Devoir maison

CPES2: Algorithmique et Applications

Pierre SENELLART

À rendre pour le 22 novembre 2020

Ce devoir maison est formé de deux exercices. Vous devez d'ici la date limite (22 novembre 2020, 23h59) soumettre votre devoir sur le site Moodle du cours, à l'endroit indiqué, sous la forme d'un unique PDF (qui peut contenir un document produit électroniquement ou des scans ou photos de documents papier manuscrits, tant que le résultat est lisible et prend moins de 10 MB). Tout retard dans le rendu sera pénalisé de points en moins. Le devoir est constitué de deux exercices notés sur 10 points chacun.

A. Étude d'un algorithme de tri (10 points)

On considère l'algorithme suivant :

Entrée: Tableau A de n éléments comparables deux à deux
Sortie: Tableau A avec ces mêmes éléments triés

1: **if** $n \leqslant 2$ **then**
2: Trier A avec tri par insertion
3: **else**
4: Trier les $\lceil 2n/3 \rceil$ premiers éléments de A en appliquant l'algorithme récursivement
5: Trier les $\lfloor 2n/3 \rfloor$ derniers éléments de A en appliquant l'algorithme récursivement
6: Trier les $\lceil 2n/3 \rceil$ premiers éléments de A en appliquant l'algorithme récursivement
7: **end if**
8: **return** A

(Ici, $\lceil \bullet \rceil$ et $\lfloor \bullet \rfloor$ dénotent respectivement les parties entières supérieures et inférieures.)

1. *(2 points)* Appliquer à la main cet algorithme à la séquence de 9 entiers suivants :

$$12, 37, 42, 21, 15, 18, 7, 3, 21$$

   en détaillant les différentes étapes du calcul.

2. *(2 points)* Écrire une fonction Python `tri` qui prend en argument une liste Python `A` et applique cet algorithme à cette liste.

3. *(2 points)* Prouver que cet algorithme est correct, c'est-à-dire que le tableau renvoyé en sortie est bien un tri du tableau passé en entrée.

4. *(1 point)* Exprimer la complexité asymptotique en temps $T(n)$ de cet algorithme en fonction de la taille du tableau d'entrée par une expression de récurrence. On s'autorisera à approximer $\lfloor k \rfloor$ et $\lceil k \rceil$ par k.[1]

1. Pour un traitement rigoureux de cette approximation, voir la partie 4.6.2 du livre de référence (Cormen et al.).

5. *(2 points)* En utilisant le théorème maître, résoudre cette récurrence : quelle est la complexité asymptotique de cet algorithme ?
6. *(1 point)* Discuter de cette complexité par rapport à celles des algorithmes classiques de tri.

B. Compteur binaire (10 points)

On veut utiliser une structure de données permettant de stocker des entiers naturels en base 2 et d'effectuer des opérations élémentaires dessus. Un entier n sera représenté par un tableau de k bits $a_0, \ldots, a_{k-1}$ tel que :

$$n = \sum_{i=0}^{k-1} a_i \times 2^i.$$

On suppose que k est fixé une fois la structure de données construite.

7. *(0.5 point)* Quel est le nombre maximum $n_{\max}(k)$ représentable par un tableau de k bits ?
8. *(4 points)* Écrire une classe Python `CompteurBinaire` implémentant cette structure de données et comportant les méthodes suivantes :
   - Un constructeur prenant en argument la valeur de k et initialisant un tableau de k bits tous mis à zéro ; comme nous n'aurons jamais besoin de changer la taille de ce tableau, on utilisera un tableau fixe tel qu'implémenté par la classe `numpy.array` du module `numpy`.
   - Une méthode `getK` sans argument renvoyant la valeur de k.
   - Une méthode `nbSetBits` sans argument renvoyant le nombre de bits parmi $a_0 \ldots a_{k-1}$ valant 1.
   - Une méthode `increment` sans argument incrémentant l'entier n représenté par le tableau de bits ; si l'entier vaut initialement $n_{\max}(k)$, le résultat de l'incrémentation est le tableau représentant le nombre 0.
   - Une méthode `decrement` sans argument décrémentant l'entier n représenté par le tableau de bits ; si l'entier vaut initialement 0, le résultat de la décrémentation est le tableau représentant le nombre $n_{\max}(k)$.
   - Une méthode `getInt` sans argument renvoyant un entier Python correspondant à l'entier représenté par le tableau de bits.
9. *(1.5 point)* Quelle est la complexité asymptotique en temps, dans le pire des cas, de chacune des méthodes de la classe `CompteurBinaire`, en fonction de k ? Justifier votre affirmation. Peut-on faire en sorte que la fonction `nbSetBits` ait une complexité asymptotique de $O(1)$, et si oui, comment ?
10. *(3 points)* On s'intéresse à la complexité *amortie* d'une séquence de m incrémentations, en commençant par un objet fraîchement construit. En proposant une fonction de potentiel appropriée, utiliser la méthode du potentiel pour calculer la complexité amortie d'une incrémentation parmi une séquence de m incrémentations, en fonction de k.
11. *(1 point)* Supposons qu'on considère maintenant une séquence de m incrémentations ou décrémentations. En proposant un exemple pathologique de telle séquence, montrer que la complexité amortie n'est pas asymptotiquement meilleure que la complexité non-amortie pour une telle séquence.