Chapitre 2: Les algorithmes de recherche et de tri

Recherche dans un tableau

- Trouver l'indice d'un élément particulier dans le tableau.
- Une première solution est de parcourir tout le tableau et de s'arrêter dès que l'on trouve la valeur cherchée.
- Il faut prévoir de s'arrêter à la fin du tableau si la valeur n'est pas présente.

Recherche séquentielle

Algorithme

fonction recherche(in tab : tableau d'entiers, in taille : entier, in val : entier) : ret entier

{Renvoie l'indice de val dans tab de taille taille si elle y est. Renvoie -1 sinon.}

Variable i, trouve : entier;

Début

trouve ← -1; i ← 0;

Tant que (trouve=-1 et i < taille) faire

    Si tab[i]=val faire

        trouve ← i;

    FSI

    i ← i+1;

FTq

retourner trouve

Fin

Recherche dichotomique

- Ne s'applique qu'à des tableaux triés dont on connaît le sens du tri.
- Principe : réduire l'intervalle de recherche
  - Choisir un élément dans le tableau : t[i]

Éléments <= t[i]	t[i]	Éléments >= t[i]

  - Si la valeur recherchée est égale à t[i], on a trouvé

Éléments <= t[i]	t[i]	Éléments >= t[i]

  - Si la valeur recherchée est > t[i], on continue la recherche dans la partie des éléments >= t[i]

Éléments <= t[i]	t[i]	Éléments >= t[i]

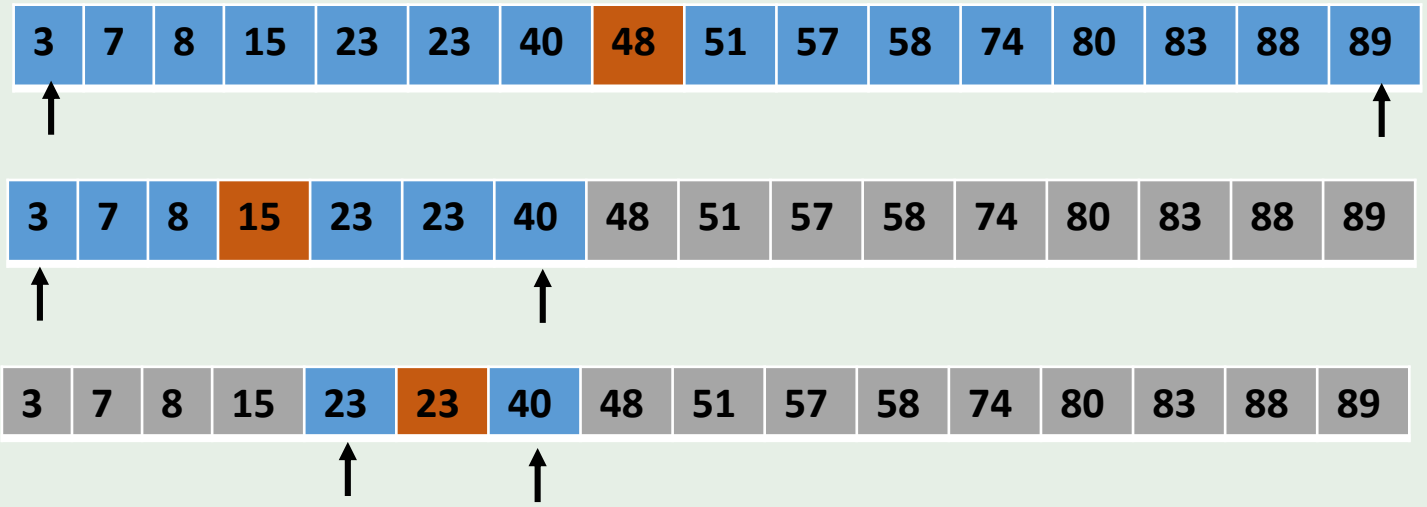
  - Si la valeur recherchée est < t[i], on continue la recherche dans la partie des éléments <= t[i]

Éléments <= t[i]	t[i]	Éléments >= t[i]

Recherche dichotomique

- Arrêt de la recherche :
  - lorsque l'on trouve l'élément recherché ;
  - lorsque l'on arrive sur un intervalle de recherche vide.
- En général, on choisit l'élément du milieu pour la séparation.

Exemple: recherche de 23



Recherche dichotomique: Algorithme itératif

- Idée de l'algorithme :
  - Vérifier si l'on est sur la case recherchée, ou que l'on a terminé la recherche.
  - Si ce n'est pas le cas, regarder de quel côté peut se trouver la valeur.
  - Si on a terminé, on renvoie l'indice si c'est la bonne case, ou -1 sinon.

