DM

Tous les algorithmes sont à écrire en pseudo-code selon les modèles vus en cours. Pour chacun d'entre eux, il faudra expliquer son comportement et justifier que l'algorithme est correct (par exemple à l'aide d'invariants de boucle). Le devoir est à rendre au format numérique à l'adresse `fpirot@lisn.fr`, au plus tard le 30/01/2022 à 23h59.

Exercice 1.
Dans cet exercice, on s'intéresse au problème du sous-tableau maximum. On nous donne en entrée un tableau `t` de n nombres, positifs et négatifs. Le but est de trouver la paire $0 \leq i \leq j \leq n-1$ telle que la somme `t[i]+t[i+1]+` $\cdots$ `+t[j]` est maximale.

1. Écrire un algorithme `MaxFrom` qui prend en entrée `t` et une position i, et retourne le sous-tableau maximum de `t` démarrant à la position i. Quelle est sa complexité ? — 2 pts
2. Écrire un algorithme `MaxTo` qui prend en entrée `t` et une position j, et retourne le sous-tableau maximum de `t` finissant à la position j. Quelle est sa complexité ? — 1 pt
3. Écrire un algorithme `MaxSubarray` de complexité quadratique qui résout le problème du sous-tableau maximum. — 2 pts
4. On cherche maintenant à obtenir un algorithme de meilleure complexité en utilisant la méthode Diviser pour Régner.
   (a) Posons une position i. On considère le tableau `t0` qui contient les éléments `t[0],...,t[i-1]` et le tableau `t1` qui contient les éléments `t[i],...,t[n-1]`. Montrer que l'on peut déduire le résultat de `MaxSubarray(t)` à partir des valeurs `MaxSubarray(t0)`, `MaxSubarray(t1)`, `MaxTo(t0,i-1)`, `MaxFrom(t1,i)`. — 2 pts
   (b) Écrire une version récursive de `MaxSubarray` qui repose sur le principe Diviser pour Régner, à l'aide de l'observation précédente. — 2 pts
   (c) Utiliser le Master Theorem pour calculer la nouvelle complexité de `MaxSubarray`. — 1 pt

Exercice 2.
Nous avons vu en TD que la recherche d'un élément au sein d'un tableau trié de n éléments peut se faire en temps $O(\log n)$. En revanche, l'insertion d'un élément x dans un tableau trié (partiellement rempli) de taille n se fait en temps $\Theta(n)$ dans le pire cas, même s'il n'y a pas besoin de réallouer de mémoire, puisque cela demande de décaller d'une case vers la droite toutes les valeurs supérieures à x dans le tableau.

Le but de cet exercice est de mettre au point une structure de données S qui propose un meilleur compromis entre la complexité de la recherche et de l'insertion d'un élément en son sein.

Lorsque S contient $n \geq 1$ nombres, on peut la décrire comme suit. S contient $b_0, \ldots, b_m$ les bits de la représentation binaire de n (b_0 est le bit de poids faible et b_m le bit de poids fort, et on a $m = \lfloor \log_2 n \rfloor$). Les n nombres contenus dans S sont rangés dans des tableaux $t_0, t_1, \ldots, t_m$, où t_i est de taille 2^i, et est entièrement rempli si $b_i = 1$ ou vide (toutes ses cases valent `None`) si $b_i = 0$. De plus, chaque tableau t_i est trié, mais on n'a aucune hypothèse sur l'ordre relatif de deux nombres contenus dans des tableaux différents.

1. Écrire un algorithme `EstPrésent` qui teste la présence d'un nombre x au sein de la structure S. Quelle est sa complexité (une complexité sous-linéaire est attendue) ? 2 pts
2. On s'intéresse maintenant à l'insertion d'un nouvel élément dans S. Cette opération repose sur le même principe que l'opération `incr` sur le compteur dans le TD2. Si $b_0 = 0$, le tableau `t0` est vide, et on peut donc insérer x directement dans `t0`. Si $b_0 = 1$ et $b_1 = 0$, on insère x et l'élement de t_0 directement dans t_1. Si $b_0 = 1$, $b_1 = 1$, et $b_2 = 0$, on insère x et les éléments de t_0 et t_1 dans t_2. Et ainsi de suite, quitte à devoir créer un nouveau tableau t_{m+1}.
   (a) Écrire un algorithme `Fusion` qui prend en entrée deux tableaux triés t_0 et t_1 de taille respective n_0 et n_1, et renvoie un tableau trié t contenant l'union des éléments de t_0 et t_1. Sa complexité devra être $O(n_0 + n_1)$. 2 pts
   (b) Écrire un algorithme `Insère` qui insère un nombre x dans la structure S (on pourra faire plusieurs appels à `Fusion`, et utiliser une fonction `Vider(t)` de complexité linéaire qui vide le tableau `t` passé en argument). On veillera à mettre à jour tous les paramètres internes de la structure S. 3 pts
   (c) Exprimer la complexité de `Insère` en fonction du nombre de 1 consécutifs à la fin de la représentation binaire de n. 1 pt
   (d) Quelle est la complexité de répéter N fois l'opération `Insère`, en partant d'une structure S vide (une complexité sous-quadratique est attendue) ? 2 pts
3. (Bonus) Décrire la procédure à suivre pour supprimer un nombre (dont on nous donne la position via un pointeur) de S, et évaluer sa complexité en fonction de la représentation binaire de n. Quelle est la complexité dans le pire cas de N insertions et/ou suppressions successives de nombres dans S ? 3 pts