

## TD2 : complexité des algorithmes itératifs

### 1 Complexité des boucles

#### 1.1 Intuition

Cette partie du TD vise à vous donner une intuition des complexités des différentes boucles. Il ne vous est donc pour l'instant pas demandé de donner de formule générale pour ce type de boucles, mais simplement d'expérimenter pour vous donner une idée de leur complexité. Pour chaque algorithme proposé ci-dessous, dessinez l'affichage qu'il produira pour  $n = 1, 3, 5$ . Comptez ensuite le nombre de caractères \*.

**données :**  $n$  : un entier

**Algorithme**

```
┌ pour  $i$  allant de 1 à  $n$  faire
└   ┌ Afficher(*)
```

**données :**  $n$  : un entier

**Algorithme**

```
┌ pour  $i$  allant de 1 à  $n$  faire
└   ┌ pour  $j$  allant de 1 à 3 faire
      └   ┌ Afficher(*)
          └   Afficher(retour à la ligne)
```

**données :**  $n$  : un entier

**Algorithme**

```
┌ pour  $i$  allant de 1 à  $n$  faire
└   ┌ pour  $j$  allant de 1 à  $n$  faire
      └   ┌ Afficher(*)
          └   Afficher(retour à la ligne)
```

**données :**  $n$  : un entier

**Algorithme**

```
┌ pour  $i$  allant de 1 à  $n$  faire
└   ┌ pour  $j$  allant de 1 à  $i$  faire
      └   ┌ Afficher(*)
          └   Afficher(retour à la ligne)
```

**données :**  $n$  : un entier

**Algorithme**

```
┌ pour  $i$  allant de 1 à  $n$  faire
└   ┌ pour  $j$  allant de 1 à  $n$  faire
      └   ┌ pour  $k$  allant de 1 à  $n$  faire
            └   ┌ Afficher(*)
                └   Afficher(retour à la ligne)
          └   Afficher(retour à la ligne)
        └   Afficher(retour à la ligne)
```

**données :**  $n$  : un entier

**Algorithme**

```
┌ pour  $i$  allant de 1 à  $n$  faire
└   ┌ pour  $j$  allant de 1 à  $n$  faire
      └   ┌ pour  $k$  allant de 1 à  $j$  faire
            └   ┌ Afficher(*)
                └   Afficher(retour à la ligne)
          └   Afficher(retour à la ligne)
        └   Afficher(retour à la ligne)
```

**données :**  $n$  : un entier

**Algorithme**

```
┌ pour  $i$  allant de 1 à  $n$  faire
└   ┌ pour  $j$  allant de 1 à  $i$  faire
      └   ┌ pour  $k$  allant de 1 à  $j$  faire
            └   ┌ Afficher(*)
                └   Afficher(retour à la ligne)
          └   Afficher(retour à la ligne)
        └   Afficher(retour à la ligne)
```

**données :**  $n$  : un entier

**Algorithme**

```
┌ pour  $i$  allant de 1 à  $n$  faire
└   ┌ pour  $j$  allant de 1 à  $i$  faire
      └   ┌ Afficher(*)
          └   pour  $k$  allant de 1 à  $n$  faire
                └   Afficher(*)
          └   Afficher(retour à la ligne)
```

## 1.2 Formalisation, dénombrement des opérations

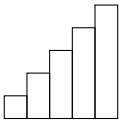
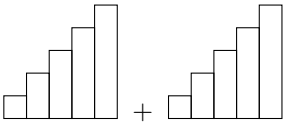
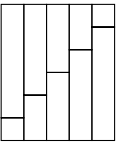
Écrivez à l'aide de sommes ( $\sum$ ) les formules permettant de calculer le nombre d'étoiles  $E$  comptées dans l'exercice précédent en fonction du paramètre  $n$ . Le but n'est pas ici de réduire les formules et de les simplifier, mais déjà de savoir les poser proprement.

## 1.3 Éléments de simplification des sommes

Pour chaque formule élaborée précédemment, simplifiez la pour obtenir une expression en fonction de  $n$  sans sommes. Vous aurez besoin pour cela de savoir réduire :

$$\sum_{i=1}^n a \quad , \quad \sum_{i=1}^n i \quad \text{et} \quad \sum_{i=1}^n i^2$$

où  $a$  ne dépend pas de  $n$ . Les deux premières sont considérées comme connues. Si vous ne vous en souvenez pas, vous pouvez essayer de compter sur vos doigts pour retrouver la première. Pour la seconde, un indice :

si  $\square = 1$ , alors  $\sum_{i=1}^5 i =$   , et  $2 \sum_{i=1}^5 i =$    $=$   .

