université de BORDEAUX	**ANNEE UNIVERSITAIRE 2015 / 2016** SESSION 1 DE PRINTEMPS **PARCOURS : L1 Maths/Info** **Code UE : J1MI2013** **Date : 18 mai 2016** **Heure : 8h30** **Durée :** 1h30 **Documents :** non autorisés. **Epreuve de Mme :** C. Blanc. **Consignes :** Vous devez répondre directement sur le sujet qui comporte 8 pages. Insérez ensuite votre réponse dans une copie d'examen comportant tous les renseignements administratifs.	Collège Sciences et Technologies

Numéro d'anonymat :

Questions de cours : (3 points)

1. Écrire la fonction `echanger` qui échange le contenu de deux cases d'un tableau d'entiers. Quelle est sa complexité ?

```
void echanger (int t[], int i, int j){
     int tmp;
     tmp=t[i];
     t[i]=t[j];
     t[j]=tmp;
}
```

2. Lorsque pour trier un tableau d'entiers en ordre croissant on applique l'algorithme du **tri à bulles** et qu'on arrête l'exécution de l'algorithme après `k` itérations `(1 <= k < n)`, combien d'éléments se trouveront à leur place définitive (et ne seront plus considérés si on reprend l'exécution) ? Justifiez votre réponse.

Dans l'algorithme décrit en cours à chaque itération on sélectionne l'élément d'indice 0 (la bulle) et, par échanges successifs, on le fait « remonter » vers la droite du tableau si il est plus grand que son voisin de droite. Sinon c'est le voisin de droite qui devient la bulle et continue la remontée. Après une itération complète l'élément le plus grand du tableau est à sa place. On répète ce procédé avec les n-1 éléments restants. Au bout de k itérations k éléments sont à leur place définitive (les k plus grands) et ne seront plus considérés dans la suite de l'algorithme.

3. Quand on utilise l'algorithme de recherche dichotomique combien d'étapes au maximum sont nécessaires pour trouver un entier `x` parmi `1000` entiers ordonnés ? Justifiez.

Le cas le pire pour la recherche dichotomique est celui où l'élément cherché n'est pas présent parmi les données. La complexité au pire pour n données est $log_2(n)$. Avec n =1000 on a $log_2(1000)$ soit 10 étapes.

Exercice 1 : Les booléens. (2,5 points)

Dans un restaurant les clients peuvent commander une entrée, un plat ou un dessert.
Pour enregistrer les commandes de `n` clients on utilise trois tableaux de `n` booléens :

- `bool E[]` pour stocker les entrées commandées. Si le client `i` commande une entrée `E[i]`sera égal à `true` sinon `E[i]` sera égal à `false`.
- `bool P[]` pour stocker les plats commandés. Si le client `i` commande un plat `P[i]`sera égal à `true` sinon `P[i]` sera égal à `false`.
- `bool D[]` pour stocker les desserts commandés. Si le client `i` commande un dessert `D[i]`sera égal à `true` sinon `D[i]` sera égal à `false`.

La synthèse des commandes se fera dans un tableau `bool C[]` qui sera initialisé par la fonction `commande`.
On suppose que les tableaux. `E`, `P` et `D` sont initialisés avant l'appel de la fonction `commande`.

1. Ecrivez la fonction `void commande(bool C[], bool E[],bool P[], bool D[], int n)` qui remplit le tableau `C` de telle façon que chaque élément `C[i]` soit égal à `true` si le client `i` a commandé une entrée, un plat et un dessert et `false` sinon.

```
/*version 1*/
void commande(bool C[], bool E[],bool P[], bool D[], int n){
  for(int i=0;i<n;i++){
    if (E[i]==true && P[i]==true && D[i]==true)
      C[i]=true;
    else
      C[i]=false;
  }
  }
/* version 2*/
void commande(bool C[], bool E[],bool P[], bool D[], int n){
  for(int i=0;i<n;i++){
        C[i]=(E[i] && P[i] && D[i]);
  }
```

2. Modifiez la fonction `commande` afin que `C[i]` soit égal à `true` si le client `i` a commandé une entrée et un dessert mais **pas** de plat et `false` sinon

```
/*version 1*/
void commande(bool C[], bool E[],bool P[], bool D[], int n){
  for(int i=0;i<n;i++){
    if (E[i]==true && P[i]==false && D[i]==true)
      C[i]=true;
    else
      C[i]=false;
  }
  }
/* version 2*/
void commande(bool C[], bool E[],bool P[], bool D[], int n){
  for(int i=0;i<n;i++){
        C[i]=(E[i] && !P[i] && D[i]);
  }
```

