Université Mohammed VI Polytechnique **Année 2020-2021**
Licence LSD **Structures de données**

TRAVAUX PRATIQUES
SÉRIE I

Exercice 1. *La suite de Fibonacci est une suite de nombres entiers définie par* $F_0 = 0$, $F_1 = 1$ *et pour tout* $n \geq 2$ *par*

$$F_n = F_{n-1} + F_{n-2}.$$

1. *Écrire une fonction* `recursive_fibo()` *qui prend en argument un entier positif* n *et calcule de manière récursive* F_n.
2. *Écrire une fonction* `iterative_fibo()` *qui prend en argument un entier positif* n *et calcule de manière itérative* F_n.
3. *Estimer la complexité des deux fonctions* `recursive_fibo()` *et* `iterative_fibo()`.
4. *Vérifier sur des grandes valeurs de* n *si la fonction* `recursive_fibo()` *provoque un dépassement de la pile d'exécution (stackoverflow) mais la fonction* `iterative_fibo()` *donne la valeur de* F_n.

Exercice 2. *Déclarer un tableau d'entiers* T *de taille* 1000. *Remplir ensuite ce tableau avec des valeurs aléatoires en utilisant les fonctions* `rand()` *et* `srand()` *de la librarie* `stdlib`. *Afficher le tableau* T *et calculer enfin son troisième plus grand élément sans utiliser un algorithm de tri.*

Exercice 3. *Dans cet exercice nous étudions quelques optimisations de l'algorithme naïf qui permet de tester si un entier* n *est premier.*

1. *Écrire une fonction* `is_prime()` *qui prend en argument un entier* n *et décide si* n *est un nombre premier. La fonction teste si* n *est divisible par un entier entre* 2 *et* $|n| - 1$. *Si un tel entier est trouvé elle retourne* 0, *sinon elle retourne* 1.
2. *Écrire une fonction* `get_primes()` *qui prend en argument un entier positif* n *et affiche tous les entiers premiers entre* 2 *et* n.
3. *En donnant à* n *des grandes valeurs, constater la lenteur de l'exécution de la fonction* `get_primes()`. *Donner une explication théorique à cette lenteur en estimant la complexité de* `get_primes()`.

   On considère maintenant les deux optimisations suivantes.
   - *On peut facilement montrer que si un entier* n *n'est pas premier alors il possède un diviseur* p *tel que* $2 \leq p \leq \sqrt{|n|}$. *En conséquence, au lieu de chercher les diviseurs potentiels de* n *entre* 2 *et* $|n| - 1$ *on peut se limiter à ceux entre* 2 *et* $\sqrt{|n|}$.
   - *Le seul nombre positif premier pair est* 2. *Ainsi, lorsqu'on cherche les diviseurs de* n *entre* 2 *et* $\sqrt{|n|}$ *on n'a besoin de tester que ceux qui sont impairs.*
4. *Remplacer les deux fonctions* `is_prime()` *et* `get_primes()` *par les nouvelles fonctions* `is_prime_opt()` *et* `get_primes_opt()` *en tenant compte des optimisations ci-dessus.*
5. *Estimer la complexité de la fonction* `get_primes_opt()`.

Date: February 25, 2021.