Complexité

1 Introduction

Exercice 1. Classement d'algorithmes.

L'objectif de cet exercice est d'introduire la notion de complexité (qui n'a rien à voir avec la difficulté de réalisation d'un code) et les comparaisons d'algorithmes. Classez les algorithmes suivants par temps d'exécution croissant. Inutile de détailler les calculs, l'intuition suffit.

1) addition de 100 entiers,
2) produit de 100 entiers,
3) moyenne des pixels d'une image carrée de 100 pixels,
4) tri d'un tableau de 100 cases,
5) recherche du plus grand élément dans un tableau de 100 cases,

Correction. Sans nécessairement classer ces algorithmes un par un, il semble possible de les regrouper. L'addition et la recherche semblent être de temps d'exécution similaire puisqu'il faut parcourir une seule fois les éléments en entrées (coefficient du polynôme ou cases du tableau). Le tri nécessite de parcourir plusieurs fois le tableau. On ne sait pas encore combien, mais il semble certain que le temps d'exécution est plus long que les deux précédents, mais plus court que le reste. La puissance étant une série de produit, elle semble logiquement plus longue que la multiplication. On peut donc proposer l'ordre suivant pour un temps d'exécution croissant :

1. addition ≃ recherche du max,
2. tri,
3. produit,
4. puissance,

Exercice 2. Algorithme VS code.

Considérons l'algorithme et le code suivants.

Algorithme 1 – Inversion de tableau
Entrée. Tableau d'entiers Tab de taille N
Sortie. Tableau d'entiers Tab inversé de taille N

```
Entier milieu ← ⌊N/2⌋ − 1
Pour cpt allant de 0 à milieu faire
  Echanger (Tab[cpt], Tab[N-cpt-1])
Fin Pour
Renvoyer Tab
```

```c
void inverseTab (int *Tab, int N)
{
  int i, tmp, milieu = N/2;
  for (i=0; i<milieu; i++)
  {
    tmp = Tab[i];
    Tab[i] = Tab[N-i-1];
    Tab[N-i-1] = tmp;
  }
```

1) Listez les différentes opérations présentes dans l'algorithme.
2) Pour chacune des opérations identifiées, comptez le nombre de fois où cette opération est réalisée au cours d'un exécution de l'algorithme.
3) Mêmes questions pour le code C.

Exercice 3. Comparaison d'algorithmes.
Considérons les trois algorithmes suivants dont l'entrée est un entier `N` et un tableau `Tab` de `N` cases. La sortie est également la même pour chaque algorithme : le tableau `Tab` dont les éléments ont été modifiés.

Algorithme 2 – Génération aléatoire

```
Tableau Tab de taille N
Pour cpt allant de 0 à N-1 faire
| Tab[cpt] ← rand(0..N-1)
Fin Pour
Renvoyer Tab
```

Algorithme 3 – Somme de cases

```
Tableau Tab de taille N
Entier tmp
Pour cpt allant de 0 à N-1 faire
| tmp ← 0
| Pour i allant de 0 à cpt faire
| | tmp ← tmp + i
| Fin Pour
| Tab[cpt] ← tmp
Fin Pour
Renvoyer Tab
```

Algorithme 4 – Génération

```
Tableau Tab de taille N
Pour cpt allant de 0 à 1000000 faire
| Tab[cpt % N] ← cpt
Fin Pour
Renvoyer Tab
```

Évaluez le nombre d'opérations réalisées dans chaque algorithme afin de répondre aux questions suivantes.

1) Pour un tableau de taille 10, quel algorithme mettra le plus de temps à s'exécuter ?
2) Même question pour un tableau de mille éléments ? Pour un million d'éléments ? Pour un milliard d'éléments ?

2 Étude de complexité

Exercice 4. Tracé de complexités.
A l'aide d'un tableur (Libre Office Calc, Excel ou autre), affichez les graphes de fonctions correspondant au complexités mentionnées dans le tableau du cours. Adaptez l'échelle si les courbes ne se distinguent pas.

Exercice 5. Notation de Landau.
Complétez le tableau suivant donnant des équivalences entre un exemple de complexité réelle, sa notation de Landau et son type de complexité.

Type	Landau	Exemple
...	...	$17N + 100$
...	$O(N^3)$	...
exponentielle	...	...
...	...	$12N(4 + \log(N)) + 7$
...	$O(2^N)$	...
factorielle	...	...

Exercice 6. Étude de complexité.
Pour chacun des programmes suivants, vous devez rédiger l'algorithme et donner sa complexité dans le meilleur et dans le pire des cas. Donnez également sa complexité en notation de Landau ainsi que son appellation. Vous pourrez vérifier votre résultat en modifiant vos programmes pour y ajouter un compteur d'opération.

1) Échange de deux variables
2) Calcul de la moyenne des éléments d'un tableau de N cases.
3) Recherche d'un élément dans un tableau.
4) Calcul du n-ième terme de Fibonacci.
5) Jeu du plus ou moins.

Exercice 7. Complexité des tris.
De la même façon que dans l'exemple du cours, déterminez la complexité dans le pire des cas du tri à bulles optimisé, du tri sélection et du tri insertion.

3 Exercices d'approfondissement

Exercice 8. Calcul de complexité.
Calculez la complexité dans le meilleur des cas, dans le pire et la complexité moyenne de l'algorithme suivant.

Algorithme 5 – Génération de tableau
Entrée. Entier N
Sortie. Tableau d'entiers Tab de taille N

```
Tableau Tab de taille N
Pour cpt allant de 0 à N-1 faire
    Faire
        Tab[cpt] ← rand(0..N-1)
    Tant que Tab[cpt] ≠ cpt
Fin Pour
Renvoyer Tab
```

Exercice 9. Calcul infaisable.
Actuellement, aucun ordinateur ou super calculateur ne peut réaliser dans un temps acceptable pour un humain plus de 2^{80} opérations. Pour chacun des types de complexité vus en cours, indiquez quelle taille de données permet d'atteindre ce nombre d'opérations.

Exercice 10. Ca tourne !

On considère qu'un ordinateur peut actuellement réaliser trois milliards d'opérations à la seconde. Pour chacune des complexités vues en cours, indiquez quelle doit être la taille des données en entrée d'un algorithme pour faire tourner un ordinateur une heure.