans la liste formée par les deux premiers : la liste des 3 éléments doit être triée par ordre croissant
  - Etc

- On trie tout le tableau en le parcourant une seule fois

Tri par insertion

- Principe :
  - Lorsqu'on insère un élément à sa place dans une liste triée, on obtient une liste triée.
- L'objectif d'une étape est d'insérer le i-ème élément à sa place parmi ceux qui précèdent.
  - trouver où l'élément doit être inséré en le comparant aux autres,
  - décaler les éléments afin de pouvoir effectuer l'insertion
- Lors du traitement de la case d'indice i :
  - on a déjà rangé dans l'ordre le contenu des cases d'indice 0 à (i-1)
  - on insère le contenu de la case i dans cette liste triée
    - on décale le contenu des cases d'indice < i, dont la valeur est supérieure à T[i]
    - on a besoin d'une variable auxiliaire pour stocker temporairement le contenu de la case i

Traitement de la case i du tableau

- On décale le contenu des cases d'indice entre (i-1) et 0 dont le contenu est supérieur à aux :
  - on doit faire le parcours du tableau de (i-1) à 0

```
aux = T[i] ;
j = i-1 ;
tant que ((j >= 0) && (tab[j] > aux)) {
     T[j+1] = T[j] ;
     j--;
}
T[j+1] = aux ;
```

Tri par insertion : exemple

Avant triage

6	3	4	2	3	5

En triant les deux premiers éléments on obtient

3	6	4	2	3	5

En insérant le troisième élément à sa place dans la liste triée on obtient

3	4	6	2	3	5

En insérant le quatrième élément à sa place dans la liste triée on obtient

2	3	4	6	3	5

En insérant le cinquième élément à sa place dans la liste triée on obtient

2	3	3	4	6	5

En insérant le sixième élément à sa place dans la liste triée on obtient

2	3	3	4	5	6

Tri par insertion

```
Procédure Tri-Insertion (T:tableau [1..n] d'entiers, n : entier)
début
  Entier i, j, aux;

  pour i de 1 à n-1 {
          aux ← T[i]
          j ← i - 1
     Tant que (j ? 0 et T[j] > aux) {
     /* on décale les éléments pour l'insertion */
                  T[j+1] ← T[j]
                  j ← j - 1
     }
     T[j+1] ← aux /* on fait l'insertion proprement dite */
   }
Fin
```

Tri par insertion : estimation du coût

- **Coût du décalage :**
  - L'indice i varie de 1 à (n-1), soit **(n-1) valeurs différentes** ;
  - Pour chaque valeur de i, l'indice j prend **au plus** j valeurs différentes ;
  - Donc, la somme pour différentes valeurs de j est :
    - 1 + 2 + 3 + … + (n-2) + (n-1) = **n x (n-1)/2**

- **Coût des opérations insertion**:
  - L'indice j varie de 1 à (n-1): ***au plus 3(n-1) opérations*** .

- **Le nombre total d'opérations** : 3(n-1) + 4n x (n-1) /2

Tri à bulles

- Sur un tableau de n éléments (numérotés de 1 à n), le principe du tri à bulles est le suivant :
  - On parcourt le tableau à trier à l'envers, en comparant les éléments consécutifs deux-à-deux tout en faisant remonter vers le début du tableau les plus petits éléments.
  - La remontée est progressive : un élément remonte s'il est plus petit que son voisin de gauche.
  - Lorsque l'on a fini de parcourir le tableau une première fois, on est sûr d'avoir placé le plus petit élément à la bonne place.
  - On parcourt donc le nouveau tableau à l'envers pour placer le second plus petit élément, troisième plus petit, jusqu'au dernier.

Tri à bulles : exemple

Avant triage	2	3	1	4	0
	2	3	1	0	4
	2	3	0	1	4
	2	0	3	1	4
	0	2	3	1	4
	0	2	3	1	4
	0	2	1	3	4
	0	1	2	3	4
	0	1	2	3	4
	0	1	2	3	4
Après triage	0	1	2	3	4

Traitement de [N-1 à 0]

Traitement de [N-1 à 1]

Traitement de [N-1 à 2]

