Les algorithmes de tri et de recherche

I. Les algorithmes de tri

II.1. Présentation

Le tri d'un tableau consiste à réordonner ses éléments dans l'ordre croissant ou décroissant. On distingue plusieurs méthodes de tri (sélection, bulles, insertion, …).

La plupart des algorithmes de tri sont basés sur deux opérations essentielles : la comparaison et l'échange (permutation).

Pour simplifier les algorithmes ultérieurs on commence par la définition d'une procédure qui a pour effet d'échanger deux valeurs A et B, de type entier par exemple :

```
0) Procédure Echange (var A, B : entier)
1) Aux ← A
2) A ← B ;
3) B ← Aux
4) Fin Echange
```

On fera appel à cette procédure dans la plupart des algorithmes que nous proposons d'exposer dans ce chapitre.

II.2. Les méthodes de tri

1. Tri par sélection

- **Principe de la méthode :**

C'est le principe le plus naturel. On commence par chercher la valeur la plus élevée du tableau, on la place alors à la dernière place en l'échangeant avec le dernier élément du tableau. Puis on réitère (répète) le procédé sur le **(n – 1)** éléments non encore triés, et ainsi de suite jusqu'aux deux premiers éléments.

- **Exemple :**

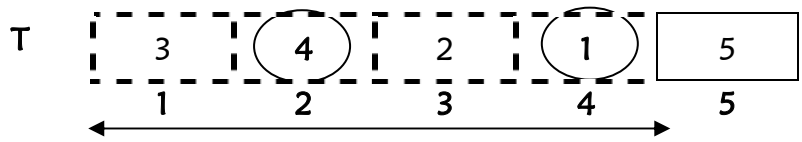
On considère le tableau **T**, non trié, ci-dessous :

T	3	4	2	5	1
	1	**2**	**3**	**4**	**5**

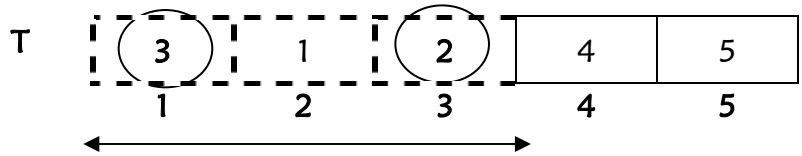
On représente ci-dessous l'évolution de celui-ci au fil des étapes du tri par sélection, en représentant sur chaque ligne en gras, dans un cercle, les éléments qui vont être échangés :

T	3	4	2	(5)	(1)
	1	**2**	**3**	**4**	**5**

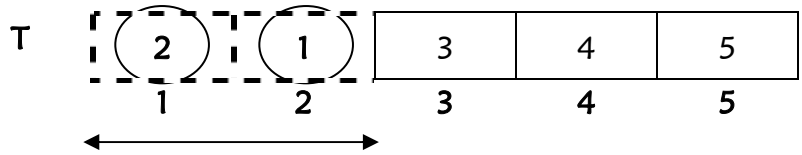
5 est la plus grande valeur : on l'échange avec **1,** le tableau devient alors :



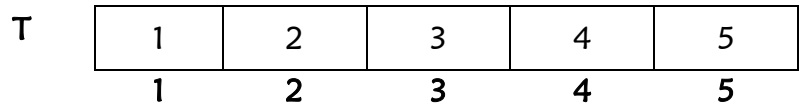
4 est la plus grande valeur restante : on l'échange avec **1,** le tableau devient alors :



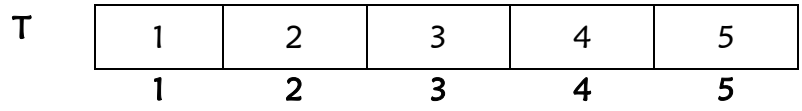
3 est la plus grande valeur restante : on l'échange avec **2,** le tableau devient alors :



2 est la plus grande valeur restante : on l'échange avec **1,** le tableau devient alors :



C'est la fin du tri



- **Mise en place de l'algorithme :**

Au vu du principe de la méthode, il est naturel que la procédure commence par une boucle descendante. On commence en effet par travailler sur le tableau entier, puis sur le **(n – 1)** éléments non triés et ainsi de suite jusqu'aux deux premier éléments du tableau.