3. Modifiez la fonction `commande` afin que `C[i]` soit égal à `true` si le client `i` a commandé exactement deux éléments **dont** un plat et `false` sinon.

```
On modifie la fonction précédente
C[i]=(E[i] && P[i] && !D[i])||( !E[i] && P[i] && D[i]);
Attention :
C[i]= P[i] && (E[i] || D[i]) ne convient pas car il inclut le cas où E[i], D[i] et
P[i] sont tous les 3 vrais en même temps on n'a donc pas exactement 2 éléments.
```

Numéro d'anonymat :

Exercice 2 : Récursivité (2,5 points)

On considère les fonctions suivantes :

```
bool mystereRec(int t[], int g, int d){
  if (g >= d)
    return true;
  if (t[g]*t[d] < 0 )
    return false;
  return mystereRec(t,g+1,d-1);
}
```

```
bool mystere(int t[], int n){
  return mystereRec(t,0,n-1);
}
```

1. Si `v1=[3,-2,2,1,2,4]`, quel sera le résultat des appels `mystere(v1,6)` ? Justifiez

g	d	g<d	t[g]	t[d]	t[g]*t[d]<0
0	5	true	3	4	false
1	4	true	-2	2	true->return **false**

La fonction retourne false dans ce cas.

2. Si `v2=[1,-5,-4,7,-3,0,8]`, quel sera le résultat de l'appel `mystere(v2,7)` ? Justifiez.

g	d	g<d	t[g]	t[d]	t[g]*t[d]<0
0	6	true	1	8	false
1	5	true	-5	0	false
2	4	true	-4	-3	false
3	3	true	7	7	false
4	2	false -> return **true**			

La fonction retourne true dans ce cas.

3. Si `v3=[1,5,-4,-2,2,8]`, quel sera le résultat de l'appel `mystere(v3,6)` ? Justifiez.

g	d	g<d	t[g]	t[d]	t[g]*t[d]<0
0	5	true	1	8	false
1	4	true	5	2	false
2	3	true	-4	-2	false
3	2	False -> return **true**			

La fonction retourne true dans ce cas.

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

mystere(t,n) retourne true si le tableau t contient des entiers dont les signes sont les mêmes pour les entiers occupant des places symétriques par rapport au milieu du tableau ou bien les entiers sont nuls.
Autrement dit t[i] et t[n-1-i] pour i ∈[0,n-1] sont de même signe ou nuls

Exercice 3 : Tri de tableau (4 points)

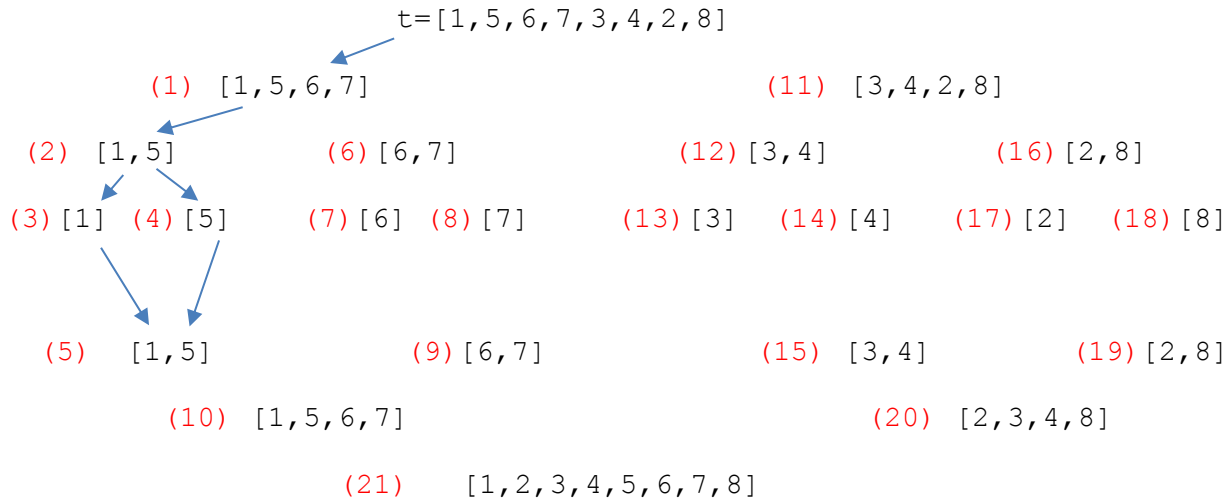
1. On applique le tri insertion au tableau `t=[7,1,6,5,3,4,2,8]`. Comment sera modifié `t` après quatre itérations ?

```
1. t=[7,1,6,5,3,4,2,8]
2. t=[1,7,6,5,3,4,2,8]
3. t=[1,6,7,5,3,4,2,8].
4. t=[1,5,6,7,3,4,2,8].
```

