Préliminaires

- un algorithme est une suite finie d'instructions qui résoud un problème
- un algorithme doit fonctionner sur tous les exemplaires du problème qu'il résoud
- la taille de l'exemplaire est formellement le nombre de bits mais on pourra réduire cette définition au nombre de composantes (cases d'un tableau)
- l'ensemble des exemplaires est le domaine de définition

Présence d'un élément dans un tableau (v2)

```
1  """
2  t est un NumPy array d'Element
3  """
4  def contains(t,v):
5    found = False
6    i = 0
7    while i < len(t) and not found:
8      if t[i] == v:
9        found = True
10     i = i + 1
11   return found
```

- le nombre de tests d'égalité d'`Element` depend-il
  - de la taille de l'exemplaire ?
  - de la ≪ forme ≫ de l'exemplaire ?

Présence d'un élément dans un tableau (v3)

```
1  # version recursive
2  def contains (t,v):
3    n = len(t)
4    if n == 0:
5      return False
6    else:
7      tt = t[1:n] # que se passe-t-il lors de cette instruction ?
8      return t[0] == v or contains(tt,v)
```

- calcul du nombre de comparaisons
- discussion de l'espace supplémentaire utilisé
- comment évaluer les algorithmes récursifs ?

Présence d'un élément dans un tableau (v4)

dans le cas où `t[i:j]` implique une recopie, on peut faire mieux en espace que la v3, on utilise une sous-fonction qui va gérer les bornes du tableau

```
1  # version recursive avec espace memoire constant
2  # p = position dans le tableau
3  def aux (t,p,v):
4    if p == len(t):
5      return False
6    else:
7      return t[p] == v or aux(t,p+1,v)
8
9  def contains_v4 (t,v):
10   return aux(t,0,v)
```

Résumé (1/3)

pour évaluer la complexité des algorithmes :

- on comptera les opérations d'intérêt vis-à-vis du type de données manipulées (ici les `Element`)

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

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, en cas de copie de tableau, consomme plus de mémoire que v1 et v2)
- ces décomptes 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 d'usage possibles 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 généralement référence à la complexité dans le pire des cas.

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

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

Le tri bulle

- principe
- code :

```
1  def bubble_sort(t):
2    global nb_cmp
3    n = len(t)
4    for i in range (2,n+1):
5      for j in range (0,n-i+1):
6        nb_cmp = nb_cmp + 1
7        if t[j] > t[j+1]:
8          # echange des valeurs aux positions j et j + 1
9          aux = t[j]
10         t[j] = t[j+1]
11         t[j+1] = aux
12   return 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