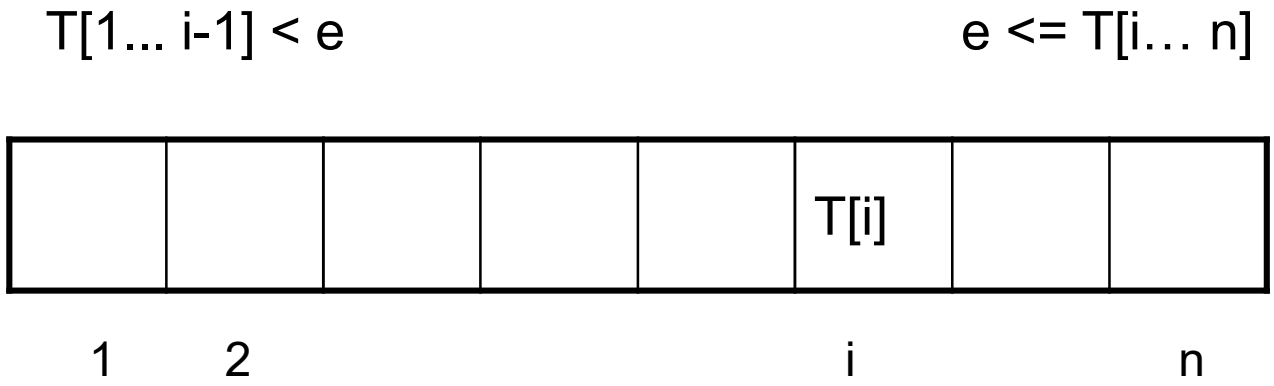
Chapitre 6
Insertion et suppression dans un tableau

1- Recherche séquentielle d'un élément e dans un tableau trié T




```
i:=1;
Tant que ((i<=n) et (T[i]<e))
      faire        i:=i+1;
Fin tant que
```

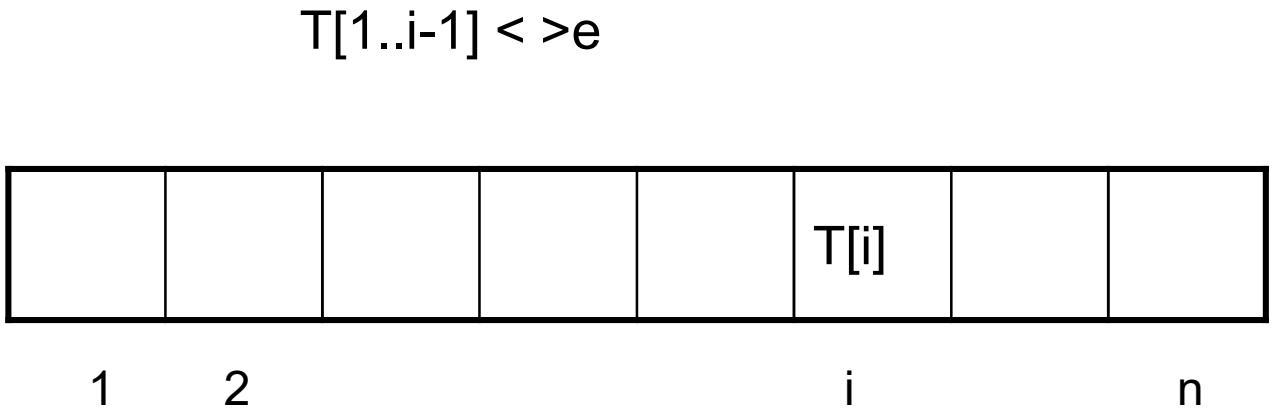
Conditions d'arrêt de la boucle:

(i=n+1) ou T[i]>=e

Si ((i=n+1) ou T[i]>e) alors e n'existe pas dans T,
Sinon e existe dans T et T[i]=e
Fin Si

- **Complexité** :
  - au mieux : *O(1)*
  - au pire : *O(n)*

2- Recherche séquentielle d'un élément e dans un tableau T non trié




```
i:=1;
Tant que ((i<=n) et (T[i]< > e))
     faire i:=i+1;
Fin tant que
```