Algorithme

fonction rechDicholter(in tab : tableau d'entiers, in début: entier, in fin : entier, in val : entier) : ret entier
{Renvoie l'indice de val dans tab de taille taille si elle y est. Renvoie -1 sinon.}
Variable min, max, milieu, trouvé : entier;
Début

Recherche dichotomique: Algorithme itératif

Algorithme (suite)

```
trouvé ← -1; min←début; max←fin;
Tant que (trouvé=-1 ou min <= max) faire
   milieu ← (min+max)/2;
   Si val= tab[milieu] faire
      trouvé=milieu;
   Sinon Si val< tab[milieu] faire
                max=milieu-1;
             Sinon
               min=milieu+1;
              Fsi
  FSi
FTq
retourner trouvé;
Fin
```

Recherche dichotomique: Algorithme récursif

- Idée de l'algorithme :
  - Vérifier si l'on est sur la case recherchée, ou que l'on a terminé la recherche.
  - Si ce n'est pas le cas, regarder de quel côté peut se trouver la valeur.
  - Si on a terminé, on renvoie l'indice si c'est la bonne case, ou -1 sinon.

Algorithme

fonction rechDichoRec(in tab : tableau d'entiers, in début: entier, in fin : entier, in val : entier) : ret entier
{Renvoie l'indice de val dans tab de taille taille si elle y est. Renvoie -1 sinon.}
Variable milieu, trouvé : entier;
Début

Recherche dichotomique: Algorithme récursif

Algorithme (suite)

```
Si début>fin
   trouvé←-1;
Sinon
   milieu ← (min+max)/2;
   Si val= tab[milieu] faire
      trouvé ← milieu;
   Sinon   Si val< tab[milieu] faire
                rechDichoRec(tab, début, milieu-1, val);
           Sinon
                rechDichoRec(tab, milieu+1,fin, val);
            FSi
   FSi
FSi
retourner trouvé;
Fin
```

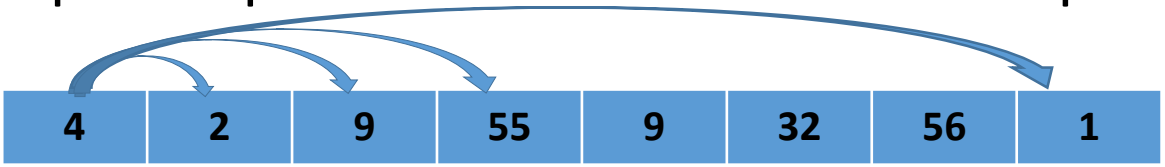
Algorithmes de tri

- La notion d'ordre n'est pas implicite dans un tableau.
- Il est souvent nécessaire d'avoir des valeurs triées. Exemple : recherche dichotomique.
- Nous allons voir les tris en ordre croissant suivants :
  - ✓tri par échange;
  - ✓tri par insertion ;
  - ✓tri par sélection-permutation ;
  - ✓tri à bulles ;

- Ces tris simples sont peu efficaces sur de grands vecteurs.
- On utilise d'autre algorithmes comme, par exemple :
  - ✓tri fusion ;
  - ✓tri rapide (quick sort).

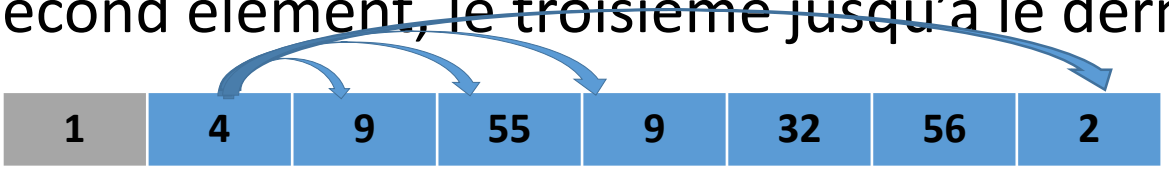
Tri par échange

- Le tri par échange est un algorithme de tri peu performant à cause de sa lenteur d'exécution
- Principe : prendre chaque élément du tableau et le comparer à tous les éléments suivant, lorsque l'ordre n'est pas respecté les deux éléments sont permutés



- Lorsque on termine de parcourir le tableau pour la première fois, on est sur d'avoir placé le premier élément à la bonne place

- On parcourt donc de nouveau le tableau pour placer le second élément, le troisième jusqu'à le dernier.



