Feuille d'exercices n°17 - Révisions diviser pour régner

Exercice 1 : Suites récurrentes de complexité

Dans cet exercice il vous est demandé de donner, pour chacune des suites suivantes :
- un Θ de son comportement asymptotique ;
- un programme OCaml **dont cette suite est la complexité**, pour des constantes a et b au choix♣
- l'arbre des appels récursifs.

$a)$ $\begin{cases} u_0 &= 1 \\ u_n &= u_{n-1} + a \quad a \in \mathbb{R}^{+\star} \end{cases}$

$b)$ $\begin{cases} u_0 &= 1 \\ u_n &= u_{n-1} + an \quad a \in \mathbb{R}^{+\star} \end{cases}$

$c)$ $\begin{cases} u_0 &= 1 \\ u_n &= au_{n-1} + b \quad a \in [2, +\infty[, b \in \mathbb{R}^{+\star} \end{cases}$

$d)$ $\begin{cases} u_0 &= 1 \\ u_n &= u_{n/2} + b \quad b \in \mathbb{R}^{+\star} \end{cases}$

$e)$ $\begin{cases} u_0 &= 1 \\ u_n &= u_{n/2} + bn \quad b \in \mathbb{R}^{+\star} \end{cases}$

$f)$ $\begin{cases} u_0 &= 1 \\ u_1 &= 1 \\ u_n &= u_{\lceil \frac{n}{2} \rceil} + u_{\lfloor \frac{n}{2} \rfloor} + b \quad b \in \mathbb{R}^{+\star} \end{cases}$

$g)$ $\begin{cases} u_0 &= 1 \\ u_n &= au_{n/2} + b \quad a \in [2, +\infty[, b \in \mathbb{R}^{+\star} \end{cases}$

Exercice 2 : Quelques équivalents classiques

Q. 1 Donner le comportement asymptotique des suites suivantes♡ au moyen de la notation Θ.

1. $\lfloor \log_2(n) \rfloor$
2. $2^{\lfloor \log_2(n) \rfloor}$

Q. 2 Donner le comportement asymptotique des sommes suivantes♠ au moyen de la notation Θ.

1. $\sum_{k=1}^{n} k$
2. $\sum_{k=1}^{n} k^2$
3. $\sum_{k=0}^{n} 2^k$
4. $\sum_{k=0}^{n} \frac{1}{2^k}$
5. $\sum_{k=0}^{n} 2^k$
6. $\sum_{k=1}^{n} \frac{1}{k}$
7. $\sum_{k=1}^{n} \log k$
8. $\sum_{k=0}^{n} k \log k$

♣. Insistons : il s'agit écrire un programme OCaml ayant cette suite comme complexité et non un programme OCaml calculant cette suite.

♡. *i.e.* implicitement indexées par n

♠. *i.e.* des suites, implicitement indexées par n, des sommes jusqu'au rang n

Exercice 3 : Sous-tableau de somme maximale

Étant donné un tableau d'entiers, le problème du sous-tableau maximum consiste à trouver un ensemble d'indices consécutifs (*i.e.* un intervalle d'indices) non vide qui maximise la somme des éléments du tableau. Par exemple, le sous-tableau maximum du tableau `[-3, 4, 5, -1, 2, 3, -6, 4]` est le tableau`[4, 5, -1, 2, 3]` de valeur 13. Étant donné un tableau T on note donc $\mathsf{stm}(T)$ la valeur du sous-tableau maximum.

Q. 1 Formaliser ce problème comme un problème d'optimisation.

Q. 2 Dans un premier temps, on considère l'algorithme de résolution ci-dessous. Quelle est la complexité de cet algorithme ?

Algorithme 1 : Sous tableau maximum

Entrée : A un tableau de taille n
Sortie : $\mathsf{stm}(A)$

```
1 A_max ← max A[1], ..., A[n] ;
2 pour i = 1 à n faire
3     pour j = i à n faire
4         sum ← 0 ;
5         pour k = i à j faire
6             sum ← sum + A[j] ;
7             si sum > A_max alors
8                 A_max ← sum
9 retourner A_max
```

Q. 3 Proposer une variante en $\Theta(n^2)$ de cet algorithme.

Dans la suite de l'exercice, on s'intéresse à des algorithmes de type diviser pour régner. Pour un tableau A indicé par $[1..n]$ avec $n > 1$, on note $A_1 = A[1..m]$ et $A_2 = A[m+1..n]$ les deux sous-tableaux définis par $m = \left\lfloor \frac{n+1}{2} \right\rfloor$. On note alors $\mathsf{stm}_1(A) = \mathsf{stm}(A_1)$ et $\mathsf{stm}_2(A) = \mathsf{stm}(A_2)$. On introduit aussi $\mathsf{stm}_3(A)$ la valeur maximum d'un sous-tableau de A qui couvre à la fois les indices m et $m+1$.