Nous ferons donc appel à la boucle **Pour … Faire**

Pour K de n à 2 Faire

…

Fin Pour

A chaque étape, c'est-à-dire pour chaque valeur de **k**, il nous faut déterminer l'indice de la valeur la plus élevée parmi **T[1], T[2], …, T[k].**

Pour ce faire, on utilise une variable **max** qu'on initialise à **1** (valeur du premier indice) puis on parcourt le tableau **[T[2], …, T[k]]** en remplaçant **max** par **i** chaque fois que la valeur de **T[i]** est supérieur à celle de **T[max]** :

```
max ← 1
Pour i de 2 à k Faire
        Si T[i] > T[max] Alors max ← i
        Fin Si
Fin Pour
```

Il reste ensuite à échanger cette valeur avec **T [k]**, en faisant appel à la procédure Echange :

Echange (T [max], T [k])

D'où l'algorithme de la procédure tri_selection :

0) **Procédure** tri_selection (**var** T : tab ; n : entier)

1) **Pour** k **de** n **à** 2 **Faire**

```
    max ← 1
    Pour i de 2 à k Faire
        Si T[i] > T[max] Alors
            max ← i
        Fin Si
    Fin Pour
    Echange (T[max], T[i])
Fin Pour
```

2) **Fin tri_selection**

<u>Remarques</u> :

(1) Dans la boucle **Pour** i **de** 2 **à** k **Faire Si** T[i] > T[max] **Alors** max ← i

On effectue **(k – 1)** comparaisons. Comme **k** varie de **n** à **2**, le nombre total de comparaison est donc :

$$\sum_{k=2}^{n}(k-1)=\frac{n(n-1)}{2}$$

De plus on effectue un échange lors de chaque étape, il y a donc au total **(n – 1)** échanges effectués. De ces deux calculs on déduit que :

L'algorithme de tri par sélection a une complexité en $\Theta(n^2)$

(2) On a bien noté que dans notre algorithme la procédure Echange est appelée à chaque étape même lorsque le plus grand élément est déjà à sa place. Pour éviter cette anomalie on aurait pu écrire :

Si max > k **Alors** Echange (T [max], T [k]) **Fin Si**

Cela présente un intérêt limité car en réduisant le nombre d'échanges effectués on augmente le nombre de comparaison

2. Tri à bulles

■ Principe de la méthode :

Soit **T** un tableau de taille **n**, on part de la valeur de **T[1]**. Elle est comparée et éventuellement échangée avec **T[2]** puis éventuellement de **T[3]**, jusqu'à ce qu'elle rencontre une valeur qui lui est supérieure. C'est alors cette dernière qui est éventuellement échangée jusqu'à ce qu'à son tour elle rencontre une valeur plus grande et ainsi de suite.

Il est clair qu'à l'issue de ce processus, la valeur de **T[n]** est la plus grande. Il reste donc à réitérer le procédé sur le tableau **[T[1], T[2], …T[n-1]]**

■ Exemple :

On considère le tableau **T**, non trié, ci-dessous :

T	4	3	2	5	1
	1	**2**	**3**	**4**	**5**

On représente ci-dessous l'évolution de celui-ci au fil des étapes du tri à bulles, en représentant sur chaque ligne en gras, dans un cercle, les éléments qui vont être échangés :

Premier passage :

T	**4**	**3**	2	5	1
	1	**2**	**3**	**4**	**5**

4 > 3 : donc on échange **4** et **3** et le tableau devient :

T	3	**4**	**2**	5	1
	1	**2**	**3**	**4**	**5**

4 > 2 : donc on échange **4** et **2** et le tableau devient :