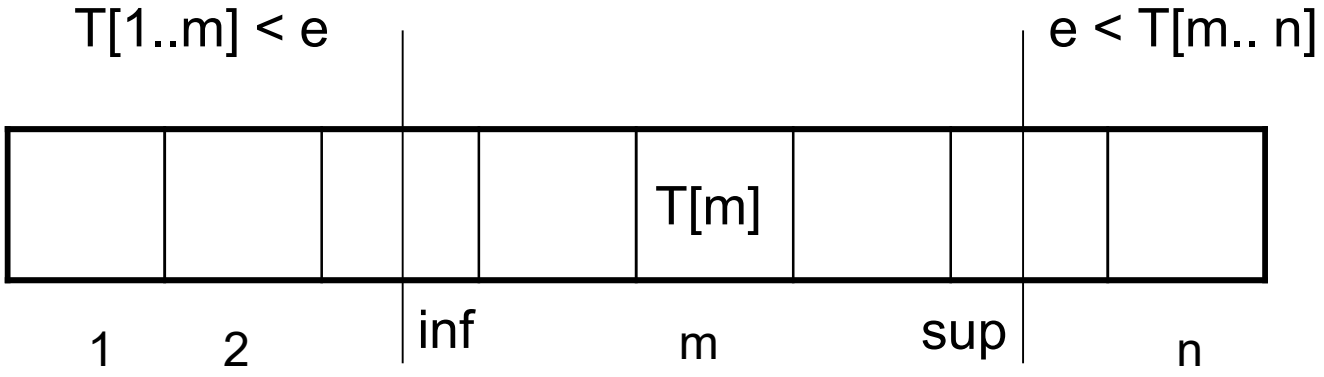
Conditions d'arrêt de la boucle:

i=n+1 ou T[i]=e

Si (i=n+1) alors e n'existe pas dans T,
Sinon e existe dans T et T[i]=e
finSi

- **Complexité** :
  - au mieux : *O(1)*
  - au pire : *O(n)*

3- Recherche dichotomique d'un élément e dans un tableau trié T




```
trouvé:=faux; inf:=1; sup:=n;
Tant que ((inf<=sup)et (non trouvé)) faire
        m:=(inf +sup) div2;
        si T[m]=e alors trouvé:=vrai;
          sinon
            si T[m]<e alors inf:=m+1;
              sinon sup:=m-1;
            fin si
        fin si
Fin tant que
```

- Complexité

- au mieux : *O(1)*
- au pire : $O(\log_2(n))$

4- Insertion d'un élément dans un tableau non trié

• A priori, on ajoute le nouvel élément en fin de tableau

• S'il existe un critère définissant l'endroit d'insertion, on est dans un cas comparable au tableau trié

5- Insertion d'un élément dans un tableau trié

- Chercher la place de l'élément à insérer
- Effectuer l'insertion

6- Méthode séquentielle de recherche de la position d'insertion dans un tableau trié

```
fonction position(T[n],e :entier)
p,i: entier;
début
        si (T[1]>e)
           alors p:=1
          sinon
                { i:=n;
                  tant que (T[i] > e)
                          faire i:=i-1
                  fin tant que
                 p:=i+1;
                }
         fin si
        retourner(p);
fin
```

- Complexité :

- au mieux : O(1)
- au pire : O(n)

7- Méthode dichotomique de recherche de la position d'insertion dans un tableau trié

```
fonction position(T[n],e:entier)
p,inf,sup,m:entier;
début
    si T[n]<=e
      alors p ←n+1;
     sinon
      { inf ←1; sup ←n;
         tant que inf< sup faire
           { m ← (inf+ sup) div 2;
             si T[m]<e
                    alors inf ← m+1;
               sinon sup ←m;
              fin si
            }
         fin tant que
       p ←sup; }
   fin si
   retourner(p);
fin
```

