Université Denis Diderot — Partiel de TO2 – L1
2012–2013

IMPORTANT. Durée **2 heures**. Il sera tenu compte de la **clarté** et de la **qualité** de votre copie. Aucun document autorisé.

A/ Questions de cours.

I - Tri d'un tableau d'entiers. Dans cette partie, on se propose de trier dans un ordre croissant (ou décroissant) un tableau d'entiers quelconques.

1. Donner un algorithme de tri (de votre choix) et présenter son code Java en l'expliquant et en le commentant.
2. Expliquer les mots "complexité en temps dans le meilleur des cas" et "complexité en temps dans le pire cas".
3. Donner alors les complexités en temps de l'algorithme de tri de la question précédente (question A/ I- 1.).

II - Tri d'un tableau spécifique. On considère maintenant un tableau d'entiers prenant seulement deux valeurs possibles a et b ($a < b$). Dans ce qui suit, nous nous intéressons à un tri en ordre croissant de ce tableau (tous les a doivent être devant tous les b).

1. Rappeler les règles à appliquer lors de la conception d'un programme récursif.
2. On se propose maintenant d'écrire un algorithme de tri récursif. Pour cela, nous allons trier le tableau entre l'indice i et l'indice j ($0 \leq i \leq j$) et écrire une méthode récursive qu'on appellera *tri2* :

```
public static void tri2(int tab[], int i, int j){
  ...
}
```

   En utilisant les observations suivantes, compléter la méthode `tri2` (veuillez justifier et commenter votre code) :
   (i) Si les indices i et j sont égaux, le tableau est trié.
   (ii) Si l'élément d'indice i est un a alors il reste à trier entre les indices $i+1$ et j.
   (iii) Si l'élément d'indice i est un b alors on l'échange avec l'élément d'indice j et il reste à trier le nouveau tableau entre les indices i et $j-1$.

B/ Recherche d'éléments.

On suppose disposer d'un tableau d'entiers tab trié de taille n contenant tous les entiers de 1 à $n+1$ sauf un. Le but de l'exercice est d'écrire une fonction pour trouver cet entier manquant. Par exemple, pour le tableau $[1, 2, 3, 5, 6]$, il s'agit de 4.

1. Expliquer brièvement le principe de la recherche dichotomique dans un tableau et donner sa complexité.
2. En s'inspirant de la recherche dichotomique, écrire une fonction *rechercheManquant* de même complexité que celle-ci pour résoudre le problème.
3. Supposons qu'il manque maintenant deux entiers dans notre tableau. Écrire une fonction *rechercheDeuxManquants* renvoyant un tableau contenant ces deux entiers, ayant également la même complexité que la recherche dichotomique.

C/ Boucle.
On considère la méthode suivante :

```
public int mystere(int x, int n) {
   int r=1;
   while(n != 0) {
      if (n%2 ==0) {
         x=x*x;
         n=n/2;
      }
      else {
         r=r*x;
         n=n-1;
      }
   }
   return r;
}
```

1. Faire tourner le code sur les appels mystere$(2,3)$ et mystere$(8,2)$. Quelles sont les valeurs retournées lors de ces deux appels ?
2. Rappelez ce qu'est un *invariant de boucle.*
3. Soit x_0 et n_0 les valeurs de x et de n lors de l'appel à la méthode. Soit $\mathcal{P}_n$ la relation définie par :

$$r \times x^n = {x_0}^{n_0}.$$

   Montrer que $\mathcal{P}_n$ est un invariant de boucle.
4. Déduire de ce qui précède ce que fait la méthode mystère en examinant la valeur de sortie dictée par cet invariant.
5. Sur l'appel mystere$(x, 2^k)$, en fonction de k, combien de *multiplications* sont effectuées ? Qu'en concluez vous ?