T	3	2	**4**	**5**	1
	1	**2**	**3**	**4**	**5**

4 < 5 : donc **4 s'arrête** et **5 entame sa montée** et le tableau devient :

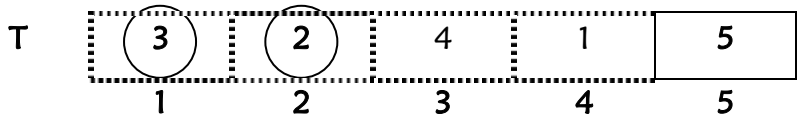
T	3	2	4	**5**	**1**
	1	**2**	**3**	**4**	**5**

5 > 1 : donc on échange **5** et **1** et le tableau devient :

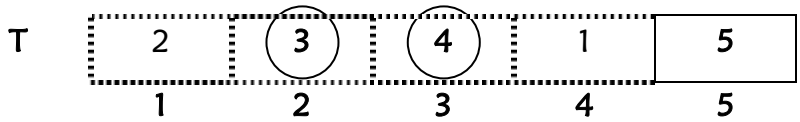
T	3	2	4	1	5
	1	**2**	**3**	**4**	**5**

La valeur **T[5]** est la plus grande. On répète le procédé sur le tableau **[T[1], T[2], T[3], T[4]]**

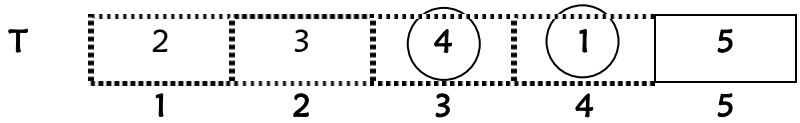
Deuxième passage :



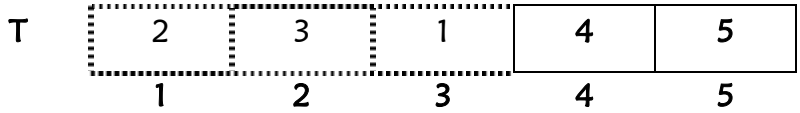
3 > 2 : donc on échange 3 et 2 et le tableau devient :



3 < 4 : donc 3 s'arrête et 4 entame sa montée et le tableau devient :

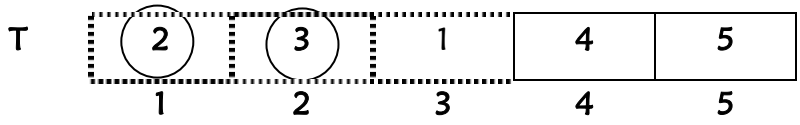


4 > 1 : donc on échange 4 et 1 et le tableau devient :

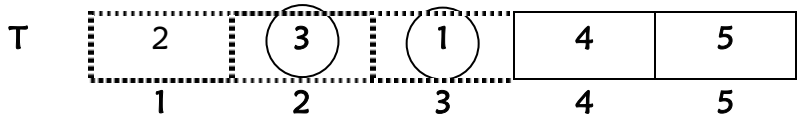


La valeur T[4] est la plus grande. On répète le procédé sur le tableau [T[1], T[2], T[3]]

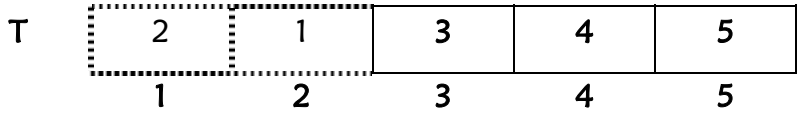
Troisième passage :



2 < 3 : donc 2 s'arrête et 3 entame sa montée et le tableau devient :

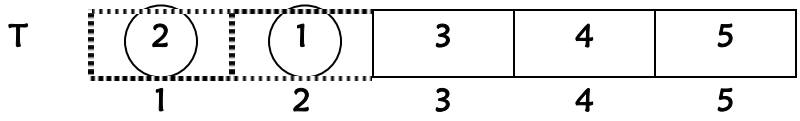


3 > 1 : donc on échange 3 et 1 et le tableau devient :

