TP3

Objectifs

Manipulation des algorithmes de tri.

Exercice 1 : tri par sélection

Il consiste à trouver dans le tableau le numéro de l'élément le plus petit, c'est-à-dire l'entier min tel que ***tab[k]*** **>=** ***tab[min]*** pour tout ***k***. Une fois ce numéro trouvé, les éléments ***tab [1]*** et ***tab[min]*** sont échangés – cet échange nécessite une variable temporaire de type entier – puis la même procédure est appliquée sur la suite d'éléments ***tab[2], ..., tab[N]***.

```
procedure triSelection(entier[] tab)
  entier i, k;
  entier min;
  entier tmp;
  pour (i de 1 à N-1 en incrémentant de 1) faire
    /* recherche du numero du minimum */
    min <- i;
    pour (k de i+1 à N en incrémentant de 1) faire
      si (tab[k] < tab[min]) alors
        min <- k;
      fin si
    fin pour
    /* echange des valeurs entre la case courante et le
minimum */
    tmp <- tab[i];
    tab[i] <- tab[min];
    tab[min] <- tmp;
  fin pour
fin procedure
```

Exercice 2 : tri bulle

Le « tri bulle » est une variante du tri par sélection. Il consiste à parcourir le tableau ***tab*** en permutant toute paire d'éléments consécutifs (***tab[k], tab[k+1]***) non ordonnés – ce qui est un échange et nécessite donc encore une variable intermédiaire de type entier. Après le premier parcours, le plus grand élément se retrouve dans la dernière case du tableau, en ***tab[N]***, et il reste donc à appliquer la même procédure sur le tableau composé des éléments ***tab[1], ..., tab[N-1]***. Le nom de ce tri provient du déplacement des « bulles » les plus grandes vers la droite.

```
procedure triBulle(entier[] tab)
  entier i, k;
  entier tmp;
  pour (i de N à 2 en décrémentant de 1) faire
    pour (k de 1 à i-1 en incrémentant de 1) faire
      si (tab[k] > tab[k+1]) alors
```

```
        tmp <- tab[k];
        tab[k] <- tab[k+1];
        tab[k+1] <- tmp;
      fin si
    fin pour
  fin pour
fin procedure
```

Exercice 3 : tri par insertion

Cette méthode de tri est très différente de la méthode de tri par sélection et s'apparente à celle utilisée pour trier ses cartes dans un jeu : on prend une carte, ***tab[1]***, puis la deuxième, ***tab[2],*** que l'on place en fonction de la première, ensuite la troisième tab[3] que l'on insère à sa place en fonction des deux premières et ainsi de suite. Le principe général est donc de considérer que les ***(i-1)*** premières cartes, ***tab[1],..., tab[i-1]*** sont triées et de placer la ie carte, ***tab[i]***, à sa place parmi les ***(i-1)*** déjà triées, et ce jusqu'à ce que ***i = N***.
Pour placer ***tab[i]***, on utilise une variable intermédiaire ***tmp*** pour conserver sa valeur qu'on compare successivement à chaque élément ***tab[i-1], tab[i-2]***, ... qu'on déplace vers la droite tant que sa valeur est supérieure à celle de ***tmp***. On affecte alors à l'emplacement dans le tableau laissé libre par ce décalage la valeur de ***tmp***.

```
procedure triInsertion(entier[] tab)
  entier i, k;
  entier tmp;
  entier k;
  pour (i de 2 à N en incrémentant de 1) faire
    tmp <- tab[i];
    k <- i;
    tant que (k > 1 et tab[k - 1] > tmp) faire
      tab[k] <- tab[k - 1];
      k <- k - 1;
    fin tant que
  tab[k] <- tmp;
  fin pour
fin procedure
```

Exercice 4 : tri rapide ou Quicksort

Cette méthode de tri, probablement la plus utilisée actuellement. Elle illustre le principe dit « diviser pour régner », qui consiste à appliquer récursivement une méthode destinée à un problème de taille donnée à des sous-problèmes similaires, mais de taille inférieure. Ce principe général produit des algorithmes qui permettent souvent d'importantes réductions de complexité.

On considère un élément au hasard dans le tableau, le pivot, dont on affecte la valeur à une variable, disons pivot. On procède alors à une partition du tableau en 2 zones : les éléments inférieurs ou égaux à pivot et les éléments supérieurs ou égaux à pivot. Si on parvient à mettre

les éléments plus petits en tête du tableau et les éléments plus grands en queue de tableau, alors on peut placer la valeur de pivot à sa place définitive, entre les deux zones. On répète ensuite récursivement la procédure sur chacune des partitions créées jusqu'à ce qu'elle soit réduite à un ensemble à un seul élément.

```
fonction partition(entier[] tab, entier debut, entier fin)
retourne entier
  entier indicePivot, i, k, tmp
  indicePivot <- entier aléatoire entre debut et fin
  tmp <- tab[indicePivot]
  tab[indicePivot] <- tab[debut]
  tab[debut] <- tmp
  k <- debut

  pour i de debut + 1 à fin faire
    si tab[i] < tab[debut] alors
      k <- k + 1
      tmp <- tab[i]
      tab[i] <- tab[k]
      tab[k] <- tmp
    fin si
  fin pour

  tmp <- tab[debut]
  tab[debut] <- tab[k]
  tab[k] <- tmp

  retourner k
fin fonction

procedure triRapideR(entier[] tab, entier debut, entier fin)
  si fin > debut alors
    entier pivot <- partition(tab, debut, fin)
    triRapideR(tab, debut, pivot - 1)
    triRapideR(tab, pivot + 1, fin)
  fin si
fin procedure

procedure triRapide(entier[] tab)
  triRapideR(tab, 1, N)
fin procedure
```

Exercice 5 : tri par fusion

Ce tri est un autre exemple de méthode qui applique le principe « diviser pour régner ». En effet, étant données deux suites d'éléments triés, de longueurs respectives ***L1*** et ***L2***, il est très facile d'obtenir une troisième suite d'éléments triés de longueur ***L1 + L2***, par « interclassement » (ou fusion) des deux précédentes suites, comme illustré dans la procédure fusion.
Pour les besoins de la procédure ***triFusion***, nous allons donner la forme suivante à la procédure fusion qui interclasse deux suites d'éléments placés dans un tableau ***tab***, respectivement entre les indices ***debut*** et ***mil*** et entre les indices ***mil + 1*** et ***fin*** :

```
procedure fusion(entier[] tab, entier[] tmp, entier debut,
entier mil, entier fin)
  entier k;
  entier i <- debut;
  entier j <- mil + 1;
  pour (k de debut à fin en incrémentant de 1) faire
    si ((j > fin) ou (i <= mil et tab[i] < tab[j])) alors
      tmp[k] <- tab[i];
      i <- i + 1;
    sinon
      tmp[k] <- tab[j];
      j <- j + 1;
    fin si
  fin pour
  pour (k de debut à fin en incrémentant de 1) faire
  tab[k] <- tmp[k];
  fin pour
fin procedure
```

La procédure récursive de tri fusion est alors :

```
procedure triFusionR(entier[] tab, entier[] tmp, entier debut,
entier fin)
  si (debut < fin) alors
    entier milieu <- (debut + fin)/2;
    triFusionR(tab, tmp, debut, milieu);
    triFusionR(tab, tmp, milieu + 1, fin);
    fusion(tab, tmp, debut, milieu, fin);
  fin si
fin procedure

procedure triFusion(entier[] tab)
  entier[] tmp <- tableau de taille N;
  triFusionR(tab, tmp, 1, N);
fin procedure
```