.1 Tris basiques

1. Pour trier un jeu de cartes une méthode possible est de lancer les cartes en l'air jusqu'à ce qu'elles retombent triées. Cette méthode s'appelle le *Bogo-Tri*. Ecrivez une fonction simulant le *Bogo-tri* sur une liste de données et évaluez sa complexité. Pouvez-vous imaginer un cas dans lequel cette méthode s'avère effective ?
2. **Récursion :** écrivez un algorithme récursif qui calcule le n^{ieme} terme de la suite de Fibonacci. Cette suite est définie de la manière suivante :
   - $F_0 = 1$
   - $F_1 = 1$
   - $F_{n+2} = F_{n+1} + F_n, \forall n \in \mathbb{N}$
3. Écrivez une version récursive de l'algorithme demandé à l'exercice 9 de la feuille de TD précédente. Au lieu de supprimer les lignes redondantes, on cherchera ici à lister les paires de doublons.
4. Écrire une fonction booléenne qui, étant donné un tableau `tab` répond vrai si le tableau est (déjà) trié.
5. Écrire une fonction booléenne de comparaison de tableaux de caractères : étant donnés deux tableaux de caractères terminés par `'\0'` (méthode de codage du C), soient u et v, renvoyer -1 si $u < v$, 0 si $u = v$, et 1 si $u > v$ (ordre lexicographique). Donner deux versions de l'algorithme : itérative et récursive.
6. Proposer une version sans sentinelle de l'algorithme de tri par insertion, et déterminer le coût supplémentaire.
7. Proposer une version itérative de l'algorithme de tri par insertion.
8. **Champion borné** Écrire un programme qui recherche dans un tableau la plus petite valeur comprises entre deux bornes `a` et `b` données par l'utilisateur.
9. Écrire deux versions, l'une récursive, l'autre itérative, d'une fonction booléenne nommée `appartient` qui reçoit comme argument un tableau d'entiers et des indices de début et de fin, ainsi qu'une valeur (entière), et répond vrai si la valeur figure dans le tableau. Pour la version récursive, on pourra raisonner de la manière suivante : *un élément appartient au tableau soit s'il se trouve dans la première case, soit s'il appartient au tableau formé en enlevant la première case.*
10. On suppose que l'on dispose d'un tableau de chaînes de caractères (de longueur maximale 20 caractères : `string[20]`), de dimension N, correspondant à un ensemble de mots.

    ```
    var Tm = array[1..N] of string[20]
    ```

    Ce tableau n'est pas trié. On veut produire un tableau trié sur l'envers des mots (on considère chaque mot « à l'envers », de la dernière lettre à la première)[1].

    (a) Écrire une fonction `avant` qui prend en argument deux chaînes de caractère, et répond vrai si la première est avant la seconde, dans le sens défini précédemment (c'est-à-dire en prenant les mots à l'envers).

    ```
    function avant(x, y : string[20]) : boolean;
    ```

    (b) Proposer un algorithme qui réalise le tri du tableau initial, en utilisant la fonction `avant`.

    (c) Discuter et justifier l'algorithme que vous avez choisi.

[1] On utilise fréquemment ce genre de technique pour travailler sur les suffixes.