Tri par échange

Algorithme

```
fonction triEchan(in-out tab : tableau d'entiers, in taille : entier) : vide
variables  i, j, echange: entiers;
Début
pour i de 0 à taille-2 faire
    pour j de i+1 à taille-1 faire
        Si T[j]<T[i] faire
            echange ← T[i];
            T[i] ← T[j];
            T[j] ← echange;
        FSi
    fpour
fpour
Fin
```

Tri par insertion

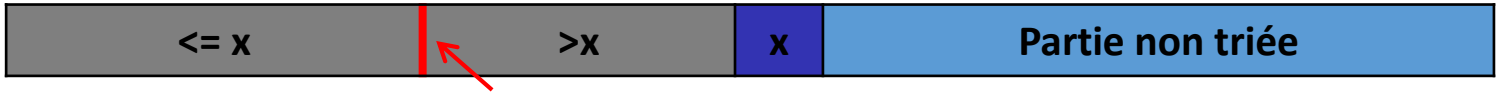
- Utilisé pour trier les cartes que l'on a en main.
- Principe : on prend un élément et on recherche sa place dans la partie déjà triée puis on l'insère à cet endroit, et ainsi de suite. . .
- Application aux tableaux :
  - ✓Deux parties dans le tableau :

partie triée	partie non triée

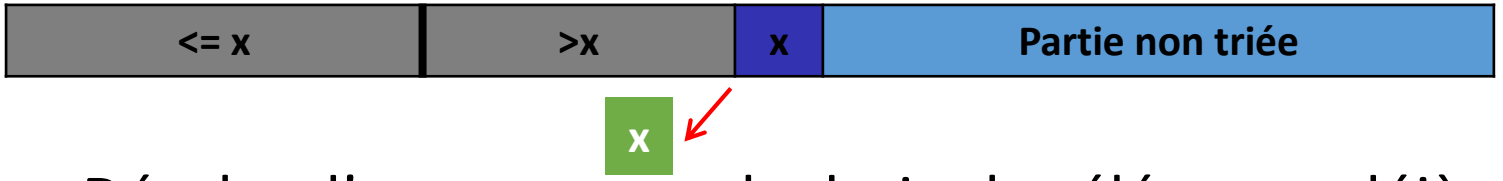
  - ✓On procède itérativement en prenant à chaque fois un élément de la partie non triée et en l'insérant à la bonne place dans la partie triée.
- À chaque fois, on prend le 1er élément de la partie non triée comme élément à insérer dans la partie déjà triée.

Tri par insertion

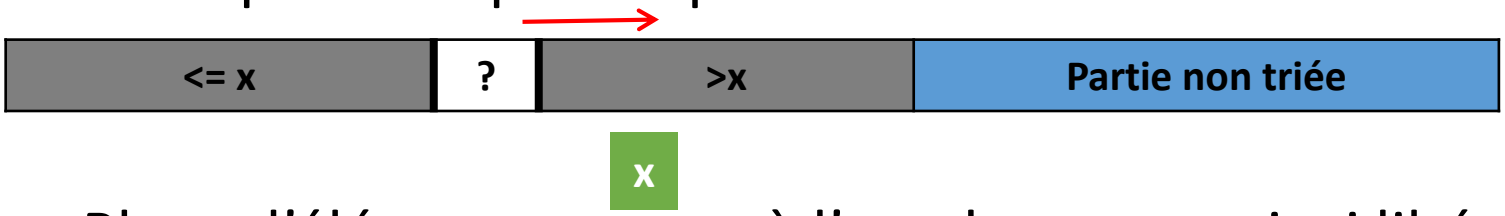
- L'insertion de l'élément courant se fait en 4 étapes :
  - Trouver le point d'insertion dans la partie triée.



  - Recopier l'élément courant



  - Décaler d'une case vers la droite les éléments déjà triés qui sont après le point d'insertion.



  - Placer l'élément courant à l'emplacement ainsi libéré

Tri par insertion

- On voit que la nouvelle partie grisée est toujours triée :

- Au départ, on a logiquement :
  partie triée vide
  partie non triée tout le tableau
- Mais on peut gagner une étape en prenant :
  partie triée 1er élément du tableau
  partie non triée le reste du tableau
- La recherche du point d'insertion et le décalage des éléments peuvent se faire en même temps.
- On termine quand la partie non triée devient vide : quand le dernier élément du tableau a été inséré dans la partie triée.

