❑ **Exercice 1** : *Représentation d'arbres binaires*

1. Dessiner tous les arbres binaires ayant 3 noeuds.
2. Dessiner tous les arbres binaires ayant 4 noeuds.
3. Dessiner un arbre binaire ayant 8 noeuds et de hauteur maximale (resp. minimale).

❑ **Exercice 2** : *Représentation en C*

On rappelle qu'on a défini en C, un arbre binaire (avec des étiquettes entières) par :

```c
struct noeud
{
    struct noeud *sag;
    int valeur;
    struct noeud *sad;
};
typedef struct noeud noeud;
typedef noeud *ab;
```

1. Rappeler la définition de la hauteur d'un arbre binaire et écrire une fonction de prototype `int hauteur(ab arbrebinaire)` qui renvoie la hauteur de l'arbre donné en argument.
2. On rappelle que dans cette implémentation, l'espace nécessaire au stockage des noeuds est alloué dynamiquement à l'aide d'instructions `malloc`. Ecrire une fonction de prototype `void libere(ab* arbrebinaire)` qui détruit l'arbre binaire donné en paramètre, en libérant l'espace alloué par ses noeuds. A la fin de l'appel `arbrebinaire` est le pointeur NULL.

❑ **Exercice 3** : *Représentation en OCaml*

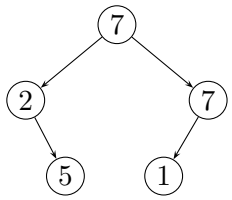
On rappelle qu'on a défini en OCaml un arbre binaire (avec des étiquettes entières) par :

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

1. Dessiner l'arbre représenté par :

```
let t = Noeud(
    Noeud(
      Noeud(Noeud(Vide,2,Noeud(Vide,3,Vide)),8,Vide),
      9,
      Noeud(Vide,12,Vide)),
    11,
    Noeud(Noeud(Vide,13,Vide),
          15,
          Vide))
```

2. Donner sa taille et sa hauteur
3. S'agit-il d'un arbre binaire de recherche ? Justifier
4. Donner la représentation en OCaml de l'arbre :



❑ **Exercice 4** : *Un peu de dénombrement*

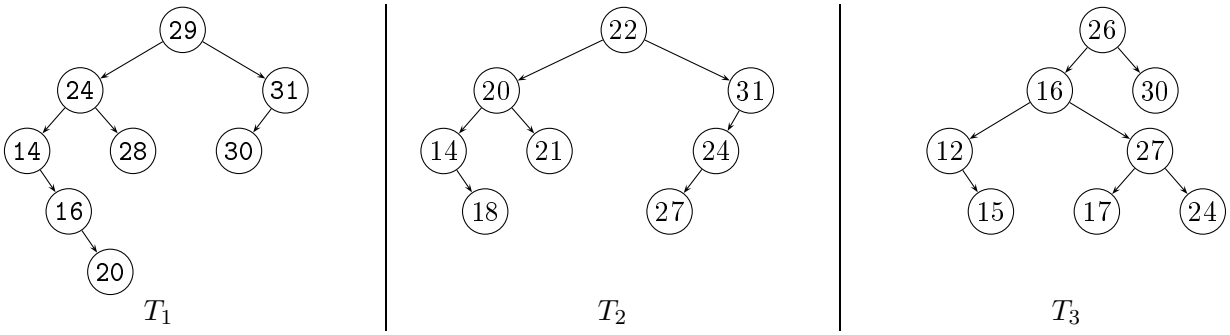
On note T_n le nombre d'arbres binaires à n noeuds.

1. Donner T_0 et déterminer une relation de récurrence liant les $(T_k)_{0\leqslant k\leqslant n}$
   ✪ Revenir à la définition vue en cours et considérer un arbre binaire à n noeud comme un arbre binaire ayant un sous arbre contenant k noeuds et un sous arbre contenant $n-k-1$ noeud.
2. Vérifier que $T_5 = 42$.
   Remarque :
   Le nombre de Catalan d'indice n est défini par :

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

   et on prouve que $T_n = C_n$.

❑ **Exercice 5** : *Parcours d'un arbre binaire*



1. Pour chacun des trois arbres binaires ci-dessus, donner l'ordre des noeuds lors d'un parcours prefixe, infixe et suffixe.
2. L'un de ces arbres binaires est-il un ABR ? Justifier

❑ **Exercice 6** : *Un peu de complexité*
On considère la fonction OCaml suivante qui prend en argument un arbre binaire tel que défini par le type de l'exercice 3

```
let rec mystere ab =
  match ab with
  | Vide -> []
  | Noeud(g,v,d) ->   v::mystere g @ mystere d;;
```

1. Ecrire une spécification et donner un nom plus approprié à la fonction `mystere`.
2. Rappeler la complexité de l'opérateur `@` et en déduire celle de la fonction `mystere`
3. Proposer une version de cette fonction ayant une complexité linéaire en fonction du nombre de noeuds de l'arbre.
   ✪ Utiliser une fonction auxiliaire avec un accumulateur en *et écrire la spécification de cette fonction.*

