1 Partie A

1. On considère le Problème 3-SAT-NAE qui prend en entrée une formule sous forme 3-CNF et renvoie vrai ssi il existe une valuation telle que les trois littéraux de chaque clause ne sont pas tous évalués de manière identique.
   (a) Montrer que si une formule est une instance positive de 3-Sat-NAE alors elle est satisfiable.
   (b) Montrer que ce problème est dans la classe NP.
   (c) A l'aide d'une réduction depuis 3-Sat, montrer que ce problème est NP-complet (indication : transformer les clauses $C_i = a_i \vee b_i \vee c_i$ en $C'_i = a_i \vee b_i \vee z_i$ et $C''_i = \neg z_i \vee c_i \vee f$).
2. On considère le Problème 3-SAT-OIT qui prend en entrée une formule sous forme 3-CNF et renvoie vrai ssi il existe une valuation telle que exactement un des trois littéraux de chaque clause soit vrai.
   A l'aide d'une réduction depuis 3-Sat, montrer que ce problème est NP-complet. Indication : transformer les clauses $C_i = a_i \vee b_i \vee c_i$ en $C'_i = a_i \vee x_i \vee y_i$ et $C''_i = \neg b_i \vee x_i \vee x'_i$ et $C'''_i = \neg c_i \vee y_i \vee y'_i$.

2 Partie B

1. Écrire une fonction `somme : int array -> int -> int` telle que l'appel `somme t i` calcule la somme partielle $\sum_{k=0}^{i} t.(k)$ des valeurs du tableau t entre les indices 0 et i inclus.

Un tableau t de $n > 0$ éléments de $[\![0, n-1]\!]$ est dit *autoréférent* si pour tout indice $0 \leq i < n$, $t.(i)$ est exactement le nombre d'occurrences de i dans t, c'est-à-dire que

$$\forall i \in [\![0, n-1]\!], \quad t.(i) = \operatorname{card}\left(\{k \in [\![0, n-1]\!] \mid t.(k) = i\}\right)$$

Ainsi, par exemple, pour $n = 4$, le tableau suivant est autoréférent :

i	0	1	2	3
$t.(i)$	1	2	1	0

En effet, la valeur 0 existe en une occurrence, la valeur 1 en deux occurrences, la valeur 2 en une occurrence et la valeur 3 n'apparaît pas dans t.

2. Justifier rapidement qu'il n'existe aucun tableau autoréférent pour $n \in [\![1; 3]\!]$ et trouver un autre tableau autoréférent pour $n = 4$.
3. Écrire une fonction `est_auto : int array -> bool` qui vérifie si un tableau de taille $n > 0$ est autoréférent. On attend une complexité en $O(n)$.

On propose d'utiliser une méthode de retour sur trace (*backtracking*) pour trouver tous les tableaux autoréferents pour un $n > 0$ donné. Une fonction `gen_auto` qui affiche tous les tableaux autoréférents pour une taille donnée vous est proposée. Cette version ne fonctionne cependant que pour de toutes petites valeurs de n (instantané pour $n = 5$, un peu long pour $n = 8$, sans espoir pour $n = 15$). On pourra vérifier qu'il existe exactement deux tableaux autoréférents pour $n = 4$, un seul pour $n \in \{5, 7, 8\}$ et aucun pour $n = 6$.

Pour accélérer la recherche, il faut élaguer l'arbre (repérer le plus rapidement possible qu'on se trouve dans une branche ne pouvant pas donner de solution).

4. Que peut-on dire de la somme des éléments d'un tableau autoréférent ? En déduire une stratégie d'élagage pour accélérer la recherche. *Indication : utiliser la fonction* `somme` *de la première question pour interrompre par un échec l'exploration lorsque* `somme t i` *dépasse déjà la valeur maximale possible.*
5. Que peut-on dire si juste après avoir affecté la case $t.(i)$, il y a déjà strictement plus d'occurrences d'une valeur $0 \leq k \leq i$ que la valeur de $t.(k)$? En déduire une stratégie d'élagage supplémentaire et la mettre en œuvre. Combien de temps faut-il pour résoudre le problème pour $n = 15$?
6. Après avoir affecté la case $t.(i)$, combien de cases reste-t-il à remplir ? Combien de ces cases seront complétées par une valeur non nulle ? À quelle condition est-on alors certain que la somme dépassera la valeur maximale possible à la fin ? En déduire une stratégie d'élagage supplémentaire et la mettre en œuvre. Combien de temps faut-il pour résoudre le problème pour $n = 30$?
7. Montrer qu'il existe un tableau autoréférent pour tout $n \geq 7$. *On pourra conjecturer la forme de ce tableau en testant empiriquement pour différentes valeurs de* $n \geq 7$. *On ne demande pas de montrer que cette solution est unique.*