Algorithmique
TP 1 : Tris et division pour régner

1 Implémentation

Dans le fichier `Tri.java`, implémentez d'abord la méthode `isSorted`, qui sera utilisée pour tester vos algorithmes. Ensuite, implémentez les quatre algorithmes de tri décrits en bas. Pour ce TP, on fait l'hypothèse que le tableau à trier (`T`) est de type `int[]`, c'est-à-dire un tableau d'entiers.

A. Tri à bulles ("*bubble sort*")
Le tri à bulles peut être décrit par le pseudocode ci-dessous :

```
fonction triBulles(T)
   n = taille de T
   pour i de n-1 à 1
       pour j de 0 à i-1
           si T[j] > T[j+1]
               échanger T[j+1] et T[j]
```

B. Tri par insertion ("*insertion sort*")
Le pseudocode de l'algorithme du tri par insertion est le suivant :

```
fonction triInsertion(T)
   n = taille de T
   pour j de n-2 à 0
      k = T[j]
      i = j+1
      tant que i < n et k > T[i]
         T[i-1] = T[i]
         incrémenter i
      T[i-1] = k
```

C. Tri par fusion ("*merge sort*")
Le tri par fusion utilise une stratégie diviser pour régner : on sépare le tableau en deux parties égales, on les trie et on fait ensuite une fusion des deux tableaux triés.

```
fonction triFusion(T, i, j)
   si i < j
      m = (i+j)/2
      triFusion(T, i, m)
      triFusion(T, m+1, j)
      fusion(T, i, m, j)
```

La fonction `fusion` est décrite par le pseudocode suivant :

```
fonction fusion(T, i, j, k)
   aux = tableau[0...k]
   pour r de i à k
      aux[r] = T[r]
   p = i, q = j+1
   pour r de i à k
      si p > j
         T[r] = aux[q], incrémenter q
      sinon si q > k
         T[r] = aux[p], incrémenter p
      sinon si aux[p] <= aux[q]
         T[r] = aux[p], incrémenter p
      sinon
         T[r] = aux[q], incrémenter q
```

2 Dichotomie

A. Dichotomie classique

Le principe de la recherche dichotomique est de chercher un élément dans les cases d'un tableau trié, en s'intéressant dans les cases entre i et j. Pour faire cela, on compare l'élément à celui de la case centrale `(i+j)/2`. Si notre élément est plus petit, on s'intéresse à la moitié de gauche, sinon à la moitié de droite (en excluant la case `(i+j)/2`). On s'arrête quand $i > j$ ou quand on a trouvé l'élément.
Implémentez la méthode `Dichotomie` qui prend en argument un tableau trié et un élément. Si l'élément se trouve dans le tableau, la méthode renvoie l'indice d'une case où se trouve l'élément. Si l'élément ne s'y trouve pas, la méthode renvoie `-1`.

B. Dichotomie en divisant par 3

On veut modifier l'algorithme de recherche dichotomique en coupant en trois au lieu de couper en deux à chaque étape :

- On compare l'élément à celui de la case `m1=(2i+j)/3`. Si notre élément est plus petit, on cherche entre i et $m1$.
- On compare l'élément à celui de la case `m2=(i+2j)/3`. Si notre élément est plus petit, on cherche entre $m1$ et $m2$.
- Sinon, on cherche entre $m2$ et j.

Si l'élément ne s'y trouve pas, la méthode renvoie `-1`.
Implémentez la méthode `Trichotomie` qui fait la recherche comme expliquée ci-dessus.

C. Nombre d'opérations

Implémentez les méthodes `CompteDichotomie` et `CompteTrichotomie` qui comptent combien de comparaisons sont faites pour faire la recherche. Ces fonctions prennent en paramètre un compteur, qu'elles vont incrémenter à chaque comparaison entre deux éléments. Quand l'élément est trouvé (ou quand on aurait dû renvoyer -1), on renvoie à la place la valeur du compteur. Ainsi, on saura combien de comparaison au total ont été faites.
Pour tester vos fonctions, décommentez la dernière partie du main. Le programme vous donnera la moyenne du nombre de comparaisons faites sur des tableaux tirés aléatoirement. Que constatez-vous ?