Résumé (1/3)

pour évaluer la complexité des algorithmes :

- on comptera les affectations de valeurs du même type que l'exemplaire
- les comparaisons de valeurs du même type que l'exemplaire

on fait correspondre ce nombre au temps mis par l'algorithme pour s'exécuter : si on associe un temps unitaire à l'opération comptée, on obtient une évaluation du temps d'exécution

- les espaces mémoires utilisés en plus de ceux de l'exemplaire et du même type que l'exemplaire

l'opération mesurée ou l'unité de mémoire utilisée est celle qui est la plus importante vis-à-vis de nos données

Résumé (2/3)

on a mesuré pour :

- les boucles `pour` : il suffit de compter l'intervalle sur lequel elle est réalisée multiplié par le nombre d'opérations réalisées dans la boucle, cela va dépendre de la taille de la donnée
- les boucles `tant que` : il faut savoir compter le nombre exact de fois où la boucle est réalisée, cela va dépendre de la taille de la donnée et de la donnée elle-même
- les récursions : il faut établir une équation qui va sommer le nombre d'opérations réalisées lors d'un appel et le nombre de fois où la procédure est appelée récursivement, il faut aussi distinguer le cas de base

Résumé (3/3)

on a trouvé que :

- tous les algorithmes ne font pas le même nombre d'opérations (v2 et v3 réalisent le moins de comparaisons)
- tous les algorithmes ne consomment pas la même quantité de mémoire (v3 consomme plus de mémoire que v1 et v2)
- ces comptages dépendent de la taille de l'exemplaire (ici la longueur du tableau), et parfois de la nature de l'exemplaire (ici le contenu du tableau)

Différents cas

- pire des cas : correspond à un exemplaire ou un ensemble d'exemplaires pour lesquels l'algorithme s'exécute en temps maximal (ou avec un espace maximal)
  dans l'exemple, quand v n'est pas présent
- meilleur des cas : correspond à un exemplaire ou un ensemble d'exemplaires pour lesquels l'algorithme s'exécute en temps minimal (ou avec un espace minimal)
  dans l'exemple, quand v est en première position
- moyenne : correspond à la complexité moyenne sur tous les exemplaires possibles
  dans l'exemple . . . il faudrait savoir dénombrer tous les cas et réaliser la moyenne !

Il se peut que le meilleur des cas et le pire des cas soient confondus.

La notion de complexité

Complexité en temps

C'est le temps mis par un algorithme pour traiter un exemplaire.
C'est une fonction de la taille d'un exemplaire.

Complexité en espace

C'est l'espace mémoire utilisé par un algorithme pour traiter un exemplaire en plus de celui de l'exemplaire lui-même.
C'est une fonction de la taille d'un exemplaire.

On distinguera complexité dans le pire des cas, dans le meilleur des cas et en moyenne.

Lorsqu'on parle simplement de complexité on fait fait généralement référence à la complexité dans le pire des cas.

Analyse des algorithmes de tri

- les tris envisagés :
  - bulle
  - selection
  - insertion, avec différentes manières d'insérer
  - fusion
  - quicksort
- protocole de test :
  - sur des tableaux de taille 1 à 100
  - sur des tableaux croissants
  - sur des tableaux décroissants
  - sur des tableaux aléatoires : moyenne réalisée sur 1000 échantillons

II Faisons le point

d'un point de vue pratique
- choisir l'opération à compter
- compter dans les boucles
- établir des équations de récurrence
- distinguer le pire des cas et le meilleur des cas

d'un point de vue théorique
- la notion de complexité

Le tri bulle

- principe
- code :

```
1  let tri_bulle t =
2    let t' = Array.copy t
3    and n = Array.length t
4    in
5    for i = 2 to n do
6      for j = 0 to n-i do
7        nb_cmp := !nb_cmp + 1;
8        if t'.(j) > t'.(j+1) then
9          (* echange des valeurs aux positions j et j + 1 *)
10         let aux = t'.(j) in
11         t'.(j) <- t'.(j+1) ; t'.(j+1) <- aux
12     done;
13   done;
14   t'
```