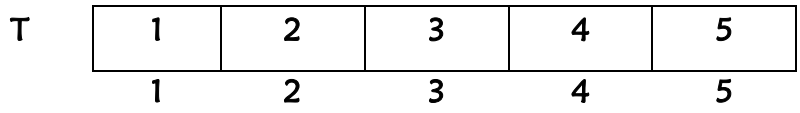


La valeur T[3] est la plus grande. On répète le procédé sur le tableau [T[1], T[2]]

Troisième passage :



2 > 1 : donc on échange 2 et 1 et le tableau devient :



C'est la fin du tri

- **Mise en place de l'algorithme :**

On travail d'abord sur le tableau complet, puis sur les **(n – 1)** premiers éléments et ainsi de suite. On emploie donc un traitement répétitif :

Pour k de n à 2 Faire

…

Fin Pour

Pour une valeur de **k** donnée, le tri à bulles consiste tout simplement à parcourir le tableau **[T[1], T[2],…, T[k]]** de gauche à droite en procédant à l'échange des éléments **T[i]** et **T[i+1]** à chaque fois que le premier est supérieur au second.

D'où l'algorithme de la procédure tri_bulles :

0) **Procédure** tri_bulles (**var** T : tab ; n : entier)
1) **Pour** k **de** n **à** 2 **Faire**
    Pour i **de** 1 **à** k-1 **Faire**
        Si T[i] > T[i+1] **Alors**
            Echange (T[i], T[i+1])
        Fin Si
    Fin Pour
  Fin Pour
2) **Fin** tri_bulles

<u>Remarques</u> :

(1) Dans la boucle **Pour** i **de** 1 **à** k -1 **Faire Si** T[i] > T[i+1] **Alors** Echange (T[i], T[i+1])

On effectue **(k – 1)** comparaisons. Comme **k** varie de **n** à **2**, le nombre total de comparaison est donc :

$$\sum_{k=2}^{n}(k-1)=\frac{n(n-1)}{2}$$

(2) Si le tableau est déjà trié, on peut procéder au minimum à **0** échanges et au maximum à $\sum_{k=2}^{n}(k-1)=\frac{n(n-1)}{2}$ échanges sinon.

(3) On obtient donc : $\frac{n(n-1)}{2}+\frac{n(n-1)}{2}=n(n-1)$ opérations

L'algorithme de tri à bulles a une complexité en $\Theta(n^2)$

3. Tri par insertion

■ Principe de la méthode :

Soit **T** un tableau de taille **n**. On considère que le tableau trié initiale est le tableau constitué du seul élément **T[1]**. On insère ensuite l'élément **T[2]** puis **T[3]** et ainsi de suite.

Lorsque les **(k – 1)** premiers éléments du tableau sont triés, il suffit pour insérer le **k**ième élément de parcourir le tableau **[T[1], T[2], …, T[k]]** de droite à gauche (ordre décroissant) et d'échanger notre élément avec les éléments rencontrés tant que ceux-ci sont plus grands.

■ Exemple :

On considère le tableau **T**, non trié, ci-dessous :

T	5	3	1	2	4
	1	2	3	4	5

On représente ci-dessous l'évolution de celui-ci au fil des étapes du tri par insertion, en représentant sur chaque ligne en gras, dans un cercle, les éléments qui vont être échangés :