- Complexité :
  - au mieux : O(1)
  - au pire : $O(\log_2(n))$

8- Insertion d'un élément e connaissant la position p souhaitée

```
début
   pour i=n à p faire
      T[i+1] ←T[i]
   finpour;
   n ←n+1;
   T[p] ←e;
fin
```

- Complexité :
  - au mieux : O(1)
  - au pire :O(n)

9- Suppression d'un élément dans un tableau trié

- **Rechercher une occurrence de la valeur e à supprimer**
- **Si on la trouve, «tasser» les éléments du tableau**

Chapitre 7
Algorithmes de Tri

Algorithmes de Tri

- **Le tri consiste à ordonner les éléments du tableau dans l'ordre croissant ou décroissant**

- **Il existe plusieurs algorithmes connus pour trier les éléments d'un tableau :**
  - Le tri par sélection
  - Le tri par insertion
  - Le tri rapide
  - …

- **Nous verrons dans la suite les algorithmes de tri par sélection, de tri par insertion et de tri rapide. Le tri sera effectué dans l'ordre croissant**

1- Tri par sélection

- **Principe** : à l'étape i, on sélectionne le plus petit élément parmi les (n - i +1) éléments du tableau les plus à droite. On l'échange ensuite avec l'élément i du tableau

- **Exemple**

9	4	1	7	3

  - **Étape 1:** on cherche le plus petit parmi les 5 éléments du tableau. On l'identifie en troisième position, et on l'échange alors avec l'élément 1 :

1	4	9	7	3

  - **Étape 2:** on cherche le plus petit élément, mais cette fois à partir du deuxième élément. On le trouve en dernière position, on l'échange avec le deuxième:

1	3	9	7	4

  - **Étape 3:**

1	3	4	7	9

1-1 Tri par sélection : algorithme

- **Supposons que le tableau est noté T et sa taille N**

```
Pour i  allant de 1 à N-1
        indice_ppe ← i ;
        Pour j allant de i + 1 à N
            Si T[j] <T[indice_ppe] alors
                indice_ppe  ← j ;
            Finsi
         FinPour

        temp ← T[indice_ppe] ;
        T[indice_ppe] ← T[i] ;
        T[i] ← temp ;
FinPour
```

1-2 Tri par sélection : complexité

- Quel que soit l'ordre du tableau initial, le nombre de tests et d'échanges reste le même

- On effectue N-1 tests pour trouver le premier élément du tableau trié, N-2 tests pour le deuxième, et ainsi de suite. Soit : **(N-1)+(N-2)+…+1 = N(N-1)/2.**

  On effectue en plus (N-1) échanges.

- La **complexité** du tri par sélection est **d'ordre N²** à la fois dans le meilleur des cas, en moyenne et dans le pire des cas

- Pour un ordinateur qui effectue **10^6** instructions par seconde on a :

N	10^3	10^6	10^9
temps	1s	11,5 jours	32000 ans

2- Tri par insertion

Le principe du tri par insertion est le suivant :

On souhaite trier un tableau de n données ; on procède par étapes ;

✓ **à la i** ème **étape (i variant de 1 à N – 1), on suppose que les i premières données sont déjà triées ;**
✓ **on considère alors la (i + 1)** ème **donnée que l'on appelle pivot ;**
✓ **on la compare successivement aux données précédentes, en commençant par la i** ème **puis en remontant dans le tableau jusqu'à trouver la bonne place du pivot (c'est-à-dire entre deux données successives, l'une étant plus petite et l'autre plus grande que le pivot ou bien en tout premier si le pivot est plus petit que les i premières données);**

✓**au fur et à mesure, on décale « d'une case vers la droite » les données plus grandes que le pivot de façon à anticiper la place de ces données après insertion du pivot ;**

✓**on met le pivot à la bonne place ; à l'issue de cette étape, les i + 1 premières données sont donc triées.**

