Algorithmique
Correction Partiel n° 1 (P1)

Info-sup S1 – Epita

9 Jan. 2018 - 10 : 00

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

	pile	file	aucune
A B C D E F	√	√	
B D E F A C			√

	pile	file	aucune
D E C B F A	√		
F E D C B A	√		

Solution 2 **(Dichotomie –** ***3 points*****)**

1. Arbre de décision de la recherche dichotomique :

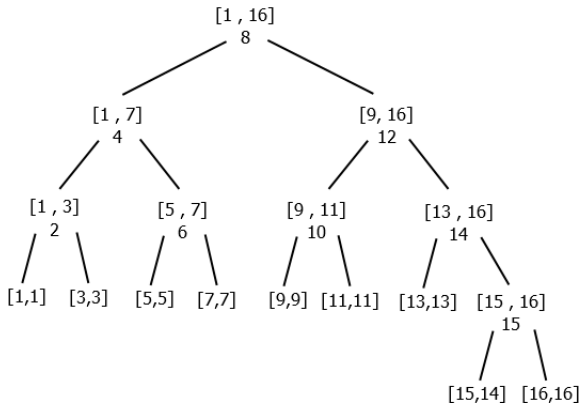

Figure 1 – Arbre de décision de la recherche dichotomique

Chaque noeud représente l'intervalle de recherche (les bornes gauche et droite) ainsi que l'indice calculé du médian. On considère ici une version de l'algorithme qui s'arrête dès que les bornes se croisent ou sont identiques.

2. (a) Nombre de comparaisons : $32 = 2 \times (15 + 1)$ (b) Longueur de la liste : 65536 (32768×2)

   $(log_2(32768) = 15)$

Solution 3 **(ALGO → Python – *4 points*)**

1. **Spécifications :**
   La fonction `test(L)` vérifie si la liste L est triée en ordre croissant.

2. La fonction Python :

```
1            def test(L):
2
3                i = 0
4                n = len(L)
5
6                while (i < n-1) and (L[i] <= L[i+1]):
7                    i = i+1
8
9                return (i >= n - 1) # or ==
```

Solution 4 **(Minimaxi – *3 points*)**

Spécifications :
La fonction `posMiniMaxi(M)` retourne un couple (*mini*, *maxi*) : positions du minimum et du maximum de la liste L si elle est non vide, dans le cas contraire elle déclenche une exception.

```
1
2            def posMiniMaxi(L):
3
4                if L == []:
5                    raise Exception("empty list")
6
7                (pMini, pMaxi) = (0, 0)
8
9                for i in range(1, len(L)):
10                   if L[i] > L[pMaxi]:
11                       pMaxi = i
12                   elif L[i] < L[pMini]:
13                       pMini = i
14
15               return (pMini, pMaxi)
```

Solution 5 **(Tri fusion – *2,5 + 5 + 2,5 points*)**

1. **Spécifications :**
   La fonction `partition (L)` sépare la liste L une liste en deux listes de longueurs quasi identiques (à 1 près) : une moitié dans chaque liste.

```
def partition(L):

    n = len(L)
    L1 = []
    for i in range(0, n//2):
        L1.append(L[i])

    L2 = []
    for i in range(n//2, n):
        L2.append(L[i])

    return (L1, L2)
```

2. **Spécifications :**
   La fonction `merge(`$L1$`, `$L2$`)` fusionne deux listes $L1$ et $L2$ triées en ordre croissant en une seule liste triée.

```
def merge(L1, L2):

    R = []
    i = j = 0
    n1 = len(L1)
    n2 = len(L2)

    while (i < n1) and (j < n2):
        if L1[i] <= L2[j]:
            R.append(L1[i])
            i = i+1
        else:
            R.append(L2[j])
            j = j+1

    for i in range(i, n1):
        R.append(L1[i])
    for j in range(j, n2):
        R.append(L2[j])

    return R
```

3. **Spécifications :**
   La fonction `mergesort(`L`)` trie la liste L en ordre croissant (pas en place : la fonction construit une nouvelle liste qu'elle retourne).

```
def mergesort(L):

    if len(L) <= 1:
        return L

    else:
        (L1, L2) = partition(L)

        return merge(mergesort(L1), mergesort(L2))
```