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
```

Correction

```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.

Correction.

Opération	Algo	Code
Affectation	N/2	2+ 2N
Division	1	1
Soustraction	N-1	2N
Addition	0	N/2
Partie entière	1	1
Échange	N/2-1	0
Accès tableau	N-2	2N
Comparaisons	0	N/2

On a l'impression que le code C comporte beaucoup plus d'opérations que l'algorithme. En fait, ce dernier les dissimule. La boucle est un exemple parlant : on ne voit pas l'incrémentation du compteur ni le test de fin pour ce dernier.

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
```

Correction

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 ?

Correction. La tableau suivant compte le nombre d'opérations effectuées dans chaque algorithme et évalue le nombre total d'opérations lorsque N vaut 10, 100 ... Le nombre d'opérations calculées pour une boucle et le nombre d'opération réalisées dans la boucle plus celles permettant au compteur d'évoluer, soit 3 opérations supplémentaire par tour (incrément = addition + affectation et comparaison). Les notations expriment des ordres de grandeur.

Algo	Nb opérations	$N = 10$	$N = 100$	$N = 1000$	$N = 10^6$	$N = 10^9$
1	$6N$	60	600	6000	6.10^6	6.10^9
2	$\frac{5}{2}(N^2 + N) + 6N$	335	25850	$2,5.10^6$	$2,5.10^{12}$	$2,5.10^{18}$
3	6000000	6.10^6	6.10^6	6.10^6	6.10^6	6.10^6

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.

Correction

Type	Landau	Exemple
linéaire	$O(N)$	$17N + 100$
cubique	$O(N^3)$	$7N(N^2 + 5)$
exponentielle	$O(2^N)$	$N^2.2^N$
NlogN ou quasi-linéaire	$O(NlogN)$	$12N(4 + \log(N)) + 7$
exponentielle	$O(2^N)$	$2^{N^3+7}\ldots$
factorielle	$O(N!)$	$7N!$

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

Correction. La complexité est en $O(1)$, soit constante, dans tous les cas.

2) Calcul de la moyenne des éléments d'un tableau de N cases.

Correction. On doit parcourir le tableau une seule fois et calculer la somme au fur et à mesure. La complexité est donc linéaire en $O(N)$.

3) Recherche d'un élément dans un tableau.

Correction. On doit parcourir le tableau une seule fois. La complexité est donc linéaire en $O(N)$.

4) Calcul du n-ième terme de Fibonacci.

Correction. L'algorithme itératif utilise la somme des deux termes précédents pour calculer le nouveau. La complexité est donc linéaire en $O(n)$.

5) Jeu du plus ou moins.

Correction. La complexité dépend de l'algorithme de résolution. L'algorithme naïf tente toutes les possibilités entre 0 et le maximum N. La complexité est alors linéaire. Une version optimisée coupe le problème en deux à chaque tentative et produit une complexité logarithmique en $O(\log(N))$.

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.

Correction

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.

Correction. On cherche le plus petit N tel que $f(N) > 2^{80}$

Complexité (f)	Taille des entrées (N)
$O(1)$	Pas de N
$O(\log_2(N))$	$2^{2^{80}}$
$O(N)$	2^{80}
$O(N\log(N))$	2^{74}
$O(N^2)$	2^{40}
$O(2^N)$	80
$O(N!)$	25

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.

Correction. Il y a 3600 secondes en une heure. Il faut donc que l'ordinateur réalise $3600 \times 3.10^9 = 1,08.10^{13} \equiv 2^{44}$ opérations. La tableau suivant donne la taille des entrées pour atteindre ce nombre d'opérations.

Complexité	Taille des entrées
$O(1)$	Pas de N
$O(\log_2(N))$	$2^{2^{44}}$ soit environ $10^{5277655813325}$ entrées
$O(N)$	2^{44} soit environ $1,8.10^{13}$ entrées
$O(N\log(N))$	2^{38} soit environ 275 milliards d'entrées
$O(N^2)$	2^{22} soit environ 4,2 millions d'entrées
$O(2^N)$	44
$O(N!)$	16

Correction