| | | |
|---|---|---|
| université de BORDEAUX | **ANNEE UNIVERSITAIRE 2016 / 2017**<br>SESSION 1 DE PRINTEMPS<br><br>**PARCOURS : L1 Maths/Info** **Code UE : 4TPM205**<br><br>**Date : 17 mai 2017** **Heure : 9h00** **Durée :** 1h30<br>**Documents :** non autorisés.<br>**Epreuve de Mme :** C. Blanc.<br><br>**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.<br>**Indiquez votre numéro d'anonymat sur chaque feuille du sujet.** | Collège Sciences et Technologies |

Numéro d'anonymat :

**Questions de cours : (3 pts)**

1. Ecrire une fonction `elementAleatoire(t,n)` qui prend en entrée un tableau t de n entiers et retourne aléatoirement un élément du tableau t.

Comme discuté en cours n'importe quel élément de t peut être retourné car nous n'avons aucune information sur le contenu du tableau.

```
def elementAleatoire(t,n) :
    return t[0]
```

Autre réponse acceptée :

```
def elementAleatoire (t,n) :
   return (t[alea(0,n)])
```

Où « alea » est une fonction qui retourne aléatoirement un entier entre 0 et n-1.

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

Les appels récursifs successifs s'empilent jusqu'à atteindre le tableau t[0]=[7] déjà trié car réduit à un seul élément puis le tableau t[1]=[2] déjà trié pour la même raison. Le premier appel à fusion se fait donc avec ces 2 tableaux et le résultat sera [2,7]. t[0] et t[1] sont donc les 2 premiers éléments de t déplacés lors du triFusion.

3. Soit (r,g,b) la couleur d'un pixel. Si les 3 composantes de ce triplet sont égales à 0 le pixel est de couleur noire. Si les 3 composantes de ce triplet sont égales à 255 le pixel est de couleur blanche. Quelle condition permet de vérifier qu'un pixel n'est ni noir ni blanc ?

Plusieurs réponses sont possibles par exemple :

```
1. not((r,g,b)==(0,0,0) or (r,g,b)==(255,255,255))
2. (r,g,b)!=(0,0,0) and (r,g,b)!=(255,255,255)
3. not((r==0 and g==0 and b==0)or (r==255 and g==255 and b==255))
4. not((r==0 and g==0 and b==0)) and not((r==255 and g==255 and b==255))
5. (r!=0 or g!=0 or b!=0)and (r!=255 or g!=255 or b !=255)
```

**Exercice 1 : Médiane (4 pts)**

**Rappel définition**: Dans un ensemble E d'entiers la **valeur médiane m** est celle qui permet de séparer l'ensemble en deux demi-ensembles (de taille identique à plus ou moins 1 près) tels que le premier demi ensemble contient les valeurs inférieures ou égales à m et le second celles qui sont supérieures ou égale m.

Lorsque les n valeurs de l'ensemble sont stockées dans un tableau trié la valeur médiane se trouve dans la case du « milieu » c'est-à-dire celle d'indice `n//2` (à plus ou moins un près selon la parité de `n`).

Dans cet exercice on considèrera que **si** un tableau `t` de `n` entiers est trié la médiane de ses éléments est `t[n//2]`. On considèrera de plus que les éléments de t sont distincts deux à deux.

Soit `t` un tableau de `n` entiers non nécessairement trié. On souhaite écrire une fonction `mediane(t,n)` qui retourne la valeur médiane des éléments de t. Une méthode évidente consiste à trier le tableau avec l'un des algorithmes de tri vu en cours puis à retourner l'élément « au milieu » du tableau trié. Pour être plus efficace on propose d'arrêter l'algorithme de tri dès que l'élément du milieu du tableau est à sa place définitive.

