Lycée René Cassin – PC **Devoir d'informatique** Durée : 1 h 30

Toutes les fonctions doivent être écrites en Python.
Les fonctions non triviales doivent être accompagnées de commentaires explicatifs.
On favorisera les fonctions dont le temps d'exécution est minimal.

La calculatrice est autorisée.

Exercice 1 – *Pile ou file*

Une pile (en anglais stack) est une structure de données fondée sur le principe « dernier arrivé, premier sorti » (LIFO pour Last In, First Out), ce qui veut dire que les derniers éléments ajoutés à la pile seront les premiers à être récupérés. Le fonctionnement est celui d'une pile d'assiettes : on ajoute des assiettes sur la pile, et on les récupère dans l'ordre inverse, en commençant par la dernière ajoutée.

Les seules fonctions opérant sur les piles (dans cet exercice) sont les suivantes :
- *p* = creer_pile() : crée une pile vide *p*
- empiler(*p*,*x*) : ajoute un élément *x* sur la pile *p*
- depiler(*p*) : enlève un élément de la pile *p* et le renvoie.
- pile_vide(*p*) : renvoie True si la pile est vide, False sinon.

Une file (queue en anglais) est une structure de données basée sur le principe du Premier entré, premier sorti (FIFO : First In, First Out), ce qui veut dire que les premiers éléments ajoutés à la file seront les premiers à être récupérés. Le fonctionnement ressemble à une file d'attente : les premières personnes à arriver sont les premières personnes à sortir de la file.

Les seules fonctions opérant sur les files (dans cet exercice) sont les suivantes :
- *f* = creer_file() : crée une file vide *p*
- enfiler(*f*, *x*) : ajoute un élément *x* dans la file *f*.
- defiler(*f*) : enlève le prochain élément de la file *f* et le renvoie.
- file_vide(*f*) : renvoie True si la file est vide, False sinon.

Les files et les piles sont implémentés en Python à l'aide des listes, mais l'utilisation des listes autrement qu'en tant que file ou pile est interdite dans cet exercice.

Question 1. Ecrire une fonction prochain(*f*), d'argument une file *f*, renvoyant le prochain élément de la file *f* sans modifier la file *f*.

Question 2. Ecrire une fonction rang(*f*, *x*), d'argument une file *f* et une variable *x*, renvoyant le rang de l'élément *x* dans *f* sans modifier la file *f*. Ainsi, rang(*f*, *x*) renverra 1 si *x* est le premier élément de *f*, 2 si *x* apparait pour la première fois en deuxième position… ou bien 0 si *x* n'est pas présent dans *f*.

Question 3. Ecrire une fonction inverser(*f*), d'argument une file *f* renvoyant une file dont les éléments sont ceux de *f* dans l'ordre inverse sans modifier *f*.

Exercice 2 – *Autour de la médiane*

Dans cette partie, nous supposerons donné un tableau *tab* contenant *n* nombres réels. Les indices du tableau vont de 1 à *n*. Nous dénoterons *tab*[*a*..*b*] le tableau pris entre les indices *a* et *b*, c'est-à-dire les cellules *tab*[*a*], *tab*[*a*+1], …, *tab*[*b*–1], *tab*[*b*]. Nous supposerons que $a \leq b$.

Nous utiliserons le tableau de taille 11 suivant pour nos exemples :

3	2	5	8	1	34	21	6	9	14	8

En Python, le tableau *tab* sera représenté par une liste de *n* variables du type float.
On rappelle que les indices des listes commencent à 0 en Python.

Question 1. Ecrire une fonction calculIndiceMax(*tab, a, b*) qui renvoie l'indice d'une case où se trouve le plus grand réel de *tab*[*a*..*b*]. Sur le tableau précédent, avec $a = 2$ et $b = 9$, la fonction renverra 6 car la case 6 contient la valeur 34.

Question 2. Ecrire une fonction nombrePlusPetit(*tab, a, b, val*) qui renvoie le nombre d'éléments dans le tableau *tab*[*a*..*b*] dont la valeur est plus petite ou égale à *val*. Sur le tableau exemple, pour une valeur de *val* égale à 5, et $a = 1$, $b = 11$, la fonction devra renvoyer la valeur 4 car seuls les nombres 3, 2, 5, 1 sont inférieurs ou égaux à 5.

Nous allons maintenant calculer un médian d'un tableau. Rappelons qu'une valeur médiane *m* d'un ensemble *E* de nombres est un élément de *E* tel que les deux ensembles $E_{<m}$ (les nombres de *E* strictement plus petits que *m*) et $E_{>m}$ (les nombres de *E* strictement plus grands que *m*) vérifient $|E_{<m}| \leq [n/2]$ (le cardinal de $E_{<m}$ est inférieur à la partie entière de $n/2$) et $|E_{>m}| \leq [n/2]$.

