université de BORDEAUX	**ANNEE UNIVERSITAIRE 2018/2019** SESSION 1 DE PRINTEMPS **PARCOURS : L1 Maths/Info** **Code UE : 4TPM205** **Date : 17 mai 2019** **Heure : 9h00** **Durée :** 1h30 **Documents :** non autorisés. **Epreuve de Mme :** C. Blanc. **Consignes :** Ce sujet comporte 8 pages d'exercices (dont une blanche) 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 :

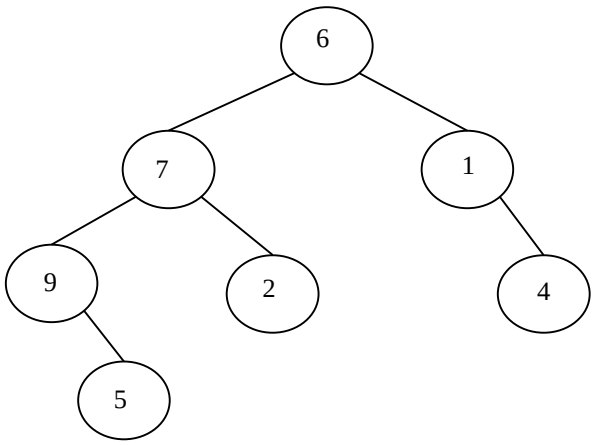
Questions de cours :

Un arbre binaire d'entiers **strictement positifs** est une structure de données hiérarchisée dans laquelle chaque objet est en relation avec au plus 3 autres objets de la structure quand ils existent : un père, un fils gauche, un fils droit.

Pour stocker un arbre binaire dans un tableau on applique les règles suivantes :
Pour un sommet s de l'arbre stocké dans la case d'indice i :

- **Son fils gauche**, s'il existe, est stocké dans la case d'indice 2*i. Sinon la case d'indice 2*i contient -1
- **Son fils droit** s'il existe est stocké dans la case d'indice 2*i +1. Sinon la case d'indice 2*i +1 contient -1
- **Son père** est stocké dans la case d'indice i//2.
- **Si le sommet n'a pas de père** on dit que c'est la racine de l'arbre et il est stocké dans la case d'indice 1.
- **Si un sommet n'a pas de fils** on dit que c'est une feuille.
- **Les sommets de valeur 0** ne sont pas des sommets significatifs pour l'arbre binaire.

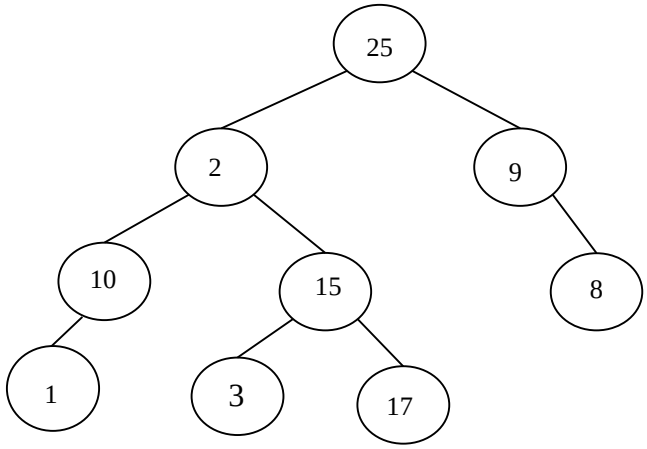
Par exemple, le tableau `t1` de taille `n1` représente l'arbre binaire A1 dont la racine est de valeur 6 :



t1	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**…**	**n1-1**
	0	6	7	1	9	2	-1	4	-1	5	-1	-1	0	0	-1	-1	0	0	-1	-1	0	…	0

Arbre binaire A1

1. Construisez le tableau `t2` de taille `n2` représentant l'arbre binaire `A2` ci-dessous :



2. On considère la fonction mystère suivante qui prend en entrée un tableau `t` de taille `n` représentant un arbre binaire :

```
def mystere(t,n):
  cpt= 0
  i=1
  while(i<=n//2) :
    if(t[2*i]== -1 and t[(2*i)+1]== -1) :
      cpt = cpt+1
    i=i+1
  return cpt
```

2.1. Quelle valeur sera retournée par l'appel `mystere(t1,n1)` ? Justifiez.

2.2. Quelle valeur sera retournée par l'appel `mystere(t2,n2)` ? Justifiez.

2.3. Dans le cas général que retourne `mystere(t,n)` lorsque t représente un arbre binaire ? Justifiez.

Numéro d'anonymat :