Tri à bulles

Procédure Tri-Bulles (T : tableau [1..n] d'entiers, n : entier)

début

```
 Entier i, k, temp;

  pour k de 0 à (n-2) {/* pour chaque passe */
        pour i de n-1 à k+1 {/* mettre le plus petit élément de la tranche T(k..n-1) en pos k */
              si (T[i]  < T[i-1]) {/* échange de T[i] et T[i-1] */
                   temp ← T[i]
                   T[i] ← T[i-1]
                    T[i-1] ← temp
                }
        }
   }
```

Fin

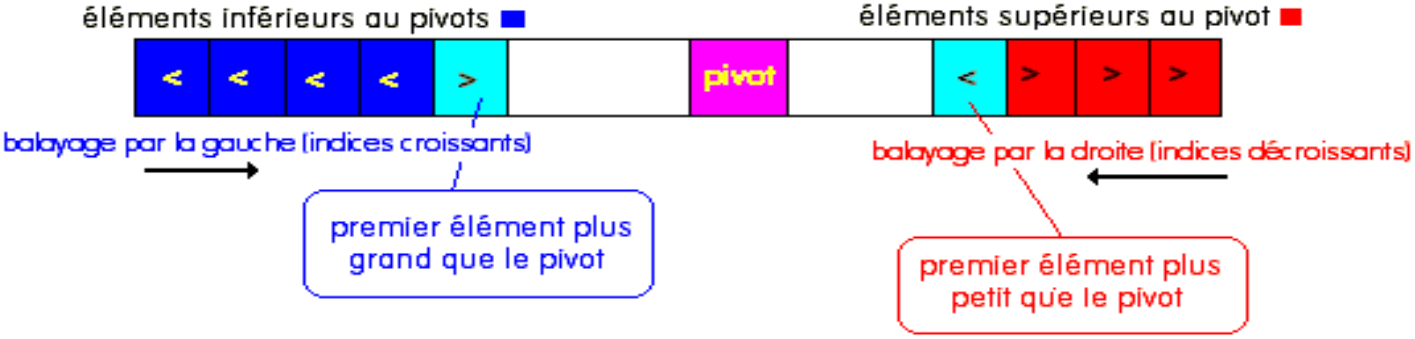
Tri par bulles : estimation du coût

- **Cas où le tableau n'est pas trié:**
  - Au maximum, ce tri effectue n(n-1)/2 comparaisons,
  - A chaque itération, ce tri effectue au plus (n-1) permutations.
  - D'une façon générale et dans le pire des cas, le tri à bulles a donc une complexité en $O(n^2)$.

- **Cas où le tableau est déjà trié**:
  - On obtient une complexité optimale de O(n).
  - Le tableau est parcouru une et une seule fois et aucune permutation n'est effectuée.

Tri rapide (Quicksort) (1/2)

- Sur un tableau L de n éléments, le principe du tri rapide est le suivant :
  - **Etape 1** : On choisit une valeur quelconque dans le tableau T (la dernière par exemple) que l'on dénomme **pivot**,
  - **Etape 2** :
    - (2.1) On construit la sous-liste L1 (à gauche) comprenant tous les éléments de L dont la valeur **est inférieure ou égale au pivot**,
    - (2.2) On construit la sous-liste L2 (droite) constituée de tous les éléments dont la valeur **est supérieure au pivot**.



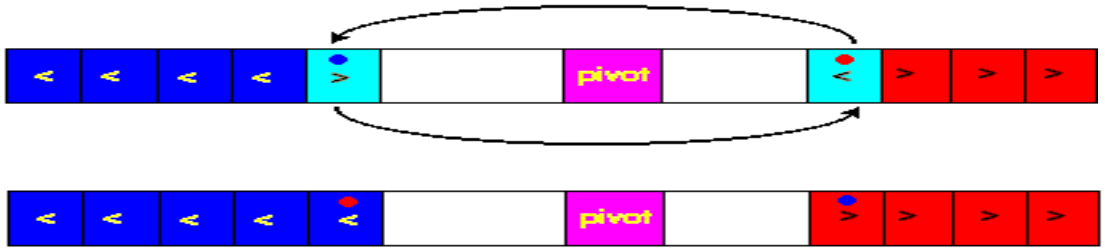
From http://discala.univ-tours.fr/LesChapitres.html/Cours4/

- **Balayage par la gauche** : on arrête ce balayage dès que l'on trouve un élément dont la valeur **est plus grande** que celle du pivot. Dans ce dernier cas cet élément n'est pas à sa place dans cette sous-partie mais plutôt dans l'autre sous-partie.

