TD4 : Algorithmes de tri

Trier des données est une tâche récurrente en informatique. Il existe de nombreux algorithmes de tri plus ou moins faciles à mettre en uvre et plus ou moins efficaces.
Voici trois algorithmes de tri : tri par insertion (algorithme 1), tri base (algorithme 2) et tri fusion (algorithme 4). Pour chacun d'entre eux, prouver sa correction totale.

Algorithme 1 Tri par insertion

entrée : un tableau d'entiers T et sa longueur n
sortie : T est trié

```
1 : pour i allant de 0 à n−1 faire
2 :     e ← T[i]
3 :     j ← i−1
4 :     tant que j ≥ 0 et e < T[j] faire
5 :         T[j+1] ← T[j]
6 :         j ← j−1
7 :     fin tant que
8 :     T[j+1] ← e
9 : fin pour
```

Algorithme 2 Tri base (ou tri radix)

entrée : un tableau T d'entiers représentés en base 10 et sa longueur n
sortie : T est trié

1 : Du chiffre le moins significatif au chiffre le plus significatif, on applique le tri selon le chiffre à T (algorithme 3)

Algorithme 3 Tri selon le chiffre

entrée : un tableau T d'entiers représentés en base 10, sa longueur n et un indice i de chiffre
sortie : T est trié selon le chiffre d'indice i ; si deux éléments ont le même chiffre d'indice i, ils restent dans le même ordre

1 : on crée dix suites vides $B_0, \dots, B_9$
2 : **pour tout** élément e de T (dans l'ordre du parcours) **faire**
3 : on place l'élément à la fin de la suite B_{e_i}, où e_i est le i^{e} chiffre de e.
4 : **fin pour**
5 : on renvoie la concaténation de $B_0, \dots, B_9$

Algorithme 4 Tri fusion

```
/**
 * entree : un tableau tab ou il faut fusionner les valeurs entre debut et milieu-1
 * et les valeurs en milieu et fin-1, les valeurs sont supposees dans l'ordre
     croissant
 * dans ces deux parties
 * sortie : la fusion voulue est faite
 */
void fusion(int *tab, int debut, int milieu, int fin){
  int g = 0, d = 0, t = debut;
  int *gauche = (int *)malloc((milieu-debut)*sizeof(int));
  int *droite = (int *)malloc((fin-milieu)*sizeof(int));

  for(int i=0; i<milieu-debut; i=i+1){
    gauche[i] = tab[debut+i];
  }
  for(int i=0; i<fin-milieu; i=i+1){
    droite[i] = tab[milieu+i];
  }

  while(g<milieu-debut && d<fin-milieu){
    if(gauche[g]<droite[d]){
      tab[t] = gauche[g];
      g = g+1;
    } else {
      tab[t] = droite[d];
      d = d+1;
    }
    t = t+1;
  }
  while(g<milieu-debut){
      tab[t] = gauche[g];
      g = g+1;
      t = t+1;
  }
  while(d<fin-milieu){
      tab[t] = droite[d];
      d = d+1;
      t = t+1;
  }

  free(gauche);
  free(droite);
}

/**
 * entree : un tableau d'entiers, a trier entre les indices debut et fin-1 compris
 * sortie : le tableau est trie entre les indices debut et fin-1
 */
void tri_fusion(int *tab, int debut, int fin){
  if (debut==fin-1) return;
  int lg = (fin-debut)/2;
  tri_fusion(tab, debut, debut+lg);
  tri_fusion(tab, debut+lg, fin);
  fusion(tab, debut, debut+lg,fin);
}
```