Pour la troisième, vous pourrez utiliser la formule :

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}.$$

Pensez à vérifier ensuite vos formules avec les décomptes réalisés précédemment.

## 2 Analyse d'algorithmes itératifs, pire et meilleur cas

Les algorithmes ont souvent un comportement qui dépend des données fournies en entrée. Il arrive donc souvent que le nombre d'opération réalisées dépende de l'organisation des données. Un outil fondamental de l'analyse de complexité consiste à déterminer le meilleur et le pire cas, afin de situer notre algorithme dans une fourchette. Il ne s'agit pas d'expliquer que pour un tableau d'une seule case, un algorithme de tri sera plus rapide que pour un tableau plus grand. Ça, tout le monde s'en doute. Mais pour une taille donnée, l'organisation des valeurs dans le tableau peut avoir une influence. Nous allons donc étudier deux algorithmes pour nous en convaincre.

### 2.1 Tri à bulles

Un premier algorithme de tri célèbre est le tri à bulles. Il consiste à chaque itération à amener le maximum de la portion de tableau restant à trier à la fin de cette portion. Les « bulles » sont donc les maximums successifs qui remontent dans le tableau. Au début, c'est tout le tableau qu'il fait trier, et à chaque itération, le maximum remonté est à sa place, et la portion de tableau restant à trier est réduite d'une case. Il s'écrit ainsi :

#### Procédure TriBulles

**données :** tab : un tableau,  $n$  sa taille

**résultat :** tab est trié

#### Algorithme

```

pour  $i$  allant de 1 à  $n - 1$  faire
  pour  $j$  allant de 0 à  $n - i - 1$  faire
    si  $tab[j] > tab[j + 1]$  alors
      échanger  $tab[j]$  et  $tab[j + 1]$ 
  
```

Étudiez le meilleur et le pire cas pour :

- le nombre de comparaisons entre éléments du tableau
- le nombre d'échanges d'éléments du tableau

## 2.2 Tri par insertion

Le tri par insertion consiste à trier le tableau en considérant que le début du tableau est trié et en insérant petit à petit les valeurs suivantes à leur place. Initialement, la portion triée du tableau est réduite à la seule première case.

### Procédure Trilnsersion

**données :**  $tab$  : un tableau,  $n$  sa taille

**résultat :**  $tab$  est trié

#### Algorithme

**pour**  $i$  allant de 1 à  $n - 1$  **faire**

$j \leftarrow i$

**tant que**  $j > 0$  et  $tab[j] < tab[j - 1]$  **faire**

        échanger  $tab[j]$  et  $tab[j - 1]$

$j \leftarrow j - 1$

Étudiez le meilleur et le pire cas pour :

- le nombre de comparaisons entre éléments du tableau
- le nombre d'échanges d'éléments du tableau

## 2.3 Étude en moyenne (si le temps le permet)

Le tri par insertion vu dans l'exercice précédent n'a pas la même complexité dans le meilleur et le pire cas. Il est donc utile de savoir en général, sur un tableau quelconque, si la complexité s'approche plus du meilleur ou du pire cas. Une approche consiste à calculer la moyenne de la complexité sur toutes les entrées possibles (de taille  $n$ ). On parle ainsi d'étude en moyenne. Ce genre d'étude peut s'avérer complexe, mais est abordable dans le cas du tri par insertion.

### 2.3.1 Position moyenne du dernier élément dans une permutation aléatoire

Considérons un ordre quelconque du tableau : chaque élément du tableau a la même probabilité de se situer dans chacune des cases du tableau. Sur un tableau de  $n$  cases, en imaginant que les  $n - 1$  premières cases ont été triées, quelle est la probabilité que le  $n^{\text{ième}}$  élément soit plus petit que le premier ? Entre le premier et le second ? À sa place ? Quelle est donc sa position moyenne au sein des éléments précédents ?

### 2.3.2 Comportement en moyenne du tri par insertion

Quel est donc en moyenne le nombre d'itérations réalisées par la boucle « tant que » ? Qu'en déduisez vous sur la complexité totale (pour le nombre d'échanges) de l'algorithme de tri par insertion en moyenne ?