université de BORDEAUX	**ANNEE UNIVERSITAIRE 2016 / 2017** SESSION 2 DE PRINTEMPS **PARCOURS : L1 Maths/Info** **Code UE : 4TPM205** **Date : 03/07/2017** **Heure : 11h30** **Durée :** 1h30 **Documents :** non autorisés. **Epreuve de Mme :** C. Blanc. **Consignes :** Ce sujet comporte 8 pages d'exercices et 2 pages d'annexe avec les codes des algorithmes de tri de tableau. Vous pouvez détacher l'annexe mais ne séparez pas les autres pages. Vous devez **répondre directement sur le sujet**, insérez ensuite votre réponse dans une copie d'examen comportant tous les renseignements administratifs. **Indiquez votre numéro d'anonymat sur chaque feuille du sujet.**	Collège Sciences et Technologies

Numéro d'anonymat :

Questions de cours : (3 pts)

1. On souhaite appliquer l'algorithme de recherche dichotomique sur un tableau `t` d'entiers. Quelle propriété doit vérifier le tableau `t` pour permettre l'application de cet algorithme. Quelle sera la complexité en temps de cette recherche si `t` contient `n` éléments ?

2. Soit t =[7,2,13,9,8,15,3,1]. Lorsqu'on applique le tri sélection sur t pour trier le tableau selon l'ordre croissant, quels sont les deux premiers éléments déplacés ? Justifiez

3. Soit (r,g,b) la couleur d'un pixel. Si les 3 composantes de ce triplet sont identiques alors le pixel sera de couleur grise. Quelle condition permet de vérifier qu'un pixel n'est pas gris ?

Exercice 1 : Minimum (6,5 pts)

1. Ecrivez une fonction `minimumSansTri(t,n)` qui calcule et retourne la valeur minimum des `n` éléments d'un tableau `t` non nécessairement trié. Quelle est sa complexité ?

Si `t` est un tableau d'entiers trié en ordre croissant alors la valeur minimum de ses éléments se trouve dans la première case du tableau. Pour trouver le minimum, une autre méthode consiste donc à trier le tableau `t` avec l'un des algorithmes de tri vus en cours puis à retourner l'élément « au début » du tableau trié. Pour être plus efficace on propose d'arrêter l'algorithme de tri dès que l'élément du début du tableau est à sa place définitive.

2. Pour trier les éléments de `t` on vous propose le choix entre l'algorithme de tri sélection, de tri insertion ou de tri rapide. Lequel de ces trois algorithmes **ne pourra-t-on pas interrompre** avant la fin du tri complet pour retourner la valeur minimum plus rapidement ? Justifiez

3. Pour écrire la fonction `minimumAvecTri(t,n)` il suffit donc d'adapter **les deux autres algorithmes** de tri pour les interrompre dès que le minimum est à sa place définitive. En modifiant les codes fournis en annexe donnez deux versions de la fonction `minimumAvecTri` (il n'est pas utile de réécrire le code non modifié).

Version 1 :

Numéro d'anonymat :

Version 2 :

4. Pour chaque version produite indiquez sa complexité. Justifiez.

5. Laquelle de ces 2 versions produit un algorithme similaire à celui de la fonction `minimunSansTri` de la question 1 ? Justifiez.

Exercice 2 : Récursivité (3pts)

On considère les fonctions suivantes :

```
def mystereRec(t,n,d):
  if (d==n):
    return True;
  if (t[d]*t[d-1] != 0):
    return False;
  return mystereRec(t,n,d+1)
```

```
def mystere(t,n):
  return mystereRec(t,n,1)
```

1. Si `v1=[5,0,-3,0,2,0]`, quel sera le résultat des appels `mystere(v1,6)` ? Justifiez

2. Si `v2=[1,0,0,-7,0,-2]`, quel sera le résultat de l'appel `mystere(v2,6)` ? Justifiez.

3. Si `v3=[1,0,3,-4,7,-2]`, quel sera le résultat de l'appel `mystere(v3,6)` ? Justifiez.

4. Quelle propriété possède un tableau `t` pour lequel `mystere(t,n)` retourne `True` ?

Numéro d'anonymat :

Exercice 3 : Tri de tableau (3,5 pts)

1. On applique le tri insertion au tableau `t=[12,3,5,10,2,6,18,4]`. Comment sera modifié `t` après quatre itérations ? Justifiez.

2. Sur le tableau `t` modifié par la question précédente appliquez le tri fusion. Vous détaillerez l'exécution du tri fusion par exemple en dessinant l'arbre des appels.

