Algorithmique
Partiel n° 1 (P1)

Info-sup S1
Epita

9 Jan. 2018 - 10 : 00

Consignes (à lire) :

- Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - Aucune réponse au crayon de papier ne sera corrigée.
- La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
- **Le code :**
  - Tout code doit être écrit dans le langage `Python` (pas de C, Caml, Algo ou autre).
  - **Tout code `Python` non indenté ne sera pas corrigé.**
  - Tout ce dont vous avez besoin (fonctions, méthodes) est indiqué en **annexe** !
- Durée : 2h00

Exercice 1 (Pile ou file ? – *2 points*)

On ajoute, dans cet ordre, les valeurs A, B, C, D, E et F à une structure linéaire vide. Pour chacun des ordres de sortie donnés sur les feuilles de réponses, indiquer si la structure en question peut être : une pile, une file (ce peut être les deux), ou aucune des deux (ni une pile, ni une file).

Exercice 2 (Dichotomie – *3 points*)

On considèrera une version de l'algorithme de recherche dichotomique qui s'arrête dès que les bornes se croisent ou sont identiques.

1. Compléter l'arbre de décision de la recherche dichotomique sur une liste de 16 éléments. Chaque noeud représente l'intervalle de recherche (les bornes gauche et droite) ainsi que l'indice calculé du médian.
2. (a) Soit une liste de 32768 éléments triés en ordre croissant. Combien de comparaisons d'éléments seront faites, au pire des cas, dans le cas d'une recherche négative (réponse entière) ?
   (b) Soit k la réponse à la question précédente. Quelle peut-être, au maximum, la longueur de la liste qui générera $k + 2$ comparaisons lors d'une recherche négative ?

Exercice 3 (ALGO → Python – *3 points*)

Soit la fonction `test`, qui utilise les opérations du type abstrait *Liste itérative* :

```
fonction test(Liste L) : booleen
variables
    entier  i
    booleen  b
debut
    b ← vrai
    i ← 1
    tant que i < longueur(L) faire
        si ieme(L, i) > ieme(L, i+1) alors
            b ← faux
        fin si
        i ← i + 1
    fin tant que
    retourne b
fin
```

1. Que fait la fonction `test` ?
2. Écrire une version `Python` de cette fonction, si possible plus optimale que la version ALGO présentée ci-dessus.

Exercice 4 (Minimaxi – *3 points*)

Écrire une fonction qui cherche la valeur minimale ainsi que la valeur maximale d'une liste d'entiers.

Elle retourne les positions des deux valeurs dans la liste.

Exemples d'applications :

```
>>> posMiniMaxi([1, 8, -2, 9, 12, -5, 0, 25, 12])
(5, 7)
>>> posMinimax([8, 5, 8, 5, 8])
(1, 0)
>>> posMinimax([])
...
Exception: empty list
```

Exercice 5 (Tri fusion (Merge sort) – *2,5 + 5 + 2,5 points*)

1. Écrire la fonction `partition` qui sépare une liste en deux (nouvelles) listes de longueurs quasi identiques (à 1 près) : une moitié dans chaque liste.

   Exemples d'application :

```
>>> partition([15, 2, 0, 4, 5, 8, 2, 3, 12, 25])
([15, 2, 0, 4, 5], [8, 2, 3, 12, 25])
>>> partition([5, 3, 2, 8, 7, 1, 5, 4, 0, 6, 1])
([5, 3, 2, 8, 7], [1, 5, 4, 0, 6, 1])
```

2. Écrire la fonction `merge` qui fusionne deux listes triées en ordre croissant en une seule nouvelle liste triée.

   Exemple d'application :

```
>>> merge([1,5,8], [2,3,4,8])
[1, 2, 3, 4, 5, 8, 8]
```

3. Pour trier une liste `L`, on procède (récursivement) de la façon suivante :
   - ▷ Une liste de longueur < 2 est triée.
   - ▷ Une liste de longueur ≥ 2 :
     - — on partitionne la liste `L` en deux sous-listes `L1` et `L2` de longueurs quasi identiques (à 1 près) ;
     - — on trie récursivement les deux listes `L1` et `L2` ;
     - — enfin, on fusionne les listes `L1` et `L2` en une liste triée.

   Utiliser les deux fonctions précédentes (quelles soient écrites ou non) pour écrire la fonction `mergesort` qui trie en ordre croissant une liste (pas en place : la fonction construit une nouvelle liste qu'elle retourne).

   Exemple d'application :

```
>>> mergesort([5,3,2,8,7,1,5,4,0,6,1])
[0, 1, 1, 2, 3, 4, 5, 5, 6, 7, 8]
```

Annexe : Fonctions et méthodes autorisées

Vous pouvez utiliser la méthode `append` et la fonction `len` sur les listes :

```
>>> help(list.append)
Help on method_descriptor:    append(...)
    L.append(object) -> None -- append object to end of L

>>> help(len)
Help on built-in function len in module builtins:    len(...)
    len(object)
    Return the number of items of a sequence or collection.
```

Vous pouvez également utiliser la fonction `range` et `raise` pour déclencher les exceptions. Rappels :

```
>>> for i in range(10):
...     print(i, end=' ')
0 1 2 3 4 5 6 7 8 9

>>> for i in range(5, 10):
...     print(i, end=' ')
5 6 7 8 9

>>> raise Exception("blabla")
...
Exception: blabla
```