Etape 1 : insertion de 3

T	5	3	1	2	4
	1	2	3	4	5

On a **3 < 5** : on échange donc **3** et **5** et le tableau devient

T	3	5	1	2	4
	1	2	3	4	5

Etape 2 : insertion de 1

On a **1 < 5** : on échange donc **1** et **5** et le tableau devient

T	3	1	5	2	4
	1	2	3	4	5

On a **1 < 3** : on échange donc **1** et **3** et le tableau devient

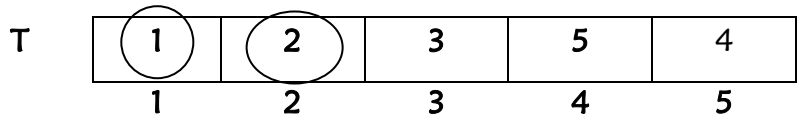
T	1	3	5	2	4
	1	2	3	4	5

Etape 3 : insertion de 2

On a **2< 5** : on échange donc **2** et **5** et le tableau devient

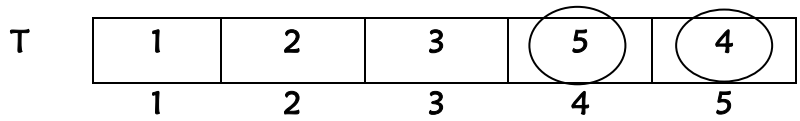
T	1	3	2	5	4
	1	2	3	4	5

On a **2< 3** : on échange donc **2** et **3** et le tableau devient

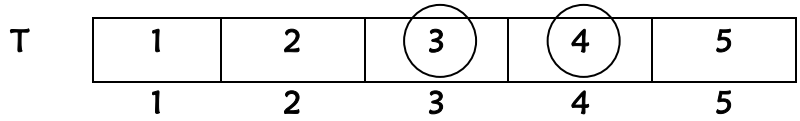


On a **1 < 2** : Fin de l'insertion

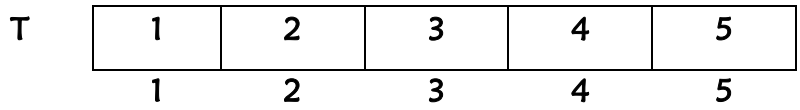
Etape 4 : insertion de 4



On a **4< 5** : on échange donc **4** et **5** et le tableau devient



On a **3**< **4**: Fin de l'insertion



- **Mise en place de l'algorithme :**

Initialement le tableau est constitué d'un seul élément **[T[1]]** est considéré comme déjà trié.

Le tri par insertion se fait ensuite en insérant, pour **k** variant de **2** à **n**, la valeur de **T[k]** au tableau **[T[1], T[1], …, T[k-1]]** préalablement trié.

On emploie donc la boucle :

Pour k de 2 à n Faire

…

Fin Pour

Pour un **k** donné, on insère la valeur de **T[k]** en parcourant le tableau **[T[1], T[1], …, T[k]]** dans l'ordre décroissant et en échangeant notre valeur avec les éléments tant que ceux-ci sont grands.

On ne connaît pas à l'avance le nombre d'échanges à effectuer, il peut y en avoir aucun, on emploie donc une boucle **Tant Que … Faire ou Répéter … Jusqu'à**.

Pour parcourir le tableau dans l'ordre décroissant on utilise un compteur **i** initialisé à **k** auquel on retire **1** à chaque passage dans la boucle. Enfin on procède aux échanges successifs par l'appel de la procédure **Echange (T[i], T[i-1]) Tant Que T[i] < T[i-1]** :

```
i ← k
Tant Que (T[i] < T[i-1] ) ET (i > 1) Faire
                    Echange (T[i] < T[i-1] )
                    i ← i – 1
Fin Tant Que
```

On note que la condition d'exécution (de continuation) est double. La condition **i > 1** a été ajoutée pour permettre de quitter la boucle si le tableau est entièrement parcouru.

D'où l'algorithme de la procédure tri_insertion :

0) **Procédure** tri_insertion (**var** T : tab ; n : entier)
1) **Pour** k **de** 2 **à** n **Faire**
2) i ← k

   Tant Que (T[i] < T[i-1]) **ET** (i > 1) **Faire**

   Echange (T[i] < T[i-1])
   i ← i – 1

   Fin Tant Que

   Fin Pour
3) **Fin** tri_insertion

Remarque :

(1) On a au maximum $\sum_{k=2}^{n}(k-1)=\frac{n(n-1)}{2}$ échanges (lorsque **T** est trié en ordre décroissant)

L'algorithme de tri par insertion a une complexité en $\Theta(n^2)$

II. Les algorithmes de recherche

II.1. Introduction

La recherche dans un tableau est un traitement très utilisé en informatique. On peut rechercher l'apparition d'un élément, sa ou ses positions, …

Dans cette partie du cours, nous allons présenter deux méthodes de recherche dans un tableau :

La recherche séquentielle et la recherche dichotomique.