1. Pour trier les éléments de `t` on a le choix entre l'algorithme de tri sélection, de tri fusion ou de tri rapide. Lequel de ces algorithmes ne pourra-t-on pas interrompre avant la fin du tri complet pour retourner la valeur médiane plus rapidement ? Justifiez

Il s'agit du triFusion pour lequel on ne peut pas savoir avant le dernier appel à fusion si un élément est à sa place définitive ou non. Par exemple si t=[5,8,6,9,2,4,1,3] juste avant la dernière fusion on aura t=[5,6,8,9,1,2,3,4] et ce n'est que lors du dernier appel à fusion que les éléments seront mis définitivement à leur place. Il faut donc que l'algorithme se termine avant qu'on puisse être sûr que la valeur médiane est en t[n//2]

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

**Version 1 :**

On va modifier le triSelection. L'idée consiste à faire tourner l'algorithme original pendant (n//2 + 1), itérations ainsi on placera les (n//2 +1) premiers minimums à leur place:

```
def mediane (t,n):
  for  i in range(n//2+1) :
    iMin=i
    for j in range(i+1,n) :
      if(t[j]<t[iMin]):
        iMin=j
    if(i != iMin):
       echanger(t,i,iMin)
   return(t[n//2])
```

Dans cet exercice une erreur fréquente mais mineure consiste à écrire range(n//2) sur la ligne 2. De fait la case t[n//2] n'est donc pas mise à jour et rien ne garantit qu'elle contient la valeur cherchée. Il faut alors retourner t[n//2-1]

Une erreur fréquente **importante** consiste à écrire range(i+1,n//2) sur la ligne 4. Dans ce cas on ne recherche pas les minimums successifs parmi tous les candidats possibles mais parmi la moitié d'entre eux. La réponse est donc fausse.

**Version 2 :**
On va modifier le triRapide. A chaque itération cet algorithme place un élément à sa place définitive et sépare les autres éléments autour du pivot p. Les éléments inférieurs à t[p] sont placés avant lui dans le tableau et les supérieurs après lui.
Si le pivot p est égal à n//2 alors on a trouvé la médiane c'est t[p].
Si le pivot est supérieur à n//2 il faut continuer sur les éléments inférieurs au pivot pour trouver la valeur médiane. Si le pivot est inférieur à n//2 il faut continuer sur les éléments supérieurs au pivot pour trouver la valeur médiane.

```
def medianeRec(t, n, d, f):
    if(d<f):
        p=separationPivot(t,n,d,f)
        if p==(n//2):
            return t[p]
        if p>n//2:
            return medianeRec(t,n,d,p-1)
        return medianeRec(t,n,p+1,f)

def mediane(t, n):
    return medianeRec(t,n,0,n-1)
```

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

Version 1 : La complexité de cet algorithme vient des 2 boucles imbriquées via lesquelles on fera : (n-1)+(n-2)+…+(n//2-1) opérations élémentaires (des comparaisons essentiellement). Donc la complexité est de l'ordre de $n^2$ comme celle de l'algorithme de triSelection.

Version 2 : Au mieux dès le premier appel p==n//2 la complexité vient de l'unique appel à la fonction séparation c'est O(n).
Au pire le pivot est toujours un extrémum et on convergera vers n//2 par pas de 1 la complexité sera $O(n^2)$. Comme pour l'algorithme de tri rapide en moyenne on aura $O(n\log_2(n))$.

Numéro d'anonymat :

**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=[5,1,3,8,2,9]`, quel sera le résultat des appels `mystere(v1,6)` ? Justifiez

| n | d | d==n | t[d] | t[d-1] | t[d]*t[d-1] > 0 | Sortie |
|---|---|---|---|---|---|---|
| 6 | 1 | False | 1 | 5 | True | False |
| | | | | | | |
| | | | | | | |
| | | | | | | |

2. Si `v2=[1,-8,3,-4,7,-2]`, quel sera le résultat de l'appel `mystere(v2,6)` ? Justifiez.

| n | d | d==n | t[d] | t[d-1] | t[d]*t[d-1] > 0 | Sortie |
|---|---|---|---|---|---|---|
| 6 | 1 | False | -8 | 1 | False | |
| | 2 | False | 3 | -8 | False | |
| | 3 | False | -4 | 3 | False | |
| | 4 | False | 7 | -4 | False | |
| | 5 | False | -2 | 7 | False | |
| | 6 | True | | | | True |

