TD6 : Complexité

1 Recherche de nombres premiers

Question 1 : Un algorithme naïf pour tester si un nombre est premier est de tester s'il est divisible par tous les entiers qui lui sont inférieurs. Quelle est la complexité de cet algorithme dans le pire des cas ?

Question 2 : Comment améliorer cette complexité en testant moins de diviseurs potentiels ?

Question 3 : L'algorithme d'Eratosthène propose une autre approche qui est de trouver tous les entiers premiers inférieurs à une borne donnée n. Son principe est de maintenir la liste des entiers de entiers de 2 à n, en entourant les entiers dont on sait qu'ils sont premiers et en barrant ceux dont on a trouvé un diviseur. À chaque itération on repère le plus petit entier non marqué (ni entouré, ni barré) : il est forcément premier, on l'entoure et on barre tous ses multiples. [Je n'écris pas la fonction car elle est à faire en TP flocon cette semaine.] Montrer que l'algorithme d'Eratosthène est en $\mathcal{O}(n \log n)$. (Indication [1] : $\sum_{k=1}^{n} \frac{1}{k} = \Theta(\log n)$.)

Question 4 : Conclure sur lequel des deux algorithme est le plus efficace.

2 Recherche dans un texte

On dispose d'un texte (une suite de caractères, qu'on suppose assez longue), et d'un extrait (une suite de caractères, qu'on suppose plus courte) et on veut donner les endroits du texte, s'il y en a, où apparaît cet extrait, par exemple sous la forme d'une suite d'indices.

Question 5 : Proposer un algorithme naïf pour ce problème.

Question 6 : Donner un majorant de la complexité dans le pire des cas.

Question 7 : On ajoute comme hypothèse (peu réaliste, mais c'est une hypothèse de travail) que tous les caractères de l'extrait sont distincts. Proposer une amélioration de l'algorithme.

Question 8 : Donner un majorant de la complexité dans le pire des cas qui améliore la borne précédente.

3 Multiplication de nombres

On s'intéresse ici à la complexité de la multiplication de deux entiers positifs à n chiffres chacun (par exemple en base 2, mais tous les raisonnements s'appliquent de manière identique dans n'importe quelle base entière), en terme de multiplications élémentaires, c'est-à-dire de multiplication entre nombres à un chiffre, et d'additions.

Question 9 : Donner un ordre de grandeur de la complexité temporelle de l'addition de deux entiers positifs à n chiffres chacun.

Question 10 : Donner un ordre de grandeur de la complexité temporelle de la multiplication de deux entiers positifs à n chiffres chacun, par la méthode que vous avez apprise à l'école (qui se transpose naturellement en base 2).

Algorithme de Karatsuba

Soit un entier $n \in \mathbb{N}^*$. On considère deux entiers positifs u et v ayant $2n$ chiffres chacun (le chiffre d'indice 0 est le chiffre le moins significatif) :

$$u = (u_{2n-1} \dots u_1 u_0)_2, \qquad v = (v_{2n-1} \dots v_1 v_0)_2 .$$

1. Pour les curieux, une façon de la montrer : on pose $H_n = \sum_{k=1}^{n} \frac{1}{k}$, $u_n = H_n - \ln n$ et $v_n = u_n - \frac{1}{n}$. On montre que les deux suites sont adjacentes, ce qui prouve l'ordre de grandeur annoncé. Une autre méthode classique consiste à passer par des intégrales, je vous la laisse.

Question 11 : On sépare l'écriture de u et v en deux moitiés : la moitié la plus significative et la moitié la moins significative :

$$u = 2^n U_1 + U_0, \qquad v = 2^n V_1 + V_0\,,$$

où les U_i, V_i sont des nombres à n chiffres. Exprimer les U_i, V_i en fonction des chiffres de u et v.

Question 12 : Montrer que

$$uv = (2^{2n} + 2^n)U_1 V_1 + 2^n (U_1 - U_0)(V_0 - V_1) + (2^n + 1)U_0 V_0\,.$$

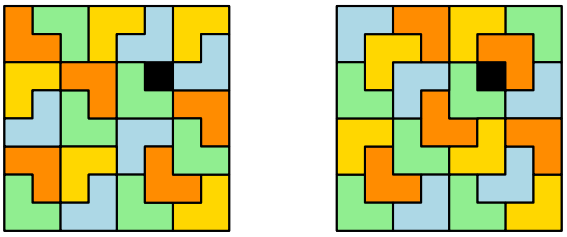
Question 13 : Sachant que multiplier par une puissance de 2 se fait en temps constant sur un ordinateur (c'est juste un décalage de bits), en déduire que uv peut être calculé en faisant 3 multiplications sur des nombres à n chiffres.

Question 14 : En déduire un algorithme récursif de calcul du produit de deux entiers ayant le même nombre de chiffres.

Question 15 : Calculer la complexité de cet algorithme.

4 Pavage

On considère une grille carrée de côté n dans laquelle une case (i, j) est interdite ($1 \leq i, j \leq n$). On suppose que n est une puissance de 2 : $n = 2^k$.
On veut paver la grille (excepté la case interdite) avec des tuiles de trois cases en forme de L, orientées dans n'importe quel sens, comme sur les exemples ci-dessous (pour lesquels $n = 8$).



Question 16 : Montrer que $2^k - 1$ est divisible par 3 (la question du pavage n'est donc pas absurde).

Question 17 : Proposer une méthode basée sur la stratégie *diviser pour régner* pour répondre à la question.

Question 18 : Évaluer le nombre d'appels récursifs effectués en fonction de n.