Q. 4 Comment calculer $\mathsf{stm}_3(A)$ efficacement ? Quelle est la complexité en fonction de n la taille de A ?

Q. 5 Connaissant $\mathsf{stm}_1(A)$, $\mathsf{stm}_2(A)$ et $\mathsf{stm}_3(A)$, quelle est la valeur de $\mathsf{stm}(A)$? Dans le cadre d'une méthode diviser pour régner qui calcule $\mathsf{stm}(A)$ en calculant récursivement et $\mathsf{stm}_1(A)$ et $\mathsf{stm}_2(A)$ en quoi consisterait l'opération de fusion ? Quelle serait la complexité de cette opération ?

Q. 6 Soit T_n le nombre d'opérations élémentaires (additions et comparaisons d'éléments du tableau) que réalise cette méthode pour un tableau de taille n. Donner l'équation de récurrence satisfaite par T_n. En déduire la complexité de la méthode.

On s'intéresse maintenant à une autre approche diviser pour régner afin de réduire encore la complexité. Pour ce faire, on définit, pour un tableau A indicé par $[1..n]$ les valeurs suivantes :

- $\mathsf{pref}(A) = \max\{\sum_{k=1}^{i} A[k] | i \in [1..n]\}$
- $\mathsf{suff}(A) = \max\{\sum_{k=i}^{n} A[k] | i \in [1..n]\}$

- $\mathsf{tota}(A) = \sum_{k=1}^{n} A[k]$

Autrement dit, $\mathsf{pref}(A)$ (resp. $\mathsf{suff}(A)$) est la valeur maximum d'un sous-tableau préfixe (resp. suffixe) de A, et $\mathsf{tota}(A)$ est la somme des valeurs de A.

Q. 7 Expliquer comment calculer ces valeurs pour un tableau A ?

Q. 8 Quelle est la complexité de l'algorithme diviser pour régner associé ? Justifier.

Exercice 4 : Éléments majoritaires

On dit qu'un élément x est majoritaire dans un multi-ensemble E de cardinal n lorsque le nombre d'occurrences de x dans E est strictement supérieur à $\frac{n}{2}$. On suppose donc un multi-ensemble E représenté par un tableau de ses éléments et on cherche si oui ou non E admet un élément majoritaire (s'il existe il est unique). Dans le cas d'une réponse affirmative on renverra l'élément en question. On s'intéresse dans cet exercice au nombre de tests d'égalité (ou inégalité évidemment) effectués entre des éléments du multi-ensemble E. On suppose définie une fonction nbOccur prenant en arguments un tableau T, un élément x, et deux indices i et j du tableau et retournant le nombre d'occurrences de x dans le tableau T entre les indices i et j (au sens large). On suppose de plus que l'appel $\mathsf{nbOccur}(T, x, i, j)$ fait $j - i + 1$ comparaisons. ♣

Q. 1 Rappeler la définition d'un algorithme naïf $\mathsf{majoritaire}(T)$ retournant s'il existe un élément majoritaire du multi-ensemble E représenté par le tableau T. On précisera la complexité pire cas de l'algorithme.

Q. 2 Supposant connu l'élément majoritaire du sous-tableau $T[0..n/2-1]$ ♡ et l'élément majoritaire du sous-tableau $T[n/2..n-1]$ où T est un tableau de taille n, donner l'élément majoritaire de T en faisant de l'ordre de n comparaisons.

Q. 3 Donner alors un algorithme du paradigme diviser pour régner permettant la résolution du problème de la recherche d'un élément majoritaire.

Q. 4 Donner la complexité pire cas dans le cas où la taille du tableau donné en entrée est une puissance de 2.

Exercice 5 : Chercher une pièce dans un lot de pièces

On cherche à retrouver une pièce défectueuse parmi n pièces d'aspects identiques. La pièce défectueuse n'est identifiable que par sa masse, en effet elle est plus légère que les autres. On sait qu'il n'y a qu'une seule pièce défectueuse. On dispose d'une balance de type Roberval, c'est-à-dire une balance à plateaux qui permet de comparer les masses de deux ensembles d'objets disjoints (en effet on ne peut pas mettre le même objet des deux côtés de la balance à la fois...).

