Corrigé CC1 : Les Tris - Octobre 2014

1. On considère le tableau $\boxed{11}\boxed{3}\boxed{43}\boxed{10}$ de taille 4 et on déroule la boucle du schéma du tri de $i = 1$ à 3.

- $i = 1$: on recherche la place où insérer $A[1] = 3$ dans le sous-tableau $\boxed{11}$. Ce sous-tableau étant de taille 1, la recherche dichotomique retourne immédiatement la position 0 car $3 \leq 11$.
  On insère alors 3 en position 0 en décalant 11 et on obtient le tableau
  $\boxed{3}\boxed{11}\boxed{43}\boxed{10}$.
- $i = 2$: on recherche la place où insérer $A[2] = 43$ dans le sous-tableau $\boxed{3}\boxed{11}$. Par recherche dichotomique, on recherche cette position dans la deuxième partie $\boxed{11}$ du sous-tableau, ce qui retourne la place 2.
  Comme 43 est déjà en position 2, il n'y a aucune valeur à décaler et le tableau est inchangé :
  $\boxed{3}\boxed{11}\boxed{43}\boxed{10}$.
- $i = 3$: on recherche la place où insérer $A[3] = 10$ dans le sous-tableau $\boxed{3}\boxed{11}\boxed{43}$. Par recherche dichotomique, on recherche cette position dans la première partie $\boxed{3}\boxed{11}$ du sous-tableau, puis dans la seconde partie $\boxed{11}$ de ce dernier, ce qui retourne la place 1.
  On décale alors les valeurs 11 et 43, puis on place 10 à l'indice 1. On obtient le tableau trié
  $\boxed{3}\boxed{10}\boxed{11}\boxed{43}$.

2.

```
/**
 * recherche dichotomique de la place où insérer une valeur
 * dans un sous-tableau trié
 * @param A : le tableau
 * @param v : la valeur
 * @param deb, fin : les deux indices entre lesquels chercher
 * @return l'indice où insérer la valeur dans le sous-tableau
 */
public static int place(Double[] A, double v, int deb, int fin) {
    if (deb==fin) {
        if (A[deb] >= v) return deb;
        else return (deb+1);
        }
    }
    int milieu = (deb+fin)/2; //milieu est la partie entière de (deb+fin)/2
    if (v <= A[milieu]) return place(A, v, deb, milieu);
    else return place(A, v, (milieu+1), fin);
    }
}
```

3.

```
/** insère la valeur A[i] du tableau A parmi les i premiers éléments déjà triés
 * @param A : le tableau
 * @param i : indice
 */
public static void inserer(Double[] A,int i){
    Double val = A[i];//valeur à insérer
    int p =place(A,val,0,(i-1));//position d'insertion
    for (int k =i; k>p;k--) {
       A[k]=A[k-1]; //décalage des valeurs
    }
    A[p]=val; //placement de la valeur A[i]
}
```

4.

```
/** tri par insertion le tableau donné en argument
 * @param A : le tableau
 */
public static void tri(Double[] A) {
    for (int i = 1; i<A.length;i++){
       inserer(A,i);
    }
}
```

5. Vérifions tout d'abord que la méthode `place` retourne bien la position p où insérer la valeur $val = A[i]$ dans une portion de tableau A déjà triée, c'est-à-dire l'indice p vérifiant $val \leq A[j]$ si et seulement si $p \leq j$, pour j parcourant les indices entre deb et fin. Nous le démontrons pas récurrence sur la taille de la portion :

- Si la taille est 1, c'est-à-dire si $deb = fin$ alors la méthode place retourne deb si $val \leq A[deb]$ et $deb + 1$ sinon. La propriété est donc bien vérifiée.
- Si $deb < fin$, alors
  - ou bien $val \leq A[milieu]$ et on retourne alors récursivement la position p où insérer val dans la portion comprise entre deb et $milieu$. On a donc $\forall j \leq milieu\ (val \leq A[j] \iff p \leq j)$ et $p \leq milieu$. La portion comprise entre deb et fin étant supposée triée, comme val est plus petite ou égale à $A[mileu]$, elle est également plus petite ou égale aux valeurs se trouvant après le milieu du tableau, et donc $\forall j\ (val \leq A[j] \iff p \leq j)$.
  - ou bien $val > A[milieu]$, alors récursivement on retourne $p \geq mileu + 1$ tel que $\forall j \leq milieu\ (val \leq A[j] \iff p \leq i)$. Ici on a val qui est strictement supérieure aux valeurs se trouvant avant le milieu, ce qui permet également de conclure.

Vérifions maintenant que la méthode `insérer` insère $A[i]$ parmi les i premiers éléments déjà triés, de façon à ce que les $i + 1$ premiers éléments soient triés : en effet, cette méthode utilise la méthode place pour trouver la position p où insérer $A[i]$, puis elle décale vers la droite les valeurs se trouvant entre les indices p et $i - 1$, c'est-à-dire toutes les valeurs supérieures ou égales à $A[i]$, et enfin elle place $A[i]$ en p.

Pour terminer, montrons que la méthode `tri` trie le tableau A de taille n donné en entrée. Notons que le sous-tableau constitué du seul premier élément est évidemment trié. On vérifie alors qu'après chaque étape i de la boucle, la portion du tableau comprise entre les indices 0 et i est triée : par induction, au début de l'étape i de la boucle, les i premiers éléments sont triés. On insère alors $A[i]$ et les $i + 1$ premiers éléments sont ainsi triés. A la fin de la boucle la portion comprise entre les indices 0 et $n - 1$ est triée, c'est-à-dire le tableau en entier est trié.

6. Rappelons qu'on a vu en cours et TD que la complexité de la recherche dichotomique est en $O(\log n)$ (bon exercice à refaire). Pour insérer la valeur $A[i]$ parmi les i premiers éléments, on fait une recherche dichotomique sur un tableau de taille i, puis on décale au plus i valeurs. Comme $\log i = o(i)$, cette insertion a donc une complexité majorée par Ki où K est une constante. L'algorithme de tri consiste à insérer $A[i]$ pour i parcourant 0 à $n - 1$, donc la complexité de cette algorithme est majorée par $K(1 + 2 + \ldots + n - 1) = K(n - 1)n/2$. La complexité est en $O(n^2)$.

On peut de plus remarquer que dans le pire des cas (tableau dans l'ordre inverse), on doit décaler i valeurs à chaque insertion, et qu'en moyenne, on doit décaler $i/2$ valeurs à chaque insertion. On en déduit que dans le pire des cas, ainsi qu'en moyenne, le tri par insertion est en $\Theta(n^2)$.