

## Feuille de travaux dirigés n° 3

### Structures de données

#### Exercice 3.1 — Tri par Insertion

On considère  $T$  un tableau **trié** de  $n$  entiers.

1. Quelle est la complexité du tri par insertion dans le pire des cas ? Et dans le meilleur ?

##### ▽ Correction

Cf. cours : meilleur des cas (tableau déjà trié)  $O(n)$ , pire des cas (tableau trié en ordre inverse)  $O(n^2)$ .

2. On modifie une valeur du tableau (par exemple  $T[5]$ ). Écrivez un algorithme qui retre le tableau. Votre algorithme doit être simple et efficace. Donnez sa complexité.

##### ▽ Correction

```
// i : indice de l'élément modifié
retrie(entier T[1...n], entier i) {
    entier e ← T[i]
    si (e > T[i+1]) {
        tant que (i < n et e > T[i + 1]) {
            T[i] ← T[i + 1]
            i ← i + 1
        }
        T[i] ← e
    } sinon si (e < T[i-1]) {
        tant que (i > 1 et e < T[i - 1]) {
            T[i] ← T[i - 1]
            i ← i - 1
        }
        T[i] ← e
    }
}
```

Complexité dans le pire des cas en  $O(n)$ , dans le meilleur des cas en  $O(1)$

3. On modifie une valeur du tableau. Appelez directement l'algorithme de tri par insertion. Quelle est la complexité obtenue ?

##### ▽ Correction

Complexité dans le pire des cas en  $O(n)$ , dans le meilleur des cas en  $O(n)$

#### Exercice 3.2 — Tri par Sélection

Sur un tableau de  $n$  éléments (numérotés de 1 à  $n$ ), le principe du tri par sélection est le suivant :

- rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 1 ;
  - rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 2 ;
  - continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.
1. Écrivez le pseudo-code correspondant à cet algorithme. Donnez sa complexité dans le pire des cas et dans le meilleur.

#### ▽ Correction

Cf. CM2

```
tri_selection(T[], n) {
  pour (i de 1 #à# n-1) {
    min ← i
    pour (j de i + 1 #à# n) {
      si (T[j] < T[min]) {
        min ← j
      }
    }
    si (min ≠ i) {
      échanger(T[i], T[min])
    }
  }
}
```

Complexité dans tous les cas en  $O(n^2)$

2. Montrez que l'invariant de boucle suivant permet de prouver la correction de l'algorithme : à la fin de l'étape  $i$ , le tableau est une permutation du tableau initial et les  $i$  premiers éléments du tableau coïncident avec les  $i$  premiers éléments du tableau trié.

#### ▽ Correction

Si on suppose par l'absurde que l'algorithme n'est pas correct, alors il existe au moins une valeur qui n'est pas à sa place dans le tableau. Contradiction avec l'invariant de boucle.

### Exercice 3.3 — Tri par Fusion

Soient  $T_1$  un tableau de  $n_1$  éléments triés par ordre croissant et  $T_2$  un tableau de  $n_2$  éléments triés par ordre croissant.

1. Écrivez un algorithme qui construit le tableau  $T$  constitué des éléments de  $T_1$  et de  $T_2$ , triés par ordre croissant.

#### ▽ Correction

Partie fusion de l'algorithme de tri fusion

```
fusion(T1[], n1, T2[], n2, T[]) {
  i ← 1 // indice pour le tableau résultat
  i1 ← 1 // indice pour le premier tableau
  i2 ← 1 // indice pour le second tableau
  tant que (i1 ≤ n1 et i2 ≤ n2) {
    si (T1[i1] < T2[i2]) {
      T[i] ← T1[i1]
      i1 ← i1 + 1
    } sinon {
      T[i] ← T2[i2]
      i2 ← i2 + 1
    }
  }
}
```

```

    i <- i + 1
  }

  // le tableau T2 est vide on ajoute tous les éléments restants dans T1
  tant que (i1 ≤ n1) {
    T[i] <- T1[i1]
    i1 <- i1 + 1
    i <- i + 1
  }

  // le tableau T1 est vide on ajoute tous les éléments restants dans T2
  tant que (i2 ≤ n2) {
    T[i] <- T2[i2]
    i2 <- i2 + 1
    i <- i + 1
  }
}

```

2. Donnez la complexité de cet algorithme.

▽ **Correction**

Complexité dans tous les cas en  $O(n_1 + n_2)$

3. Considérez que vous avez un tableau de taille 16. Comment pouvez-vous appliquer le principe précédent pour obtenir un algorithme efficace de tri ? Écrivez le pseudo-code de cet algorithme. Généralisez cet algorithme à toute taille de tableau. Donnez sa complexité.

