TD 7 & 8 – Le tri

Exercice 1 – tri bulle

Soit l'implantation suivante du tri bulle :

```
public void TriBulle (int[] tab) {
  int K, L;

  for (K = 0; K < tab.length - 1; K++) {
    for (L = tab.length - 1; L >= K + 1; L--) {
      if (tab[L] < tab[L-1])
        Echanger (L-1, L, tab);
    }
  }
}
```

a) Exécuter à la main le tri bulle sur la liste
3 7 6 4 10 5 8 2

b) En déduire une amélioration possible de l'algorithme
c) Modifier l'implantation ci-dessus pour intégrer cette amélioration

Exercice 2 – tri par énumération

Le tri par énumération est un tri par insertion. Chaque élément à insérer est comparé à tous les autres éléments de manière à déterminer le nombre d'éléments inférieurs à l'élément. On obtient de cette manière le rang de l'élément dans la liste définitive triée et il ne reste plus qu'à l'insérer à la place correspondante. En supposant la liste implantée sous forme de tableau :

a) écrire l'algorithme correspondant à cette description
```
void enumeration(int[] tab)
```

   indication : il sera plus facile de chercher dans un premier temps le rang de chacun des éléments du tableau, et ensuite seulement de déplacer les éléments pour les positionner à la bonne place

b) calculer sa complexité en termes de comparaisons et d'affectations. Est-ce un tri efficace ?

Exercice 3 – tri par tas et tri rapide

Soit la liste :
31 25 63 21 23 42 9 52 16

a) Exécuter à la main le tri par tas sur cette liste
b) Exécuter à la main le tri rapide sur cette même liste

Exercice 4 – application du tri rapide pour la recherche du k^{ème} plus petit élément

On se propose d'appliquer le principe du tri rapide pour rechercher le k^{ème} plus petit élément d'un tableau (c'est à dire celui qui se trouverait à la place d'indice k-1 si le tableau était trié).

Pour cela, on fait le constat suivant : lorsque, dans le tri rapide, on choisit un élément pivot et que l'on construit les sous-tableaux l_1 et l_2 des éléments qui sont respectivement plus petits et plus grands que l'élément pivot, ce dernier est déplacé jusqu'à sa place définitive. Par exemple, si l'élément choisi comme pivot est le 5ème plus petit élément du tableau, il sera déplacé à l'indice 4 (en numérotant les indices à partir de 0).

a) en déduire un algorithme pour trouver le k^{ème} plus petit élément d'un tableau
b) on donne la méthode de partition qui prend comme arguments un tableau d'entiers, et les indices `bi` et `bs` indiquant sur quelle partie du tableau appliquer la méthode. Celle-ci fait le choix d'un pivot selon la méthode vue en cours, trie la portion du tableau en deux sous-tableaux et renvoie la position finale du pivot.

   → écrire la méthode `int chercherKème(int[] tab, int bi, int bs, int k)` qui renvoie l'indice du k^{ème} plus petit élément du tableau `tab`.

```
int partition(int[] tab, int bi, int bs) {
  int pivot = (bi + bs) / 2;
  int min = bi ;

  // mise en ordre des éléments d'indice bi, pivot, bs
  if (tab[bi] > tab[bs]) min = bs ;
  if (tab[min] > tab[pivot]) min = pivot ;
  if (bi != min) Echanger(tab, bi, min) ;
  if (tab[bs] < tab[pivot]) Echanger(tab, bs, pivot) ;

  // déplacement des éléments mal placés
  while (bi < bs) {
    while ((bi < pivot) && (tab[bi] <= tab[pivot]) bi++ ;
    while ((bs > pivot) && (tab[bs] >= tab[pivot]) bs-- ;
    if (bi < bs) {
      Echanger(tab, bi, bs) ;
      if (bi == pivot) pivot = bs ;
      else if (bs == pivot) pivot = bi ;
    }
  }
  return pivot ;
}
```