**IFT 2125 Hiver 2018** **Pierre McKenzie**
**Introduction à l'algorithmique**

**Devoir 3**

Remise : le vendredi 23 mars (au *début* de la démo)

*See further down for the English translations of questions 1 and 2.*

1. Implantez en Python une variante du tri par fusion (mergesort, B&B Section 7.4.1) où l'exemplaire donné en entrée est divisé en 4 sous-exemplaires, de tailles le plus possible égales, pour les fins des 4 appels récursifs. Votre méthode

   $$\text{mergesort4}(A, n)$$

   doit prendre en entrée un tableau $A$ de $n$ entiers et retourner le tableau trié.

   Déposez votre méthode sur Studium.

   Quel est l'ordre exact du temps d'exécution $t(n)$ de votre méthode en pire cas ? (Justifiez en vous servant de la théorie développée en cours.)

   Supposons que, dans un délire d'enthousiasme, nous poussions la stratégie jusqu'à diviser l'exemplaire du problème en $\lfloor\sqrt{n}\rfloor$ sous-exemplaires de tailles à peu près $\sqrt{n}$, résolvant ces $\sqrt{n}$ sous-exemplaires récursivement avant de refusionner. Soit $T(n)$ le temps d'exécution de cet algorithme délirant. Est-ce que $O(T(n)) \subseteq O(t(n))$ ? (Justifiez.)

2. L'algorithme ci-dessous manipule de grands entiers :

   `fonction` turbo$(a, n)$
   `si` $n == 1$ `alors retourner` $a$
   `sinon si` $n == 2$ `alors retourner` $a \times a$
   `sinon`
   $b \leftarrow$ nombre de bits à 1 dans binaire$(n)$
   `si` $b == 1$ `alors`
   $z \leftarrow$ nombre de zéros dans binaire$(n)$
   $p \leftarrow$ nombre dont le binaire est $1\underbrace{000\cdots0}_{\lfloor\frac{z}{2}\rfloor \text{ zéros}}$
   $q \leftarrow$ nombre dont le binaire est $1\underbrace{000\cdots0}_{\lceil\frac{z}{2}\rceil \text{ zéros}}$
   `retourner` turbo(turbo$(a, p)$,$q$)
   `sinon`
   $p \leftarrow n$ dont les $\lceil\frac{b}{2}\rceil$ bits à 1 les moins significatifs sont remplacés par 0
   `retourner` turbo$(a, p) \times$ turbo$(a, n - p)$.

   (a) Exhibez l'effet de turbo$(a, 51)$ en traçant les appels à turbo engendrés.

   (b) Posez une récurrence décrivant le nombre $t(k)$ d'appels à turbo engendrés par l'appel turbo$(a, 2^{2^k})$ (en comptant ce dernier appel).

   (c) Soit $T(n)$ le nombre d'appels engendrés par turbo$(a, n)$. Avec justification, donnez une fonction $f(n)$ telle que
   $T(n) \in \Theta(f(n)|n$ est une puissance de deux d'une puissance de deux).

3. Vous aurez réalisé que turbo$(a, n)$ calcule $a^n$. Or nous avons étudié un autre algorithme, expoDC$(a, n)$, qui fait cela. Combien de multiplications requièrent respectivement turbo$(a, n)$ et expoDC$(a, n)$ pour le calcul de $a^{51}$ ?

   Est-il possible de faire encore mieux que cela, en termes du nombre de multiplications, pour le calcul de $a^{51}$ ? Par exemple, $a^{13}$ s'exprime en 5 multiplications :

   $$x_1 = a \times a \ ; \ x_2 = x_1 \times x_1 \ ; \ x_3 = x_2 \times x_2 \ ; \ x_4 = a \times x_3 \ ; \ x_5 = x_2 \times x_4.$$

*You will have noted that turbo$(a, n)$ computes $a^n$. We have seen another algorithm, namely expoDC$(a, n)$, for doing this. How many products are performed by turbo$(a, n)$ and by expoDC$(a, n)$ respectively for the computation of $a^{51}$ ?*
*Is it possible to do better than that, in terms of the number of products, for computing $a^{51}$ ? For example, $a^{13}$ can be done using 5 products :*

$$x_1 = a \times a \ ; \ x_2 = x_1 \times x_1 \ ; \ x_3 = x_2 \times x_2 \ ; \ x_4 = a \times x_3 \ ; \ x_5 = x_2 \times x_4.$$

4. Solutionnez le problème 7.42 de Brassard et Bratley ("switches") ; décrivez votre construction et la récurrence qui justifie la taille demandée.
*Solve problem 7.42 in B&B ; describe your construction and exhibit the recurrence that supports your claim on the size of the circuit.*

*Questions 1 and 2 in English now*

1. Implement in Python a variant of mergesort (B&B Section 7.4.1) in which the input instance in divided into 4 smaller instances, of sizes as much as possible equal, for the purpose of the 4 recursive calls. Your method

$$\text{mergesort4}(A, n)$$

must take a table $A$ of $n$ integers and return the sorted table.

Upload your method on Studium.

What is the exact order of the execution time $t(n)$ of your method in the worst case ? (Justify using the theory developped in class.)

Suppose we went haywire and applied the same strategy but divided the input instance into $\sqrt{n}$ instances of size roughly $\sqrt{n}$. Let $T(n)$ be the execution time of this haywire algorithm. Does $O(T(n)) \subseteq O(t(n))$ hold ? (Justify.)

2. The following algorithm manipulates large integers :

`fonction` turbo$(a, n)$
`si` $n == 1$ `alors retourner` $a$
`sinon si` $n == 2$ `alors retourner` $a \times a$
`sinon`
  $b \leftarrow$ number of 1-bits in the binary rep of $n$
  `si` $b == 1$ `alors`
    $z \leftarrow$ number of 0-bits in the binary rep of $n$
    $p \leftarrow$ number whose binary rep is $1\underbrace{00\cdots0}_{\lfloor \frac{z}{2} \rfloor \text{ zéros}}$
    $q \leftarrow$ number whose binary rep is $1\underbrace{00\cdots0}_{\lceil \frac{z}{2} \rceil \text{ zéros}}$
    `retourner` turbo(turbo$(a, p)$,$q$)
  `sinon`
    $p \leftarrow n$ whose $\lceil \frac{b}{2} \rceil$ least significant 1-bits are replaced by 0
    `retourner` turbo$(a, p)$ $\times$ turbo$(a, n - p)$.

— Illustrate the effect of turbo$(a, 51)$ by tracing the recursive calls spawned.
— Set the recurrence describing the number $t(k)$ of calls to turbo spawned by turbo$(a, 2^{2^k})$ (including this call).
— With justification, give a function $f(n)$ such that $t$ from part (b) satisfies $t(n) \in \Theta(f(n)|n$ is a power of two of a power of two).