Q. 1 On considère deux ensembles disjoints de pièces A et B. Quelles sont les issues possibles d'une comparaison de A et B grâce à la balance Roberval ? En supposant que A et B sont de même cardinal, que peut on déduire concernant la pièce défectueuse dans chacun des cas.

♣. Les suppositions des deux phrases précédentes sont là pour vous faire gagner du temps, il **faut** que vous sachiez implémenter un tel algorithme nbOccur.

♡. comprendre : on sait s'il existe un élément majoritaire et s'il existe, on sait qui il est

Q. 2 En utilisant les remarques faites à la question précédentes, proposer un algorithme efficace pour retrouver la pièce défectueuse. *On attend une explication ou un pseudo-code de l'algorithme.*

Q. 3 Quelle est la complexité temporelle de l'algorithme proposé ? Justifier.

Exercice 6 : Multiplication d'entiers par algorithme de Karatsuba

Dans cet exercice on s'intéresse au problème de la multiplication de grands entiers machines. On suppose les entiers représentés par la séquence indicée des bits de leur décomposition en base 2. Par exemple, l'entier 14 est représenté par la séquence $(1, 1, 1, 0)$. On appellera alors taille d'un entier la longueur d'une telle séquence. On suppose de telles séquences stockées dans des tableaux permettant l'indexation en temps constant. On remarque que les multiplications et divisions entières par des puissances de 2 correspondent à des décalages de bits. En effet si $(u_{n-1}, u_{n-2}, \dots, u_0)$ est la représentation d'un entier u et $p \in \mathbb{N}$ alors $u/2^p$ est représenté par $(u_{n-p-1}, u_{n-p-2}, \dots, u_p)$ et $u \times 2^p$ par $(u_{n-1}, u_{n-2}, \dots, u_0, \underbrace{0, 0, \dots 0}_{p})$. Dans tout l'exercice on s'intéresse uniquement au problème de la multiplication de deux entiers de même taille et on mesure la complexité au regard du nombre de multiplications de nombres à 1 bit. Ainsi l'opération 1×0 est considérée comme une opération atomique de coût 1 alors que le coût de calcul de 1110110×101110 dépend de l'algorithme choisi pour faire le calcul de $\times$.

Q. 1 Expliquer en quoi l'hypothèse d'une taille commune des deux opérandes n'est pas trop simplificatrice.

Q. 2 Rappeler l'algorithme de multiplications naïf d'entiers (celui appris en primaire). Quelle est sa complexité ?

On remarque que si $u = u_a + 2^p u_b$ avec $u_a < 2^p$ et $v = v_a + 2^p v_b$ avec $v_a < 2^p$ alors $u \times v = u_a \times v_a + 2^p(u_b \times v_a + v_b \times u_a) + 2^{2p} u_b \times v_b$.

Q. 3 Proposer un algorithme de multiplication d'entiers utilisant le paradigme diviser-pour-régner et utilisant la remarque précédente. Faire une rapide étude de complexité dans le cas où les entrées sont de taille 2^p. Cet algorithme diviser-pour-régner présente-t-il un avantage par rapport à l'algorithme naïf.

Q. 4 En s'intéressant à la valeur de $u_a v_a + u_b v_b - (u_a - u_b)(v_a - v_b)$ proposer un algorithme faisant 3 appels récursifs sur des entrées dont la taille est divisée par 2.

Q. 5 Faire une étude rapide de complexité dans le cas où les entiers en arguments sont de taille $n = 2^p$.

Q. 6 Donner la complexité pire cas de l'algorithme de la question 4 au moyen d'un Θ. Au vu de la question précédente, on pourra intuiter puis démontrer par récurrence un encadrement sur le nombre de multiplications élémentaires effectuées par l'algorithme de la question 4 sur deux entrées de taille respectivement n et m.

Q. 7 Dans les questions précédentes nous avons ignorer le coût des additions et soustractions sur les grands entiers. On ne suppose plus que c'est le cas : une addition, soustraction sur deux entiers de tailles n et m coûte dorénavant $\max(n, m)$. Faire une rapide étude de complexité de l'algorithme de la question 4 (on se contentera des entiers de la forme 2^p). Conclure.