TD1: Complexité

Préambule

L'objectif de cette séance est de s'exercer aux compétences suivantes :

- lire un algorithme (itératif ou récursif) en pseudo-code
- exécuter à la main un algorithme (itératif ou récursif) sur un exemple jouet
- comparer l'efficacité de plusieurs algorithmes pour résoudre un même problème
- évaluer la complexité de temps de calcul d'algorithmes élémentaires
- évaluer la complexité de temps de calcul d'algorithmes complexes

Exercice 1. Trier ces fonctions par ordre croissant de complexité asymptotique :

a. $\Theta(n)$
b. $\Theta(n \log n)$
c. $\Theta(n \log \log n)$
d. $\Theta(n^{1.0001})$
e. $\Theta\big(n\,(\log n)^{9999}\big)$
f. $\Theta\big(2^{n/3}\big)$
g. $\Theta(1.9^n)$
h. $\Theta(n/\log n)$

Exercice 2. Supposez que chacune des expressions ci-dessous donne le temps d'exécution $T(n)$ d'un algorithme pour résoudre un problème de taille n.

Sélectionnez le(s) terme(s) dominant(s) ayant la plus forte augmentation de n et indiquez la complexité Grand-Theta de chaque algorithme.

Expressions	Termes dominants	$O(\ldots)$
$5 + 0.001n^3 + 0.025n$		
$500n + 100n^{1.5} + 50n \log(10n)$		
$0.3n + 5n^{1.5} + 2.5n^{1.75}$		
$n^2 \log_2 n + n(\log_2 n)^2$		
$n \log_3 n + n \log_2 n$		
$3 \log_8 n + \log_2 \log_2 \log_2 n$		
$100n + 0.01n^2$		
$0.01n + 100n^2$		
$2n + n^{0.5} + 0.5n^{1.25}$		
$0.01n \log_2 n + n(\log_2 n)^2$		
$100n \log_3 n + n^3 + 100n$		
$0.003 \log_4 n + \log_2 \log_2 n$		

Exercice 3. Comparez l'ordre de croissance asymptotique des paires de fonctions suivantes. Dans chaque cas dites si $f(n) \in \Theta(g(n))$, $f(n) \in O(g(n))$ ou $f(n) \in \Omega(g(n))$.

a. $f(n) = 100n + \log n$ et $g(n) = n + (\log n)^2$

b. $f(n) = \log n$ et $g(n) = \log(n^2)$

c. $f(n) = n^2/\log n$ et $g(n) = n(\log n)^2$

d. $f(n) = \sqrt{n}$ et $g(n) = (\log n)^5$

e. $f(n) = n2^n$ et $g(n) = 3^n$

Exercice 4. Etant donné un tableau T de n entiers, le minimum de T peut être défini formellement comme l'entier m qui satisfait les deux propositions suivantes :

i. il existe i dans $0, \ldots, n-1$ tel que $T[i] = m$; et

ii. pour tout j dans $0, \ldots, n-1$, nous avons $m \leq T[j]$

Un algorithme naïf (force brute) pour calculer le minimum d'un tableau est donc le suivant :

Algorithme 1 MinNaïf

Données: un tableau T de n entiers
Résultat: le minimum de T

```
1: for i de 0 à n − 1 do
2:     m ← T[i] ; cond ← true
3:     for j de 0 à n − 1 do
4:         if m > T[j] then cond ← false
5:     end for
6:     if cond = true then return m
7: end for
```

1. Évaluez la complexité de temps de calcul (dans le pire cas) de chaque ligne de l'algorithme MinNaïf. Déduisez-en la complexité de temps de calcul global de l'algorithme.
2. Considérons maintenant que la deuxième boucle commence à $j = i$. Quelle est la différence dans le résultat de l'algorithme ?
3. Considérez-vous que la modification proposée est significative ?

Si vous avez déjà eu l'occasion d'implémenter la recherche du minimum dans un tableau, vous avez probablement proposé l'algorithme standard suivant.

Algorithme 2 MinStandard

Données: un tableau T de n entiers
Résultat: le minimum de T

```
1: m ← T[0]
2: for i de 0 à n − 1 do
3:     if T[i] < m then m ← T[i]
4: end for
5: return m
```

4. Évaluez la complexité de temps de calcul (dans le pire cas) de chaque ligne de l'algorithme MinStandard. Déduisez-en la complexité de temps de calcul global de l'algorithme.
5. Comparez la complexité avec l'algorithme précédent.

Exercice 5.

1. Évaluez la complexité de chaque ligne de l'algorithme de recherche séquentiel (cf ci-après) dans un tableau trié et déduisez la complexité globale de l'algorithme (dans le pire cas).

Algorithme 3 PosSeq

Données: un tableau T de n entiers triés, un entier x à rechercher
Résultat: le plus petit indice d'une valeur de T supérieur ou égale à x

```
1: i ← 0
2: while i < n et T[i] < x do
3:     i ← i + 1
4: end while
5: return i
```

2. Évaluez la complexité de chaque ligne de l'algorithme de recherche dichotomique (cf ci-après) dans un tableau trié et déduisez la complexité globale de l'algorithme.

Algorithme 4 RechercheDichotomique

Données: un tableau T de n entiers triés, un entier x à rechercher
Résultat: le plus petit indice d'une valeur de T supérieur ou égale à x

```
1: d ← −1; f ← n
2: while d + 1 < f do
3:     m ← (d + f) // 2                    // Division entière
4:     if T[m] < x then
5:         d ← m
6:     else
7:         f ← m
8:     end if
9: end while
10: return f
```

Exercice 6. Considérez l'algorithme suivant :

Algorithme 5 EstPalindrome

Données: une chaîne S de n caractères
Résultat: vrai si S est un palindrome, faux sinon

```
1: for i de 0 à n/2 do
2:     if S[i] ≠ S[n − 1 − i] then
3:         return false
4:     end if
5: end for
6: return true
```

1. Indiquez ce que retourne cet algorithme lorsque S vaut “abracadabra” et lorsque S vaut “esoperesteicietserepose”.
2. Évaluez la complexité de l’algorithme EstPalindrome.

Considérez l’algorithme suivant :

Algorithme 6 ContientPalindromeTailleFixe

Données: une chaîne S de n caractères, un entier $t < n$
Résultat: vrai si S contient un palindrome de taille t

```
1: for i de 0 à n − 1 − t do
2:     T ← S[i] . . . S[i + t − 1]
3:     if EstPalindrome(T) then
4:         return true
5:     end if
6: end for
7: return false
```

3. Indiquez ce que retourne ContientPalindromTailleFixe lorsque S vaut “abracadabra” et lorsque S vaut “esoperesteicietserepose” et que t vaut 3.
4. Évaluez la complexité de chacune des lignes de l’algorithme ContientPalindromTailleFixe et déduisez en la complexité globale de cet algorithme.

Considérez l’algorithme suivant :

Algorithme 7 PlusLongPalindrome

Données: une chaîne S de n caractères
Résultat: la longueur du plus long palindrome contenu dans S

```
1: t ← n
2: while ContientPalindromeTailleFixe(S,t) = false do
3:     t ← t − 1
4: end while
5: return t
```

5. Indiquez ce que retourne PlusLongPalindrome lorsque S vaut “reveries” et lorsque S vaut “esoperesteicietserepose”.
6. Évaluez la complexité de chacune des lignes de l’algorithme PlusLongPalindrome et déduisez en la complexité globale de cet algorithme.