- **Balayage par la droite** : on arrête ce balayage dès que l'on trouve un élément dont la valeur **est plus petite** que celle du pivot. Dans ce dernier cas cet élément n'est pas à sa place dans cette sous-partie mais plutôt dans l'autre sous-partie.

Tri rapide (Quicksort) (2/2)

- Sur un tableau L de n éléments, le principe du tri rapide est le suivant :
  - **Etape 3** : On procède à l'échange des deux éléments mal placés dans chacune des sous-parties.



From http://discala.univ-tours.fr/LesChapitres.html/Cours4/

  - **Etape 4** : On continue le balayage par la gauche et le balayage par la droite tant que les éléments sont bien placés (valeur inférieure par la gauche et valeur supérieure par la droite), en échangeant à chaque fois les éléments mal placés.
  - **Etape 5** : La construction des deux sous-listes est terminée dès que l'on atteint (ou dépasse) le pivot.
  - ***Il reste à recommencer les mêmes opérations sur les parties L1 et L2 jusqu'à ce que les partitions ne contiennent plus qu'un seul élément.***

Tri rapide : Exemple

- L = [4, 23, 3, 42, 2, 14, 45, 18, 38, 16], pivot = 16
  - **Balayage à gauche :**
    - 4 < 16 => il est dans la bonne sous-liste, on continue
      - liste en cours de construction : [4, 16]
    - 23 > 16=> il est mal placé, on arrête le balayage gauche,
      - liste en cours de construction : [4, **23**, 16]
  - **Balayage à droite :**
    - 38 > 16 => il est dans la bonne sous-liste, on continue
      - liste en cours de construction : [4, **23**, 16, 38]
    - 18 > 16 => il est dans la bonne sous-liste, on continue
      - liste en cours de construction : [4, **23**, 16, 18, 38]
    - 45 > 16 => il est dans la bonne sous-liste, on continue
      - liste en cours de construction : [4, **23**, 16, 45, 18, 38]
    - 14 < 16 => il est mal placé, on arrête le balayage droit,
      - liste en cours de construction : [4, **23**, 16, **14**, 45, 18, 38]
  - **Echange des deux éléments mal placés** :
    - [4, **23**, 16, **14**, 45, 18, 38] => [4, 14, 16, 23, 45, 18, 38]

==> On reprend le balayage gauche à l'endroit où l'on s'était arrêté

Tri rapide : Exemple

- L = [4, 14, 3, 42, 2, 23, 45, 18, 38, 16], pivot = 16
  - **Balayage à gauche :**
    - 3 < 16 => il est dans la bonne sous-liste, on continue
      - liste en cours de construction : [4, 14, 3, 16, 23, 45, 18, 38]
    - 42 > 16=> il est mal placé, on arrête le balayage gauche,
      - liste en cours de construction : [4, 14, 3, **42**, 16, 23, 45, 18, 38]
  - **Balayage à droite :**
    - 2 < 16 => il est mal placé, on arrête le balayage droit,
      - liste en cours de construction : [4, 14, 3, **42**, 16, **2**, 23, 45, 18, 38]
  - **Echange des deux éléments mal placés** :
    - [4, 14, 3, **42**, 16, **2**, 23, 45, 18, 38] => [4, 14, 3, 2, 16, 42, 23, 45, 18, 38]

==> On arrête le balayage car on a atteint le pivot

- **Nous obtenons :**
  - L1 = [4, 14, 3, 2]
  - L2 = [23, 45, 18, 38, 42]
  - L **=** L1 **+ pivot +** L2 **=** [4, 14, 3, 2, 16, 23, 45, 18, 38, 42]

Tri rapide : Exemple

- En appliquant la même démarche au deux sous-listes L1 et L2, nous obtenons avec L = [4, 14, 3, 2, 16, 23, 45, 18, 38, 42]
  - L_{11}= [ ]
  - L_{12}=[3, 4, 14]
  - L_1=L_{11} + pivot + L_{12} = (2,3, 4, 14)
  - L_{21}=[23, 38, 18]
  - L_{22}=[45]
  - L_2=L_{21} + pivot + L_{22} = (23, 38, 18, 42, 45)
  - L = [(2,3, 4, 14), 16, (23, 38, 18, 42, 45)]