❑ **Exercice 1** : *Nombre d'arêtes*

1. Rappeler la définition d'un arbre binaire.
2. Soit a un arbre binaire à n noeuds ($n \geqslant 1$), montrer que a possède $n-1$ arêtes.
3. On rappelle l'implémentation des arbres en OCaml utilisée en cours :

```
type ab =
 |Vide
 |Noeud of ab *  int * ab;;
```

   En utilisant cette implémentation, écrire une fonction `nb_aretes` de signature `ab -> int` et qui renvoie le nombre d'arêtes d'un arbre binaire

❑ **Exercice 2** : *Recherche dans un* ABR

1. Rappeler la définition d'un arbre binaire de recherche
2. On suppose maintenant qu'on a inséré dans un ABR initialement vide tous les entiers compris en 0 et 999. On effectue la recherche de l'entier 666 dans cet arbre. Parmi les séquences de valeurs suivantes, lesquelles peuvent être la séquence de noeuds parcourus jusqu'à atteindre 666 ? :
   — 487, 503, 911, 954, 499, 651, 672, 668, 666
   — 951, 812, 803, 798, 751, 670, 589, 652, 653, 666
   — 985, 112, 251, 306, 444, 503, 574, 602, 605, 681, 666
   — 844, 511, 845, 603, 702, 651, 699, 660, 670, 665, 666
   — 303, 404, 541, 752, 749, 742, 592, 603, 666
3. Proposer un algorithme qui prend en entrée une séquence d'entiers $u_0, \dots u_n$ avec u_n la valeur cherchée et vérifie que cette séquence peut effectivement constituer la suite de noeuds visités lors de la recherche réussie d'un nombre dans un tel ABR. L'algorithme doit avoir une complexité temporelle en $O(n)$.
4. En fournir une implémentation en OCaml, en supposant que la séquence est donnée sous la forme d'un tableau d'entiers de OCaml. La signature de votre fonction sera donc `int array -> bool`
5. Soit t un tableau représentant la suite de valeurs obtenue lors de la recherche réussie d'un élément dans un ABR, proposer un algorithme *de complexité linéaire* permettant de trier ce tableau. En donner l'implémentation en OCaml.

❑ **Exercice 3** : *Collision dans une table de hachage*

Pour une chaine de caractères $s = c_0 \dots c_{n-1}$, on considère la fonction de hachage :

$$h(s) = \sum_{i=0}^{n-1} 31^i \times c_i$$

1. Calculer le hash de la chaine `"AB"`.
2. Montrer qu'il existe deux chaines de caractères de longueur 2, formées de lettres minuscules (code 97 à 177) ou majuscules (code 65 à 90) et produisant la même valeur pour h.
3. En déduire une façon de construire un nombre arbitraire de chaînes de caractères de longueurs quelconques ayant la même valeur pour la fonction h.
4. Pour implémenter cette fonction en langage C, on propose une fonction de signature `int hash(char *s)`. Qu'en pensez-vous ?
5. Proposer une implémentation *efficace* pour cette fonction en langage C.
6. Déterminer, grâce à la question 3, deux chaines de 8 caractères produisant une collision et le vérifier.