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

| n | d | d==n | t[d] | t[d-1] | t[d]*t[d-1] > 0 | Sortie |
|---|---|---|---|---|---|---|
| 6 | 1 | False | 0 | 1 | False | |
| | 2 | False | 3 | 0 | False | |
| | 3 | False | -4 | 3 | False | |
| | 4 | False | 7 | -4 | False | |
| | 5 | False | -2 | 7 | False | |
| | 6 | True | | | | True |

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

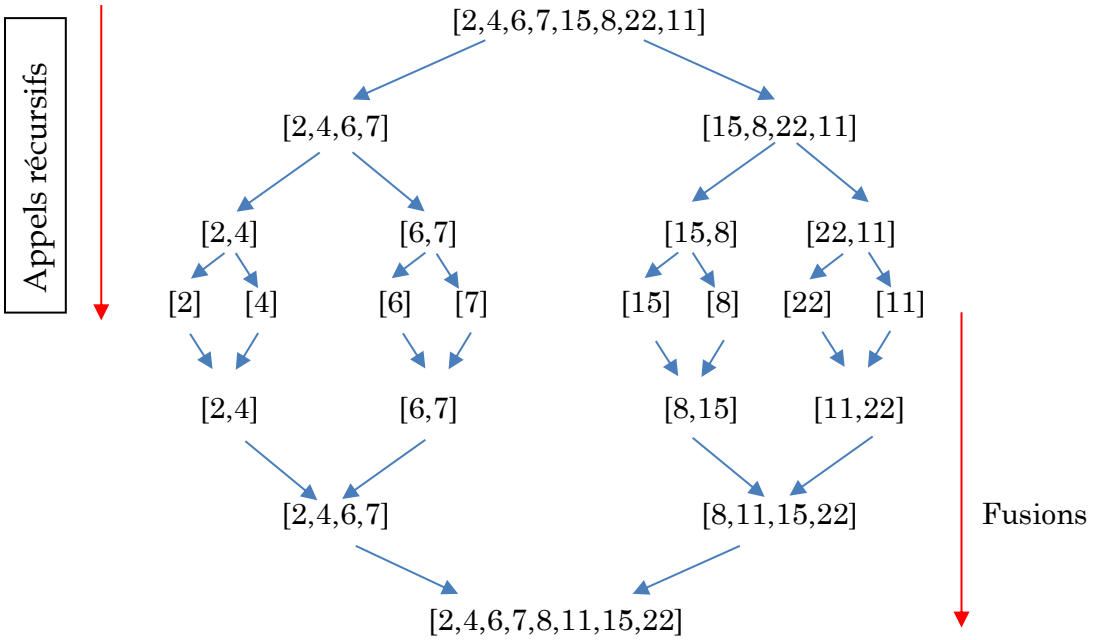
Deux éléments consécutifs non nuls dans le tableau ne sont pas de même signe.

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

1. On applique le tri sélection au tableau `t=[15,6,8,11,2,4,22,7]`. Comment sera modifié `t` après quatre itérations ?

```
Itération 1 : t=[2,6,8,11,15,4,22,7]
Itération 2 : t=[2,4,8,11,15,6,22,7]
Itération 3 : t=[2,4,6,11,15,8,22,7]
Itération 3 : t=[2,4,6,7,15,8,22,11]
```

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.

Non. Le tri fusion se déroule sans vérifier si les données observées sont triées avant d'arriver à une zone d'observation réduite à une case. Donc l'efficacité n'est pas améliorée par l'application du tri insertion.