Voir cours.

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.

Les appels récursifs observent le tableau t sur des zones de plus en plus petites jusqu'à avoir une seule case qui sera triée. Quand on observe une seule case de t elle est triée on peut commencer la fusion. En rouge l'ordre dans lequel les actions sont faites.



3. Est-ce que le fait d'appliquer le tri insertion 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 : (8 points).

On considère la fonction `mystery` suivante :

```
void mystery(int t[], int n, int x){
  int i=0;
  int 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=2`. Quel sera le contenu de `t1` à la fin de l'exécution de cet appel ?

i	j	i<j	t[i]==x	Echange	t
0	9	true	false	oui	[2,0,2,2,0,2,3,1,3,1]
0	8	true	true	non	[2,0,2,2,0,2,3,1,3,1]
1	8	true	false	oui	[2,3,2,2,0,2,3,1,0,1]
1	7	true	false	oui	[2,1,2,2,0,2,3,3,0,1]
1	6	true	false	oui	[2,3,2,2,0,2,1,3,0,1]
1	5	true	false	oui	[2,2,2,2,0,3,1,3,0,1]
1	4	true	true	non	[2,2,2,2,0,3,1,3,0,1]
2	4	true	true	non	[2,2,2,2,0,3,1,3,0,1]
3	4	true	true	non	[2,2,2,2,0,3,1,3,0,1]
4	4	false			

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

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

La fonction `mystery` place les x au début du tableau t et les autres éléments (différents de x) à la suite.

3. On souhaite écrire une fonction `void separation(int t[], int n, int 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 `separation(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 `separation`.

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

En supposant que le tableau t contient plus de 2 valeurs différentes, la fonction `separation` produira moins de 3 zones si :

- *x est le minimum des éléments de t*
- *x est le maximum des éléments de t*
- *x n'est pas un élément de t*

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

C'est une version du quicksort x sera donc le pivot.

- *On choisit x parmi les éléments de t (par exemple x est le premier élément de t).*
- *On applique la fonction `separation` au tableau t par rapport à x.*
- *On récupère les valeurs d et f qui marquent le début et la fin de la zone contenant les x après la séparation*
- *On appelle récursivement la fonction séparation sur les éléments du tableau situé entre 0 et d-1 et entre f+1 et n-1 en choisissant pour chaque zone un nouveau pivot*
- *On ne touche pas aux éléments situés entre d et f ils sont égaux à x et sont à leur place définitive.*
- *etc*

Numéro d'anonymat :

3.3. Complétez le code de la fonction `separation` ci-dessous :

```
1. void separation (int t[], int n, int x){
2.     int i,j,k;
3.     i=0;
4.     j=0;
5.     k=n-1;
6.     while ( j<=k ){
7.       if (t[j]<x){
8.         echanger(t, j,i);
9.         j++;
10.        i++;
11.      }
12.     else{
13.       if (t[j]>x) {
14.         echanger(t,j,k);
15.         k++;
16.       }
17.       else
18.         j++ ;
19.     }
20.   }
21. }
```

3.4. Quelle est la propriété vérifiée par l'entier `i` tout au long de l'exécution de la fonction `separation` ?

i est l'indice du premier élément de t (dans l'ordre croissant des indices) qui n'est pas inférieur à x. t[i-1] == x si i-1>=0

3.5. Même question pour `j` ?

j est l'indice du premier élément de t non encore traité par l'algorithme.

3.6. Même question pour `k` ?

k est l'indice du dernier élément de t non encore traité par l'algorithme. t[k+1]> x si k+1<=n-1
Quand j>k tous les éléments de t sont traités et à leur place.

FIN