3. Est-ce que le fait d'appliquer le tri sélection au tableau `t` avant de lui appliquer le tri fusion améliore l'efficacité du tri fusion ? Justifiez.

Exercice 4 : (4 points).

On considère la fonction `mystery` suivante :

```
def mystery(t, n, x):
  i=0
  j=n-1
  while(i < j):
    if (t[i] == x):
      i=i+1
    else :
      echanger(t,i,j)
      j=j-1
```

1. Soit `t1=[1,0,2,2,0,2,3,1,3,2]` un tableau de `10` entiers. Détaillez l'application de la fonction `mystery` au tableau `t1` pour `n=10` et `x=1` . Quel sera le contenu de `t1` à la fin de l'exécution de cet appel ?

2. Que fait la fonction `mystery` dans le cas général ?

Numéro d'anonymat :

3. On souhaite écrire une fonction `ordre(t,n,x)` qui, appliquée à un tableau `t` de `n` entiers, déplace les éléments de `t` de telle façon que :

   - tous les éléments **inférieurs** à `x` sont placés au début du tableau `t`
   - tous les éléments **supérieurs** à `x` sont placés à la fin du tableau `t`
   - entre ces 2 zones on place tous les éléments égaux à `x`.

   Par exemple si `t=[0,4,2,3,1,0,2,1,0,3,4,3,1,2,0]` et `x=3` après l'appel `ordre(t,15,3)` on aura `t=[0,0,2,1,0,2,1,0,2,1,3,3,3,4,4]`.

Chaque élément de `t` ne sera observé **qu'une seule fois** au cours de l'exécution de la fonction `ordre`.

3.1. Dans quelles conditions la fonction `ordre` produira-t-elle **moins** de 3 zones différentes dans le tableau `t` modifié ?

3.2. **Sans écrire de fonctions** décrivez comment la fonction `ordre` pourrait être utilisée pour construire un algorithme récursif de tri d'un tableau t d'entiers.

3.3 Quelle différence existe-t-il entre votre algorithme et l'algorithme du tri rapide vu en cours ?

FIN

Annexe : t est un tableau d'entiers de taille n.

Tri sélection :

```
1  def triSelection (t,n):
2    for  i in range(n) :
3      iMin=i
4      for j in range(i+1,n) :
5        if(t[j]<t[iMin]):
6          iMin=j
7      if(i != iMin):
8        echanger(t,i,iMin)
```

Tri insertion :

```
1  def triInsertion(t, n):
2    for i in range(1,n):
3      e=t[i]
4      j=i-1
5      while (j>=0 and t[j]>e):
6        t[j+1]=t[j]
7        j=j-1
8      t[j+1]=e
```

Tri à bulle :

```
1  def triBulle(t,n):
2    for i in range(n-1,0,-1) :
3      for j in range(i) :
4        if(t[j]>t[j+1]) :
5          echange(t,j,j+1)
```

Tri Fusion :

```
1  def fusion(t, d, f):
2    r=creerTableau(f-d)
3    m=(d+f)//2
4    i1=d
5    i2=m
6    k=0
7    while (i1<m and i2<f):
8      if (t[i1] < t[i2]):
9        r[k] = t[i1]
10       i1=i1+1
11     else :
12       r[k] = t[i2]
13       i2=i2+1
14     k=k+1
15   while (i1 <m):
16     r[k] = t[i1]
17     i1=i1+1
18     k=k+1
19   while (i2 <f):
20     r[k] = t[i2]
21     i2=i2+1
22     k=k+1
23   for k in range(f-d):
24     t[d+k]=r[k]
```

```
1  def triFusion (t,d,f):
2    if (d<f-1):
3      m=(d+f)//2
4      triFusion(t,d,m)
5      triFusion(t,m,f)
6      fusion(t,d,f)
```

1er Appel :

```
triFusion(t,0,n)
```

Tri rapide :

```
1  def separationPivot(t,n, d, f):
2    p = d
3    i = d
4    j = f
5    while(i<=j):
6      if (t[i]<t[p]):
7        i=i+1
8      else :
9        echanger(t,i,j)
10       if (i==p):
11         p=j
12       else :
13         if (j==p):
14           p=i
15       j=j-1
16   echanger (t, i, p)
17   return (i)
```

```
1  def triRapideRec(t, n, d, f):
2    if(d<f):
3      p=separationPivot(t,n,d,f)
4      triRapideRec(t,n,d,p-1)
5      triRapideRec(t,n,p+1,f)

   def triRapide(t, n):
     triRapideRec(t,n,0,n-1)
```