- estimation du nombre d'opérations de comparaison
- décompte exact

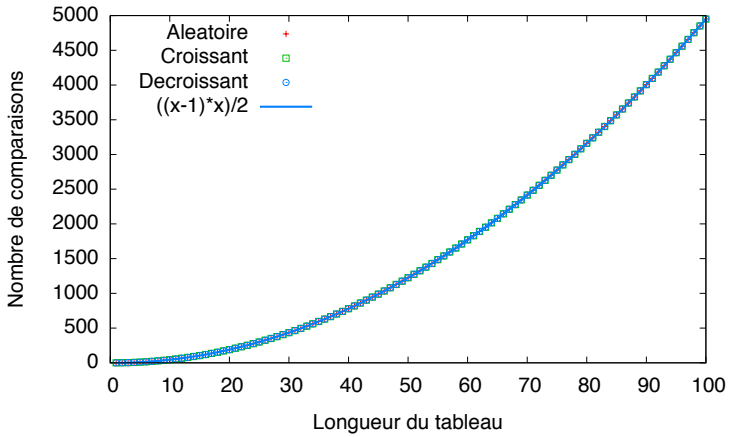
Détermination expérimentale

```
pour l allant de 1 à 100 faire
  creer un tableau croissant de taille l, le trier
  imprimer le nombre de comparaisons, raz du nb. comparaisons
  creer un tableau decroissant de taille l, le trier
  imprimer le nombre de comparaisons, raz du nb. comparaisons
  pour i allant de 1 à 1000 faire
    creer un tableau aleatoire de taille l, le trier
    sauvegarder le nombre de comparaisons
  fin pour
  imprimer le nombre de comparaisons
fin pour
```

```
...
10 45.000000 45.000000 45.000000
11 55.000000 55.000000 55.000000
12 66.000000 66.000000 66.000000
13 78.000000 78.000000 78.000000
...
```

```
gnuplot 'mydata.txt' using 1:2 title 'Croissant', \
        '' using 1:3 title 'Decroissant', \
        '' using 1:4 title 'Aleatoire'
```


Tri bulle - analyse



Le tri sélection

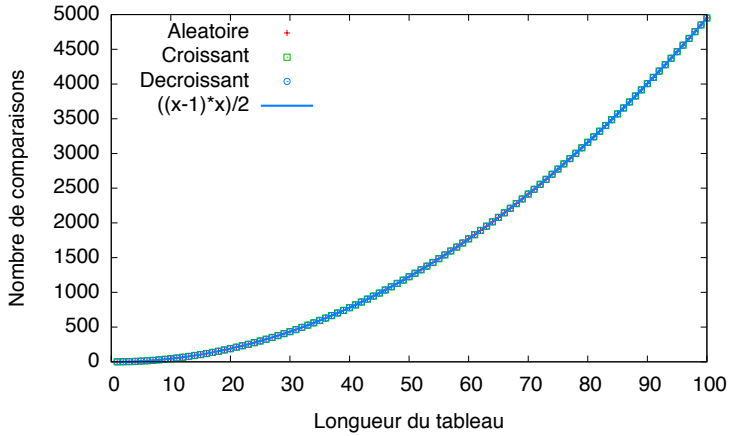
- principe, exemple
- code

```
1  let tri_selection t =
2    let t' = Array.copy t
3    and n = Array.length t in
4    for i = 0 to n - 2 do
5      (* recherche du plus petit element de la tranche t'.(i..n-1) *)
6      let ind_min = indice_minimum t' i (n - 1) in
7      (* echanger t'.(i) et t'.(ind_min) si necessaire *)
8      if i <> ind_min then begin
9        let aux = t'.(i) in
10       t'.(i) <- t'.(ind_min) ; t'.(ind_min) <- aux
11     end ;
12   done ;
13   t'
```

- estimation du nombre d'opérations de comparaison
- décompte exact


```
1  let indice_minimum t a b =
2    let ind_min = ref a in
3    for i = a + 1 to b do
4      nb_cmp := !nb_cmp + 1;
5      if t.(i) < t.(!ind_min) then begin
6        ind_min := i
7      end;
8    done ;
9    !ind_min
```

Tri sélection - analyse




```
1  let inserer t i =
2    let k = ref i
3    and aux = t.(i) in
4    (* on insere t.(i) dans la tranche t.(0..i-1) *)
5    while (!k >= 1) && (aux < t.(!k - 1)) do
6      nb_cmp := !nb_cmp + 1;
7      t.(!k) <- t.(!k - 1) ;
8      k := !k - 1
9    done ;
10   if (!k >= 1) then nb_cmp := !nb_cmp + 1;
11   t.(!k) <- aux
```

il ne faut rien oublier de compter

Le tri insertion

- principe, exemple
- code

```
1  let tri_insertion t =
2    let t' = Array.copy t
3    and n = Array.length t in
4    for i = 1 to n - 1 do
5      inserer t' i
6    done ;
7    t'
```

- décompte exact

Pour éviter cet écueil : on surcharge l'opérateur de comparaison pour que le comptage se fasse à chaque appel à la fonction de comparaison

```
1  let cmp a b =
2    nb_cmp := !nb_cmp + 1;
3    a < b
4
5  let inserer t i =
6    let k = ref i
7    and aux = t.(i) in
8    (* on insere t.(i) dans la tranche t.(0..i-1) *)
9    while (!k >= 1) && (cmp aux t.(!k - 1)) do
10     t.(!k) <- t.(!k - 1) ;
11     k := !k - 1
12   done ;
13   t.(!k) <- aux
```

Tri insertion - analyse