Exercice 1 : Récursivité

Soit t un tableau de n éléments, on considère la fonction suivante :

```
def mystere(t,n,k):
  if n==0:
     return True
  return(t[n-1]%k==0 and mystere(t,n-1,k))
```

1. Si `v1=[14,0,7,84,35,49]`, quel sera le résultat de l'appel `mystere(v1,6,7)` ? Justifiez.

2. Si `v2=[64,32,8,20,16]`

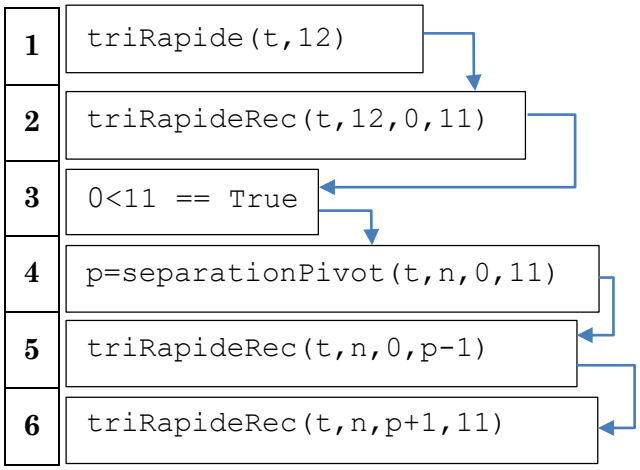
Quel sera le résultat de l'appel `mystere(v2,5,8)` ? Justifiez.

Quelles valeurs de k>0 fourniraient une réponse différente pour le tableau v2 ? Justifiez.

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

Exercice 2 : Récursivité

Soit `t` un tableau d'entiers de taille `12`. `t=[10,2,7,3,1,9,12,24,5,13,6,4]`
Pour trier ce tableau on choisit d'appliquer l'algorithme du tri rapide. Les fonctions qui s'exécuteront sont donc :



On suppose que l'algorithme s'exécute normalement lors des étapes `N°1,2,3,4 et 5` **mais qu'il est interrompu** brutalement avant de réaliser l'étape `N°6`.

En justifiant soigneusement votre réponse dites comment sera modifié le tableau t après cette interruption.

Exercice 3 : Tri de tableau

1. Soit `w=[5,8,1,45,22,7,3,11]` un tableau de 8 éléments. En détaillant les étapes, appliquez le tri à bulle sur le tableau w.

2. Appliquez le tri fusion sur le tableau w. Vous détaillerez l'exécution du tri fusion par exemple en dessinant l'arbre des appels et en indiquant l'ordre des appels.

Numéro d'anonymat :

Exercice 4 : Majorité absolue

On dit qu'un entier `x` est un élément majoritaire dans un tableau `t` de `n` entiers si le nombre de fois (noté `NB_x`) où `x` est présent dans `t` est supérieur strictement à `n//2.` (`NB_x > n/2`).
Par exemple :

- si `ta=[1,1,1,1,1,1,2,2,3],1` est l'élement majoritaire dans `ta`.
- si `tb=[1,1,2,2,2,2,3,3]` il n'y a pas d'élement majoritaire dans `tb`.
- si `tc=[1,2,3,4]` il n'y a pas d'élement majoritaire dans `tc`.

On souhaite écrire une fonction `majoritaire(t,n)` qui prend en entrée un tableau **`t` trié de `n` entiers** et retourne un entier i tel que :

- Si `0<=i<n` alors `t[i]` représentera un des éléments majoritairement présent dans `t`
- Si `i == -1` alors aucun élément n'est majoritaire dans t.

1. En remarquant que dans un tableau trié tous les éléments de même valeur sont consécutifs écrivez la fonction majoritaire en complétant le code suivant :

```
1 def majoritaire(t,n):
2   cpt=1
3   i=1
4   e=t[0]
5   while(          ): #definir la zone à explorer dans le tableau
6       if t[i]==e: #des elements consecutifs sont egaux
7
8
9
10      else :
11
12
13
14  return -1
```

2. Sans écrire d'algorithme décrivez une méthode permettant de connaitre l'élément majoritaire d'un tableau t de n entiers non trié. Donnez la complexité en temps de votre méthode et ses éventuels défauts.

3. Modifiez la fonction `majoritaire` pour écrire la fonction `sontTousDifferents(t,n)` qui prend en entrée un tableau d'entiers triés et retourne `True` si tous les éléments de `t` sont différents et `False` sinon.

Annexe : t est un tableau d'entiers de taille n. Les algorithmes produisent un tri croissant.

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          echanger(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)
```