Prénom, NOM : .....

Devoir en temps libre n° 3

Ce troisième devoir en temps libre est à préparer pour le jeudi 6 décembre 2018. Les questions comportant le symbole ✍ sont à rédiger sur feuille et à rendre au début du TP, tout comme ce sujet complété avec la partie « vrai ou faux ». Comme d'habitude, le code est à rendre à 13h15, pendant le TP (vous aurez 15 minutes pour prendre connaissance de vos éventuelles erreurs et pour les corriger). L'archive `dm3.zip` est disponible sur le site `https://cpge.info` avec le fichier à compléter `dm3.py` ainsi que le script de vérification `verif_dm3.py`.

Avant de traiter les exercices on s'assurera d'avoir bien compris et travaillé le cours, le TD n° 1, le TP n° 8, le TP n° 9 et leur corrigé.

1 Vrai ou faux ?

On considère le programme suivant, en supposant en argument un entier $n \in \mathbb{N}$:

PYTHON
```
def myst(n):
    m = n + 2
    while n != 0:
        if m == n:
            n = n - 1
        else:
            m = m - 1
```

1. `n` est un variant de boucle du `while` de `myst` ?
   ☐ Vrai ☐ Faux
2. `m` est un variant de boucle du `while` de `myst` ?
   ☐ Vrai ☐ Faux
3. `n + m` est un variant de boucle du `while` de `myst` ?
   ☐ Vrai ☐ Faux
4. `n * m` est un variant de boucle du `while` de `myst` ?
   ☐ Vrai ☐ Faux
5. `myst` termine toujours ?
   ☐ Vrai ☐ Faux
6. `myst` renvoie toujours `None` ?
   ☐ Vrai ☐ Faux
7. `n <= m` est un invariant de boucle du `while` de `myst` ?
   ☐ Vrai ☐ Faux

2 Correction, terminaison et complexité d'une fonction

On considère le programme suivant, en supposant une liste d'entiers en argument :

PYTHON
```
def mystere(liste):
    res = 0
    for i in range(len(liste)):
        if liste[i] > 0:
            res += 1
    return res
```

8. ✍ Que fait le programme `mystere` ? On demande une explication simple en une phrase, qui pourrait constituer la chaîne de documentation. Proposer un nom simple pour cette fonction.
9. ✍ Trouver un invariant de boucle et justifier *très précisément* la correction de la fonction.
10. ✍ Estimer la complexité de cette fonction.
11. ✍ Réécrire cette fonction en utilisant une boucle `while` plutôt qu'une boucle `for`.
12. ✍ Donner un variant de boucle, en justifiant *très précisément*, et montrer que votre fonction termine.

13. ✍ Trouver un invariant de boucle et justifier *très précisément* la correction de la fonction.
14. ✍ Estimer la complexité de cette deuxième version.
15. ✍ Réécrire cette fonction en utilisant une boucle `for` sur les éléments plutôt que sur les indices

3 Entiers factoriels

16. Écrire une fonction `est_factoriel(n)` qui vérifie si un entier $n \geq 1$ est factoriel, *i.e.* $\exists k \in \mathbb{N}^*, n = k!$. On écrira une fonction efficace qui ne calcule pas inutilement un grand nombre de fois la même chose, de la même manière que pour tester si un entier est un entier de Fibonacci.
17. ✍ Montrer que votre fonction termine.
18. ✍ Donner un invariant de boucle qui permet de montrer que votre fonction est correcte.
19. ✍ Montrer que la complexité temporelle de cette fonction est en $O(n)$. On demande simplement de majorer la complexité, pas de l'estimer précisément.

Définition 3.1

Un *nombre premier factoriel* est un nombre premier de la forme $k! - 1$ ou $k! + 1$ avec $k \in \mathbb{N}^*$.

20. Écrire une fonction `nombres_premiers_non_factoriels(n)` qui renvoie la liste dans l'ordre croissant de tous les entiers premiers plus petits ou égaux à n qui ne sont pas des nombres premiers factoriels .

4 Dépouillement d'élections

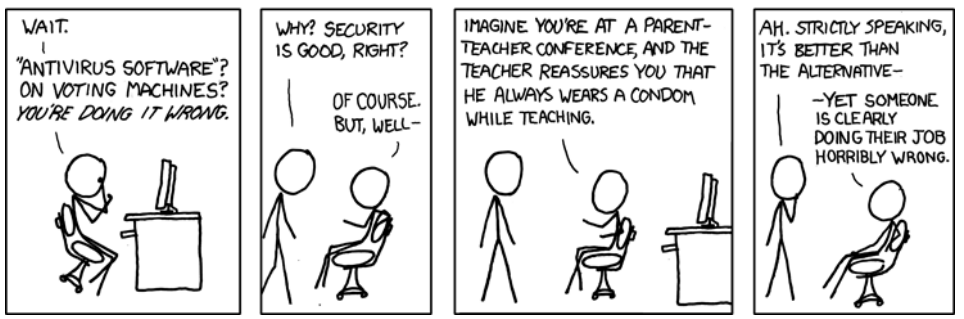
Le problème suivant consiste à faire le décompte du résultat des élections dans le pays JJ-BOUBOU. La règle électorale y est très laxiste, puisque l'on peut être élu sans s'être présenté. Chaque votant identifié par un numéro d'électeur $i \in [\![0, n-1]\!]$ vote pour un autre habitant `vote[i]`, où `vote[i]` est un entier positif arbitrairement grand qui représente le numéro de sécurité sociale d'un habitant du pays JJ-BOUBOU. On dispose de la liste `vote` qui contient les votes des n habitants du pays des JJ-BOUBOU. Par exemple, avec trois habitants, la liste `vote` pourrait être `vote_ex = [123456789101112, 666, 123456789101112]`.

Dans le pays JJ-BOUBOU, un candidat ayant obtenu strictement plus de 50% des suffrages exprimés est directement élu au premier tour des élections.

21. Écrire une fonction `majorite_absolue(vote, k)` qui retourne `True` si le candidat de numéro de sécurité sociale `k` possède la majorité absolue et `False` sinon. Par exemple, `majorite_absolue(vote_ex, 666)` renvoie `False`.
22. Écrire une fonction `gagnant_tour1(vote)` qui retourne le gagnant du premier tour s'il existe et `None` sinon. On pourra utiliser la fonction de la question précédente et on se contentera d'une fonction faisant le calcul en temps quadratique par rapport à n.
23. Supposons la liste `vote` triée dans l'ordre croissant. Écrire alors une fonction `gagnant_tour1_lineaire(vote)` qui retourne *en temps linéaire* par rapport à n le gagnant du premier tour s'il existe et `None` dans le cas contraire.

Si aucun habitant n'est élu au premier tour on recommence complètement les élections. Au second tour, le candidat ayant obtenu le plus de voix est élu. S'il y a égalité de voix, tous les candidats ayant obtenu le plus de voix sont élus (ex æquo).

24. Écrire une fonction `gagnant_tour2(vote)` qui retourne dans une liste les gagnants du second tour. On prendra garde à ne faire apparaître qu'une seule fois le numéro de sécurité sociale des gagnants dans la liste finale. On pourra utiliser les ensembles PYTHON (`set`). On se contentera d'une fonction faisant le calcul en temps quadratique par rapport à n.
25. Supposons à nouveau la liste `vote` triée en ordre croissant. Écrire alors une fonction `gagnant_tour2_lineaire(vote)` qui retourne en temps linéaire par rapport à n le gagnant du second tour.



https://xkcd.com/463/