2-1 Tri par insertion : algorithme

Les données sont rangées dans un tableau T entre les indices 1 et N.
On utilise deux variables entières notées i, j et une variable pivot du même type que les données du tableau.

```
pour i de 2 à N
        j ← i ;
        pivot ←T[j] ;
        tant que j ≥ 2 et T[j – 1] > pivot, faire
                T[j ] ← T[j – 1] ;
                j ← j – 1 ;
        Fin tant que
        T[j] ← pivot.
Fin pour
```

2-2 Tri par insertion : complexité

- Ce tri est en $O(N^2)$ dans le pire des cas (cas où le tableau est trié dans l'ordre inverse) et en O(N) dans le meilleur des cas (cas où le tableau serait déjà trié avant application de l'algorithme) .
- le tri par insertion est en moyenne en $O(N^2)$.
- C'est donc encore un tri peu efficace mais qui a aussi le mérite de la simplicité.

- Le tri par insertion a une complexité dans le pire des cas du même ordre que celle du tri par sélection ;
- néanmoins, si les données à trier sont souvent presque triées (dans le sens souhaité), il vaudra mieux choisir le tri insertion.

3- Tri rapide

- **Le tri rapide est un tri récursif basé sur l'approche "diviser pour régner" (consiste à décomposer un problème d'une taille donnée à des sous problèmes similaires mais de taille inférieure faciles à résoudre)**

- **<u>Description du tri rapide :</u>**

  1) On considère un élément du tableau qu'on appelle pivot,

  2) on partitionne le tableau en 2 sous tableaux : les éléments inférieurs ou égaux à pivot et les éléments supérieurs à pivot. on peut placer ainsi la valeur du pivot à sa place définitive entre les deux sous tableaux

  3) on répète récursivement ce partitionnement sur chacun des sous tableaux crées jusqu'à ce qu'ils soient réduits à un à un seul élément.

❖ **Plus précisément, on utilise une fonction nommée partition qui s'applique à un tableau T;**

❖ **cette fonction extrait la donnée pivot, du tableau T, par exemple la première (c'est ce que nous choisirons plus bas) puis elle réorganise le tableau de sorte qu'on trouve d'abord les données ne dépassant pas la valeur du pivot (on appellera sous-tableau gauche cette partie du tableau), puis la donnée pivot, puis les données supérieures au pivot (on appellera sous-tableau droite cette partie du tableau).**

❖ **Pour appliquer le tri rapide à un tableau ayant au moins deux données, on commence par appliquer la fonction partition, puis on applique le tri rapide au sous-tableau gauche, puis on applique le tri rapide au sous-tableau droite.**

3-1 Fonction de partition

```
Fonction Partition(tableau T : réel par adresse, p,r: entier par valeur)
    Variables  i, j: entier ;
     pivot: réel;
Début
      pivot← T[p];  i←p+1;  j ← r;
    TantQue (i <= j)
          TantQue (i<= r et T[i] <= pivot)     i ← i+1;  FinTantQue
          TantQue (j>= p et T[j] > pivot )  j ← j-1;    FinTantQue
          Si  i < j  alors
                    Echanger(T[i], T[j]);  i ← i+1;  j ← j-1 ;
          FinSi
    FinTantQue
     Echanger(T[j], T[p]);
     return j;
Fin
```

Exemple

- Soit le tableau suivant

1	2	3	4	5	6	7	8	9	10	11	12
9	4	18	11	15	5	17	1	10	7	12	6

- Les données sont les valeurs qui figurent dans la seconde ligne du tableau T suivant, la première précisant les indices des cases de T :
- On suppose qu'on applique la fonction partition avec p = 1 et r = 12.
- La variable pivot vaut T[1] = 9.
- Au premier passage dans la boucle externe, i s'arrête à 3, j s'arrête à 12.
- On procède à l'échange et on obtient :

1	2	3	4	5	6	7	8	9	10	11	12
9	4	6	11	15	5	17	1	10	7	12	18

Après cet échange i passe 4 et j à 11.

- Au deuxième passage dans la boucle, i s'arrête à 4 et j s'arrête à 10. On procède à l'échange et on obtient

1	2	3	4	5	6	7	8	9	10	11	12
9	4	6	7	15	5	17	1	10	11	12	18

avec i = 5 et j = 8.