II.2. Les méthodes de recherche

1. La recherche séquentielle

Cette méthode consiste à parcourir tout le tableau élément par élément et le comparer avec la valeur de l'élément à chercher.

Il s'agit d'un traitement répétitif à condition d'arrêt.

Application 1 :

Chercher la position de la première occurrence d'un élément **e** dans un vecteur **V** contenant **n** entiers (on suppose que **V** est définit)

Spécification de la fonction recherche_seq :

Résultat : chercher la position de la première occurrence de **e**.

Traitement :

La recherche de la position de la première occurrence de **e** dans le vecteur **V** est un traitement répétitif qui se fait à l'aide d'une boucle à condition d'arrêt.

Le traitement s'arrête quand on trouve la position de **e** dans **V** ou bien on dépasse la fin du vecteur. On utilise alors une variable booléenne **trouve** initialisée à **faux**

D'où l'algorithme de la fonction demandée :

```
0) Fonction recherche_seq (V : tab ; n, e : entier) : entier
1) i ← 1
2) Trouve ← faux
   Répéter
            Si V[i]=e Alors
                  Trouve=vrai
                  Sinon i ← i+1
            Fin Si
   Jusqu'à ((i>n) OU (Trouve))
   Si Trouve Alors
      recherche_seq ← i
   Sinon
      recherche_seq ← 0
   Fin Si
3) Fin recherche_seq
```

Application 2 :

Chercher le nombre d'apparition d'un élément **e** dans un vecteur **V** contenant **n** entiers ainsi que les positions des occurrences de cet élément.

2. la recherche dichotomique

Ce type de recherche s'effectue dans un tableau ordonné.

Le principe est le suivant :

- On divise le tableau en deux parties sensiblement égales (c.à.d à un élément prés).
- On compare la valeur à chercher avec l'élément du milieu.
- Si elles ne sont pas égales, on s'intéresse uniquement à la partie contenant les éléments voulus et on délaisse l'autre partie.

▪ On recommence ces **3** étapes jusqu'à avoir un seul élément à comparer.

Application 1 :

Chercher la position de la première occurrence d'un élément **e** dans un vecteur **V** contenant **n** entiers (on suppose que **V** est défini)

Spécification de la fonction recherche_dico :

Résultat : chercher la position de la première occurrence de **e**.

Traitement :

On suppose que **V** est déjà trié (sinon utiliser une méthode de tri pour l'ordonner)

La recherche de la position de la première occurrence de **e** dans le vecteur **V** est un traitement répétitif qui se fait à l'aide d'une boucle à condition d'arrêt.

Le traitement s'arrête quand on trouve la position de **e** dans **V** ou bien on la borne gauche de l'intervalle devient supérieur à la borne droite. On utilise alors une variable booléenne **trouve** initialisée à **faux,** et en initialisant la borne gauche **(inf)** à **1** et la borne de droite **(sup)** à **n**, la taille du vecteur.

D'où l'algorithme de la fonction demandée :

```
0) Fonction recherche_dico (V : tab ; n, e : entier) : entier
1) Trouve ← Faux
2) Inf ← 1
3) Sup ← n
4) Répéter
            Mil ← (inf + sup) div 2
            Si V[mil] = e Alors
                  Trouve ← vrai
               Sinon Si V[mil] < e Alors
                  Inf ← mil + 1
                  Sinon sup ← mil – 1
            Fin Si
   Jusqu'à ((inf > sup) OU (Trouve))
5) Fin recherche_dico
```

Application 2 :

Chercher le nombre d'apparition d'un élément **e** dans un vecteur **V** contenant **n** entiers ainsi que les positions des occurrences de cet élément.