▽ **Correction**

Tri fusion

```

tri_fusion(T[], n) {
  m <- n / 2
  T1[] <- tri_fusion(T[1...m], m)
  T2[] <- tri_fusion(T[m+1...n], n - m)
  fusion(T1, m, T2, n - m, T)
}

```

Complexité en  $O(n \log(n))$

### Exercice 3.4 — Tri de Matrice

Proposez une méthode pour trier les éléments d'une matrice.

▽ **Correction**

Linéarisation de la matrice puis votre tri préféré.

### Exercice 3.5 — Structure de Tas

#### Rappel de cours :

Un tas descendant est un arbre binaire vérifiant les propriétés suivantes :

- la différence maximale de profondeur entre deux feuilles est de 1 (*i.e.* toutes les feuilles se trouvent sur la dernière ou sur l'avant-dernière ligne);

- les feuilles de profondeur maximale sont “tassées” sur la gauche.
- chaque nœud est de valeur inférieure à celle de ces deux fils.

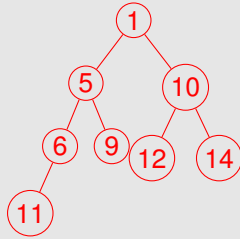
Un tas ou un arbre binaire presque complet peut être stocké dans un tableau, en posant que les deux descendants de l’élément d’indice  $n$  sont les éléments d’indices  $2n$  et  $2n + 1$  (pour un tableau indicé à partir de 1). En d’autres termes, les nœuds de l’arbre sont placés dans le tableau ligne par ligne, chaque ligne étant décrite de gauche à droite.

L’insertion d’un élément dans un tas se fait de la façon suivante : on place l’élément sur la première case libre et on échange l’élément et son père quand ce dernier est supérieur et qu’il existe.

L’opération de **tamissage** consiste à échanger la racine avec le plus petit de ses fils, et ainsi de suite récursivement jusqu’à ce qu’elle soit à sa place.

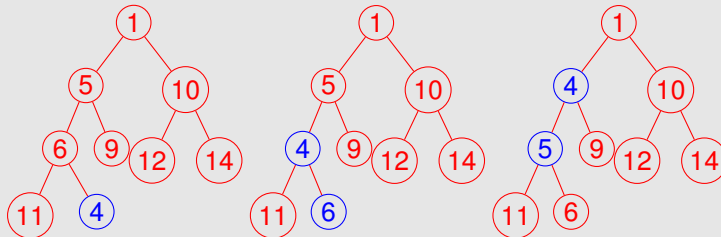
1. Dessinez un tas correspondant au tableau  $T$  : [1, 5, 10, 6, 9, 12, 14, 11]

▽ **Correction**



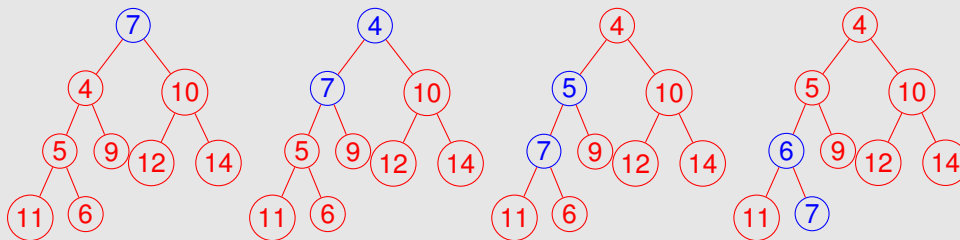
2. Insérez la valeur 4 dans le tas.