Une méthode naïve consiste donc à parcourir les éléments de l'ensemble et à calculer pour chacun d'eux les valeurs de $|E_{<m}|$ et $|E_{>m}|$ mais il est possible de faire mieux comme nous allons le voir. Une méthode plus efficace consiste à trier le tableau par ordre croissant et à retourner la cellule du milieu dans le tableau trié. Cette méthode requiert $O(n \ln n)$ opérations. Il existe une méthode optimale en temps linéaire $O(n)$ pour trouver le médian d'un ensemble de *n* éléments. Les questions suivantes ont pour but d'en proposer une implémentation.

Nous allons utiliser une fonction partition(*tab, a, b, indicePivot*) qui prend en paramètre un tableau d'entiers *tab*[*a*..*b*] ainsi qu'un élément du tableau *tab*[*a*..*b*] appelé *pivot*, repéré par son indice dans le tableau *indicePivot*. Ainsi, *pivot* = *tab*[*indicePivot*]. La fonction devra réordonner les éléments de *tab*[*a*..*b*] en mettant en premier les éléments strictement plus petits que *pivot*, $tab_{<pivot}$, puis les éléments égaux à *pivot*, $tab_{=pivot}$ et en dernier les éléments strictement plus grands $tab_{>pivot}$. Sur le tableau exemple, en prenant comme valeur de *pivot* 8, on obtient :

3	2	5	1	6	8	8	21	34	9	14

Notez que dans le résultat, les nombres plus petits que le pivot 3, 2, 5, 1, 6 peuvent être dans n'importe quel ordre les uns par rapport aux autres.

Question 3. Ecrire la fonction partition(*tab, a, b, indicePivot*). La fonction agira directement sur le tableau *tab* et retournera le nouvel indice de la case où se trouve la valeur *pivot*.

Remarquons que le $[n/2]$-ème élément dans l'ordre croissant d'un tableau de taille *n* est un élément médian du tableau considéré. Nous allons donc non pas programmer une méthode pour trouver le médian mais plus généralement pour trouver le *k*-ème élément d'un ensemble. Nous allons utiliser l'algorithme suivant :

On cherche le *k*-ème élément du tableau *tab*[*a*..*b*].

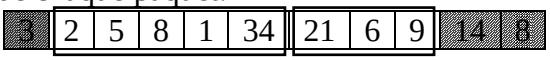
- Si $k = 1$ et $a = b$ alors renvoyer $tab[a]$
- Sinon, soit $p = tab[a]$. Partitionner le tableau $tab[a..b]$ en utilisant le pivot p en mettant en premier les éléments plus petits que p. Soit i l'indice de p dans le tableau résultant.
  - Si $i - a + 1 > k$ chercher le k-ème élément dans $tab[a..i-1]$ et renvoyer cet élément.
  - Si $i - a + 1 = k$ renvoyer le pivot.
  - Si $i - a + 1 < k$ chercher le $(k - i + a - 1)$-ème élément dans $tab[i+1..b]$ et renvoyer cet élément.

Question 4. Ecrire une fonction elementK(*tab*, *a*, *b*, *k*) qui réalise l'algorithme de sélection du *k*-ème élément dans le tableau $tab[a..b]$ décrit précédemment et renvoie cet élément.

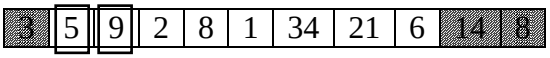
Question 5. Supposons que dans l'algorithme précédent nous voulions rechercher le premier élément mais qu'à chaque étape le pivot choisi est le plus grand élément, quel est un ordre de grandeur du nombre d'opérations réalisées par votre fonction ?

L'algorithme précédent ne semble donc pas améliorer le calcul du médian. Le problème vient du fait que le pivot choisi peut être mauvais c'est-à-dire qu'à chaque étape un seul élément du tableau a été éliminé. En fait, si l'on peut choisir un pivot p dans $tab[a..b]$ tel qu'il y ait au moins $(b - a)/5$ éléments plus petits et $(b - a)/5$ plus grands alors on peut montrer que l'algorithme précédent fonctionne optimalement en temps $O(n)$.
Pour choisir un tel élément dans $tab[a..b]$, on réalise l'algorithme choixPivot suivant où chaque étape sera illustrée en utilisant le tableau donné en introduction en prenant $a = 2$ et $b = 9$.

- On découpe le tableau en paquets de 5 éléments plus éventuellement un paquet plus petit.
  On calcule l'élément médian de chaque paquet.



  S'il n'y a qu'un paquet, on renvoie son médian. Sinon, on place ces éléments médians au début du tableau.



- On réalise choixPivot sur les médians précédents. Dans notre exemple, on recommence donc les étapes précédentes en prenant $a = 2$ et $b = 3$.

Question 6. Ecrire la fonction choixPivot(*tab*, *a*, *b*) qui réalise l'algorithme précédent et renvoie la valeur du pivot.