- Au troisième passage dans la boucle, i s'arrête à 5 et j s'arrête à 8. On procède à l'échange et on obtient :

1	2	3	4	5	6	7	8	9	10	11	12
9	4	6	7	1	5	17	15	10	11	12	18

avec i = 6 et j = 7.

- Au quatrième passage dans la boucle, i s'arrête à 7 et j s'arrête à 6. Le test « i < j ? » est négatif car on a i > j.

On sort de la boucle externe « tant que i ≤ j » car on a i > j ; on échange T[1] avec T[j], i.e. T[6] ; on obtient :

1	2	3	4	5	6	7	8	9	10	11	12
5	4	6	7	1	9	17	15	10	11	12	18

La fonction partition va retourner l'indice 6. On remarque que la donnée de valeur 9 est à sa place définitive, qu'avant elle se trouvent des données inférieures à 9 et, après, des données supérieures à 9.

3-2 Procédure Tri rapide

```
Procédure TriRapide(tableau T : réel par adresse, p,r: entier par valeur)
    variable q: entier;
Début
    Si  p < r  alors
        q=Partition(T,p,r);
        TriRapide(T,p,q-1);
        TriRapide(T,q+1,r);
    FinSi
 Fin
```

- A chaque étape de récursivité on partitionne un tableau T[p..r] en deux sous tableaux T[p..q-1] et T[q+1..r] tel que chaque élément de T[p..q-1] soit inférieur ou égal à chaque élément de T[q+1..r] .
- L'indice q est retourné par la fonction de partitionnement

- Sur notre exemple, il reste à appliquer le tri rapide entre les indices 1 et 5 puis entre les indices 7 et 12.

L'application de TriRapide (1, 5) **appelle** Partition(1, 5) **qui conduit au tableau :**

1	2	3	4	5	6	7	8	9	10	11	12
1	4	5	7	6	9	17	15	10	11	12	18

et renvoie la valeur 3. On applique alors TriRapide(1, 2), **qui appelle** Partition(1, 2) **qui ne change pas le tableau et retourne la valeur 1, puis** TriRapide (1, 0) **qui ne fait rien, puis** TriRapide (4, 5) **qui appelle** Partition(4, 5) **qui conduit au tableau :**

1	2	3	4	5	6	7	8	9	10	11	12
1	4	5	6	7	9	17	15	10	11	12	18

et retourne l'indice 4.

- Puis, TriRapide(4, 3) **ne fait rien ainsi que** TriRapide (5, 5). **Le tri rapide entre les indices 1 et 5 est terminé.**

- L'application de TriRapide (7, 12) **appelle** Partition(7, 12) **qui conduit au tableau :**

1	2	3	4	5	6	7	8	9	10	11	12
1	4	5	6	7	9	12	15	10	11	17	18

et renvoie la valeur 11. On applique alors TriRapide(7, 10), **qui appelle** Partition(7, 10) **qui conduit au tableau :**

1	2	3	4	5	6	7	8	9	10	11	12
1	4	5	6	7	9	10	11	12	15	17	18

et retourne l'indice 9.

- Puis, TriRapide(7, 8) **fait appel à** Partition(7, 8) **qui ne change pas le tableau et retourne l'indice 7, puis** TriRapide(7, 6) **ainsi que** TriRapide(8, 8) **ne font rien. On « remonte » avec** TriRapide(12, 12) **qui ne fait rien. - L'algorithme est terminé.**

3-3 Tri rapide : complexité et remarques

- **La complexité du tri rapide dans le pire des cas est en $O(N^2)$**
- **La complexité du tri rapide en moyenne est en O(N log N)**
- **Le choix du pivot influence largement les performances du tri rapide**
- **Le pire des cas correspond au cas où le pivot est à chaque choix le plus petit élément du tableau (tableau déjà trié)**
- **différentes versions du tri rapide sont proposés dans la littérature pour rendre le pire des cas le plus improbable possible, ce qui rend cette méthode la plus rapide en moyenne parmi toutes celles utilisées.**