Numéro d'anonymat :

**Exercice 4 : (6,5 pts)**

On considère la fonction `mystery1(t,n,h,k)` qui prend en entrée deux tableaux `t` et `h` d'entiers positifs ou nuls. `t` contient `n` éléments et `h` est un tableau de taille `k`.

```
def mystery1(t, n, h, k):
 for i in range(k):
    h[i]=0
 for i in range(n):
   h[t[i]]=h[t[i]]+1
```

- Soit t=[1,0,2,2,0,2,3,1,3,2] un tableau de 10 entiers. Quel sera le contenu du tableau h après l'appel `mystery1(t,10,h,4)` ?

```
h est un tableau de 4 éléments initialisé à 0
h=[2,2,4,2]
```

- Quelle condition doit être vérifiée par k pour que la fonction ne produise pas une erreur ?

```
k doit être supérieur au maximum des éléments de t.
```

- Que fait la fonction `mystery1` dans le cas général ?

```
La fonction mystery1 construit l'histogramme du tableau t. C'est-à-dire que
pour une case i de h, la valeur h[i] représente le nombre d'occurrences de
i dans t.
Par exemple dans t il y a deux 0 donc h[0]==2 , il y a quatre 2 donc
h[2]==4, etc.
```

Numéro d'anonymat :

- Modifiez la fonction `mystery1` pour écrire la fonction `nbInferieurs(t,n,h,k)` qui prend en entrée les mêmes données que `mystery1` et modifie le tableau `h` de façon à ce que pour tout i ∈[0,k-1], `h[i]` représente le nombre d'éléments de `t` inférieurs ou égaux à `i`. Par exemple si t=[1,0,2,2,0,2,3,1,3,2] après l'appel `nbInferieurs(t,10,h,4)` on aura h=[2,4,8,10].

```
def mystery1(t, n, h, k):
 for i in range(k):
    h[i]=0
 for i in range(n):
   h[t[i]]=h[t[i]]+1
 for i in range (1,k):
   h[i]=h[i]+h[i-1]
```

- Quelle est la complexité de cette fonction ? Justifiez votre réponse.

Cette fonction exécute 3 boucles séparées, deux sur le tableau h et une sur le tableau t donc la complexité est de l'ordre de O(n+k).

- Dans cette question on utilise la fonction `nbInferieurs` qui fonctionne comme expliqué à la question 2 même si vous n'avez pas su l'écrire.

  On considère la fonction `enigma(t,n,p)` qui prend en entrée un tableau `t` de `n` entiers positifs ou nuls et un entier `p`.

```
def enigma(t,n,p) :
  d=creerTableau(p,0)          #temps de calcul p
  r=creerTableau(n,0)          #temps de calcul n
  nbInferieurs(t,n,d,p)        #temps de calcul n+p
  for i in range(n-1,-1,-1): #temps de calcul n
      r[d[t[i]]-1]=t[i]
      d[t[i]]=d[t[i]]-1
  for i in range(n):           #temps de calcul n
      t[i]=r[i]
```

4.1. Soit t=[1,0,2,2,0,2,3,1,3,2] un tableau de 10 entiers. Quel sera le contenu du tableau t après l'appel `enigma(t,10,4)` ?

t=[0,0,1,1,2,2,2,2,3,3]

4.2. Que fait la fonction `enigma` dans le cas général ? Précisez le rôle du paramètre `p`.

La fonction enigma trie le tableau t il s'agit de l'algorithme de tri par dénombrement.

4.3. Quelle est la complexité de cette fonction ? Justifiez.

La complexité est O(n+p)
Voir l'algorithme pour les détails.

4.4. Donnez un avantage et un inconvénient de cette méthode par rapport à d'autres méthodes pour produire le même résultat.

Avantage : temps de calcul est linéaire (n+p) donc cette méthode est beaucoup plus efficace que les algorithmes vus en cours.

Inconvénients :

- Cette méthode s'applique aux tableaux d'entiers seulement.
- On doit connaitre la valeur max de t pour définir d.
- Plus cette valeur maximum est petite et plus la méthode est efficace. Au contraire si le maximum est grand la taille du tableau d sera grande aussi donc la mémoire sera encombrée.

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)
```