▽ **Correction**



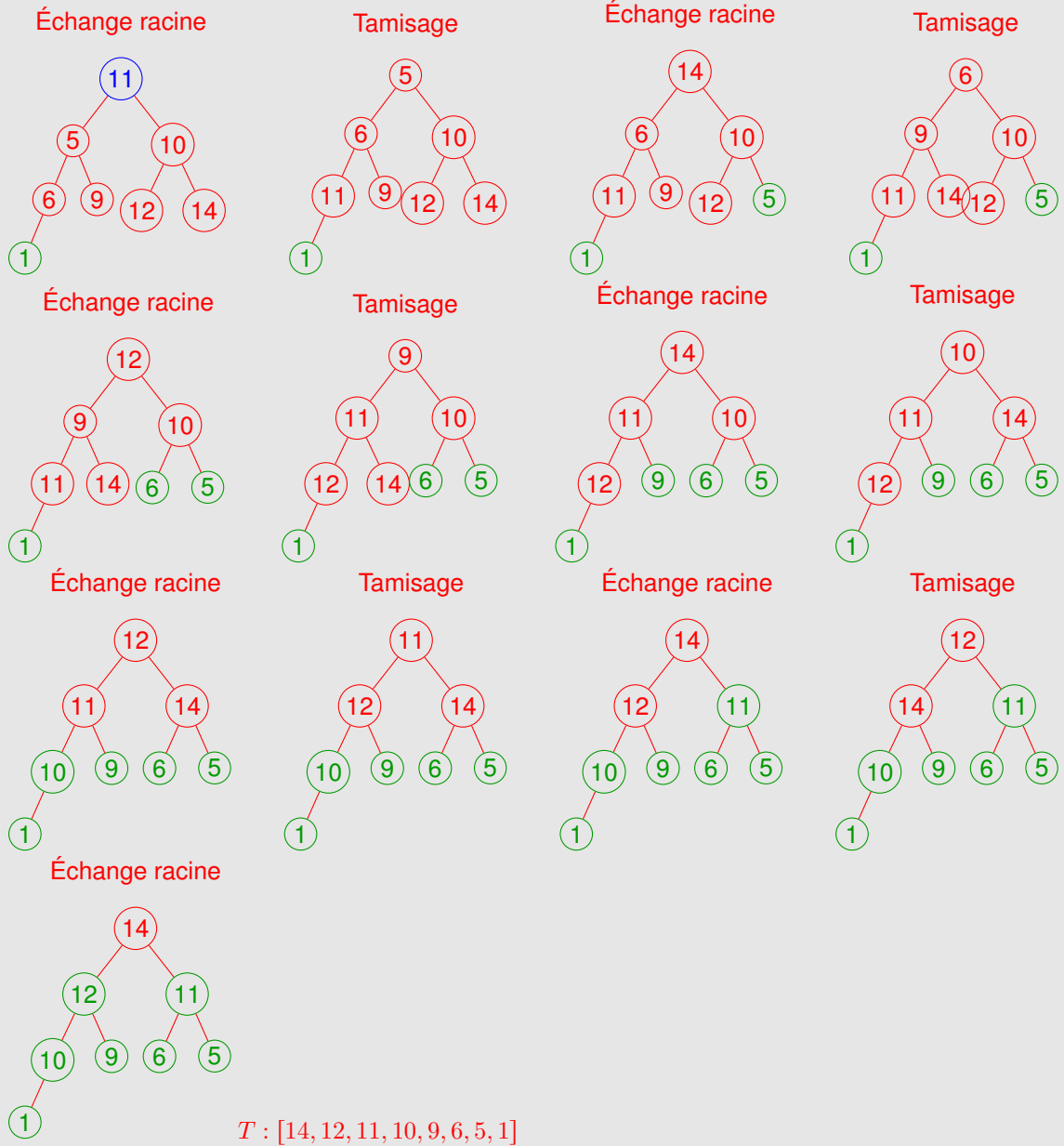
3. Remplacez la racine du tas par la valeur 7 et détaillez le tamissage.

▽ **Correction**



4. Appliquez l’algorithme suivant pour trier le tableau  $T$  (on fera des dessins successifs montrant l’évolution du tas) :
  - on commence par transformer le tableau en tas descendant ;
  - on échange la racine avec le dernier élément du tableau, et on restreint le tas en ne touchant plus au dernier élément, c’est-à-dire à l’ancienne racine ;
  - on tamise la racine dans le nouveau tas, et on répète l’opération sur le tas restreint jusqu’à l’avoir vidé et remplacé par un tableau trié.

▽ **Correction**



5. Est-ce que cet algorithme trie le tableau  $T$  de façon croissante ou décroissante ?

▽ **Correction**

De façon décroissante

6. Que faudrait-il changer pour avoir le tableau trié dans l'ordre inverse ?

▽ **Correction**

Utiliser un tas ascendant (chaque nœud est de valeur supérieure à celle de ces deux fils).