Corrigé X-ENS 2007

Informatique tronc commun 2ème année

1)

```
def occurences(t,n) :
    rep = [0 for i in range (256)]
    for pos in range(n) :
        rep[t[pos]] += 1
    return rep
```

2)

```
def min(t,n) :
    occ = occurences(t,n)
    occ_min = len(t)
    ent_min = -1
    for i in range (256) :
        if occ[i] < occ_min :
            occ_min = occ[i]
            ent_min = i
    return ent_min
```

3)

```
def tailleCodage(t,n) :
    taille = 1
    pos = 0
    while pos < n :
        val = t[pos]
        continu = 1
        pos += 1
        while pos < n and t[pos] == val :
            continu += 1
            pos += 1
        if continu == 1 :
            taille += 1
        else :
            taille += 3
    return taille
```

4)

```
def codage(t,n) :
    diese = min(t,n)
    rep = [diese]
    pos = 0
    while pos < n :
        val = t[pos]
        continu = 1
        pos = pos + 1
        while pos < n and t[pos] == val :
            continu += 1
            pos += 1
        if continu == 1 :
            rep.append(val)
        else :
            rep.append(diese)
            rep.append(continu - 1)
            rep.append(val)
    return rep
```

5)

```
def comparerRotations(t,n,i,j) :
    for k in range (n) :
        ind_i = (i + k) % n
        ind_j = (j + k) % n
        if t[ind_i] > t[ind_j] :
            return 1
        elif t[ind_j] > t[ind_i] :
            return -1
    return 0
```

6)

```
def codageBW(t,n) :
    tri_rot = triRotations(t,n)
    rep = []
    cle = -1
    for pos in range(n) :
        elem = tri_rot[pos][-1]
        if tri_rot[pos] == 1 :
            cle = pos
        rep.append(elem)
    rep.append(cle)
    return rep
```

7)
L'appel à la fonction `triRotations(t,n)` est en $O(n^2 ln(n))$ car la fonction fait $nln(n)$ appels à une fonction en $O(n)$.
La boucle s'effectue n fois. Les opérations à l'intérieur de la boucle sont en $O(1)$. Donc la complexité de la boucle est en $O(n)$.
La complexité est donc en $O(n^2 ln(n) + n) = O(n^2 ln(n))$.

8)

```
def frequences(tp,np) :
    return occurences(tp,np - 1)
```

9)

```
def triCarDe(tp,np) :
    occ = frequences(tp,np)
    tab = []
    for i in range(256) :
        for j in range (occ[i]) :
            tab.append(i)
    return tab
```

C'est en temps linéaire car la somme des occurrences est égal à n.

Pour la fonction suivante, il y a deux fonctions proposées. Une fonction optimisée, et une fonction raisonnablement trouvable ;

10)

```
def trouverIndices(tp,np) :
    dico = {}
    sec_l = triCarDe(tp,np)
    for i in range (np - 1) :
        if not tp[i] in dico :
            dico[tp[i]] = []
        dico[tp[i]].append(i)

    rep = [-1 for i in range(np-1)]
    for i in range (np - 2,-1,-1) :
        rep[i] = dico[sec_l[i]].pop()
    return rep
```

Cette version s'effectue en $O(np)$.

```
def trouverIndices(tp,np) :
    dico = {}
    sec_l = triCarDe(tp,np)
    rep = [-1 for i in range(np - 1)]
    for i in range (np - 1) :
        if not sec_l[i] in dico :
            dico[sec_l[i]] = 0
        a_ignorer = dico[sec_l[i]]
        for j in range (np - 1) :
            if tp[j] == sec_l[i] :
                if a_ignorer == 0 :
                    rep[i] = j
                    dico[sec_l[i]] += 1
                    break
                else :
                    a_ignorer += -1
    return rep
```

Il y a deux boucles qui s'effectue en $O(np)$. Cette deuxième version s'effectue donc en $O(np^2)$.

11)

```
def decodageBW(tp, np) :
    indices = trouverIndicesOpt(tp,np)
    rep = []
    pos = tp[-1]
    while len(rep) != np - 1 :
        rep.append(tp[pos])
        pos = indices[pos]
    return rep
```

La complexité de la boucle est en O(np). La complexité dépend donc de la fonction précédente, c'est la même.