TP3 : Parallélisme

Dans ce TP, nous programmerons deux algorithmes qui opèrent sur des tableaux (de nombres entiers). Une technique fréquente consiste à découper le tableau en deux parties disjointes et de faire exécuter le même traitement sur les deux parties en parallèle, par deux *exétrons*.

1 Maximum

Pour commencer, recherchons le maximum d'un tableau en cherchant en parallèle le maximum de sa première moitié et le maximum de sa seconde moitié.

1.1 Calcul en parallèle Pour exécuter un calcul en parallèle, il suffit de construire une instance de `Thread` et de lui envoyer le message `start`. On peut attendre que le calcul soit terminé en lui envoyant le message `join`.

Le calcul en question est décrit par un objet satisfaisant l'interface `Runnable`. Pour pouvoir récupérer le résultat d'un tel calcul, précisons-en l'interface.

Définissez une interface `Computation` qué étend l'interface `Runnable` en lui ajoutant une méthode `getResult` retournant un entier (le résultat du calcul, en supposant qu'il soit terminé).

1.2 Programmez un objet `maxInRange(int[] a, int i, int j)` qui satisfait l'interface `Computation` et qui, lorsqu'exécuté, recherche séquentiellement l'élément maximal dans le tableau `a` entre les indices `i` (inclus) et `j` (exclus).

1.3 Programmez une fonction qui recherche en parallèle l'élément maximal de chaque moitié d'un tableau puis renvoie l'élément maximal du tableau.

2 Tri fusion

Le tri fusion est un des algorithmes de tri efficaces qui se prête le mieux à une exécution en parallèle.

2.1 Programmez un algorithme de tri fusion en place[1] et séquentiel. Il s'appuiera sur une fonction `merge(int[] a, int i, int j, int k, int l)` qui opère la fusion (en place) des deux segments triés `a[i..j]` et `a[k..l]`.

Une telle fonction est donnée dans l'annexe A.

2.2 Programmez un objet (récursif) `sort(int[] a, int b, int e)` qui satisfait l'interface `Runnable` et qui, lorsqu'exécuté, trie en parallèle les deux moitiés du segment de tableau `a[b..e]` puis les fusionne.

En déduire une fonction `sort(int[] a)` qui trie en place le tableau `a`.

1. Ici, *en place* signifie que le tableau initial est modifié par l'opération de tri. Cela ne signifie pas que l'algorithme utilise un espace mémoire auxiliaire borné (indépendant de la taille du tableau à trier).

A Fusion

```
static void merge(
    final int[] a,
    final int leftB,
    final int leftE,
    final int rightB,
    final int rightE) {
  final int size = leftE - leftB + rightE - rightB + 2;
  final int[] tmp = new int[size];

  int i = 0;
  int l = leftB;
  int r = rightB;

  while ( l <= leftE && r <= rightE ) {
    final int v = a[l];
    final int w = a[r];
    int x;
    if (v <= w) {
      x = v;
      ++l;
    } else {
      x = w;
      ++r;
    }
    tmp[i++] = x;
  }

  while ( l <= leftE ) { tmp[i++] = a[l++]; }
  while ( r <= rightE ) { tmp[i++] = a[r++]; }

  final int leftSize = leftE - leftB + 1;
  final int rightSize = rightE - rightB + 1;
  System.arraycopy(tmp, 0, a, leftB, leftSize);
  System.arraycopy(tmp, leftSize, a, rightB, rightSize);
}
```