Tri par insertion

Algorithme

fonction triInsert(**in-out** *tab* : tableau d'entiers, **in** *taille* : entier) : **vide**
variables valeur, limite, place : entiers;
Début
pour *limite* **de** 1 **à** *taille*-1 **faire**
    valeur ← *tab*[*limite*];
    place ← *limite;*
    tant que *place* >= 0 et *tab*[*place*-1] > *valeur* **faire**
        tab[*place*] ← *tab*[*place*-1];
        place ← *place*-1;
    ftant
    tab[*place*] ← *valeur;*
fpour
Fin

Tri par sélection-permutation

- On aimerait ne plus avoir à modifier la partie déjà triée
  - On distingue toujours les deux parties des éléments mais cette fois :

partie triée (élts <= élts non triés)	**partie non triée (élts >= élts triés)**

  - Ajout d'un élément à la fin de la partie triée mais il doit être inférieur aux éléments de la partie non triée
- Principe :
  - on cherche le plus petit élément dans la partie non triée (sélection) et on le place au début de cette partie (permutation).
- Nouvelle zone triée = ancienne zone triée, plus cet élément.
- Nouvelle zone non triée = ancienne zone non triée, moins cet élément.

Tri par sélection-permutation

- La sélection-permutation d'un élément se fait en 2 étapes :
- Trouver le minimum dans la partie non triée : x

partie triée <= partie non triée>=

- Permuter avec le premier élément de la partie non triée

partie triée <= partie non triée>=

partie triée <= partie non triée>=

Tri par sélection-permutation

- On voit que la nouvelle partie grisée est toujours triée :

- Au départ, on a logiquement :
  partie triée = vide
  partie non triée = tout le tableau
- Cette fois, on peut gagner une étape à la fin (dernier élément).
- On termine quand la partie non triée ne contient plus qu'un seul élément : il s'incorpore directement dans la partie triée.

Tri par sélection-permutation

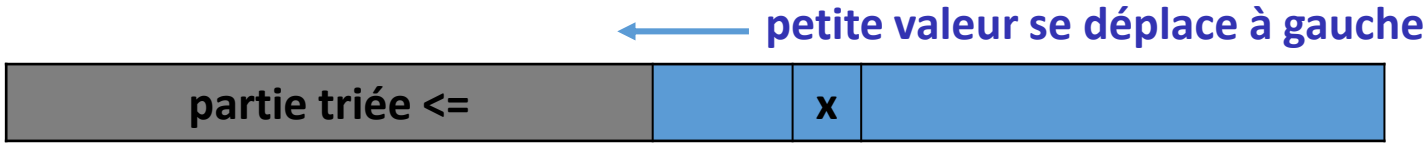
Algorithme

```
fonction triSélect(in-out tab : tableau d'entiers, in taille : entier) : vide
Variables iMin, min, limite, i  : entier;
Début
pour limite de 0 à taille - 2 faire
   iMin ← limite;
   pour i de limite+1 à taille-1 faire
          si tab[i] < tab[iMin] alors
             iMin ← i;
          fsi
   fpour
   min ← tab[iMin];
   tab[iMin] ← tab[limite];
   tab[limite] ← min;
fpour
Fin
```

Tri a Bulles

- On aimerait éviter le calcul des minimums (recherche coûteuses).
- On distingue toujours la partie triée (<=) et la partie non triée :

- On fait descendre les éléments les plus petits par parcours successifs :



- **Principe :**

  Parcours droite–gauche de la partie non triée avec comparaison deux à deux des éléments consécutifs et remise en ordre éventuelle.

- Nouvelle zone triée = ancienne zone triée plus élément minimum de la zone non triée.
- Nouvelle zone non triée = ancienne zone non triée, moins cet élément.

Tri a Bulles

Algorithme

```
fonction triBulles(in-out tab : tableau d'entiers, in taille : entier) : vide
Variables tmp, limite,i : entiers
pour limite de 0 à taille-2 faire
  pour i de taille-1 à limite+1 pas <-1> faire
     si tab[i] < tab[i-1] alors
          tmp ← tab[i];
          tab[i] ← tab[i-1];
          tab[i-1] ← tmp;
     fsi
   fpour
fpour
```

Devoir à domicile (à rendre début janvier 2017)

- Implémenter en langage C les différents type d'algorithmes de tri vus dans ce chapitre et les autres qui existent (tri par fusion, tri rapide, …).
- Comparer la performance machine (temps d'éxécutions) des différents algorithmes sur des exemples aléatoires des tableaux.
- Illustrer vos comparaisons à l'aide des tableaux et des graphiques.
- Réaliser un rapport accompagné d'un CD comportant les programmes sources.

- N.B: Vous pouvez travaillez en groupe de deux personnes maximum.