❑ **Exercice 7** : *Reconstruction d'un arbre à partir de parcours*

1. Est-il possible de reconstruire de façon unique un arbre binaire à partir de son parcours préfixe et de son parcours postfixe ?
2. Quel est l'arbre binaire dont le parcours préfixe est `16; 6; 2; 8; 9; 12; 24; 19; 26` et le parcours infixe `2; 6; 8; 9; 12; 16; 19; 24; 26`

❑ **Exercice 8** : *Arbre binaire de recherche*

1. Rappeler la caractérisation d'un arbre binaire de recherche par le parcours infixe
2. En utilisant le parcours infixe, écrire une fonction `est_abr` de signature `ab -> bool` qui indique si l'arbre donné en paramètre est un ABR ou non. Donner sa complexité, on supposera qu'on dispose déjà de la fonction qui renvoie le parcours infixe de l'arbre et que cette fonction a une complexité en $O(n)$.
3. On propose la solution suivante pour déterminer si un arbre est un ABR :

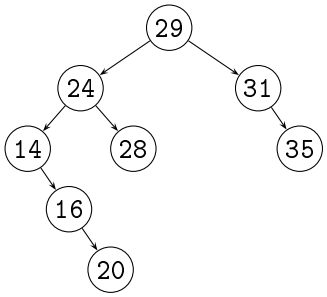
```
let rec est_abr tabr =
  match tabr with
  | Vide -> true
  | Noeud (g, v, d) -> est_abr g && est_abr d && plus_petit g v && plus_grand d v
;;
```

Selon vous, quel est le rôle des fonctions `plus_petit` et `plus_grand` ? Ecrire ces fonctions.

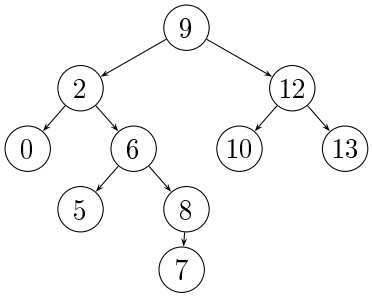
4. La solution précédente est-elle correcte ? Quelle est sa complexité ?
5. Pour tester si un arbre est un ABR, on propose de parcourir l'arbre en donnant l'intervalle de valeurs dans lequel doit se trouver les éléments. Initialement l'intervalle est celui des entiers représentables (qu'on peut obtenir avec `Int.min_int` et `Int.max_int`), puis à chaque fois qu'on descend à gauche (resp. à droite) on met à jour la borne droite (resp gauche) de l'intervalle. Ecrire cette nouvelle méthode pour tester si un arbre est un ABR.
6. Quelle est la complexité de cette fonction ?

❑ Exercice 9 : *Opérations sur un* ABR

1. Rappeler l'algorithme d'insertion d'un élément dans un arbre binaire de recherche
2. Détailler l'insertion de la valeur 17 dans l'arbre ci-dessous (en ayant vérifié qu'il s'agit bien d'un ABR)



3. Ecrire une fonction `extraire_min` en OCaml de signaure `abr -> int * abr`, qui renvoie un couple composé du minimum d'un arbre binaire de recherche et de cet arbre privé du noeud de valeur minimale. On gère le cas de l'arbre vide avec `failwith`
4. Afin de supprimer une valeur dans un ABR, on propose de procéder de la façon suivante : si l'arbre est vide la suppression est impossible, sinon on descend dans le sous arbre gauche (resp. droit) si la racine est strictement plus grande (resp. petite) que la valeur a supprimé. Si la racine est égale à la valeur à supprimer alors on la remplace par le minimum du sous arbre droit que l'on supprime. Détailler le fonctionnement de cette méthode sur l'arbre suivant d'où on veut supprimer la valeur 2 :



5. Donner une implémentation en OCaml sous la forme d'une fonction `supprime` de signature `abr -> int -> abr`

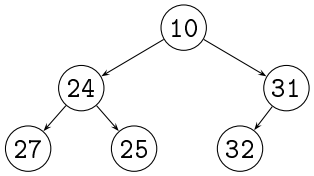
❑ Exercice 10 : *Recherche dans un* ABR

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

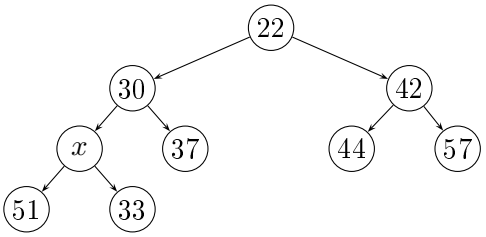
— 303, 404, 541, 752, 749, 742, 592, 603, 666

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'une liste d'entiers de OCaml. La signature de votre fonction sera donc `int list -> bool`
5. Soit t une liste 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 11 : *Sur les tas*



T_1



T_2

1. Vérifier que T_1 est un tas binaire min.
2. Pour quelles valeurs de x, T_2 est-il un tas ?
3. Donner la représentation de T_2 par un tableau.
4. Dessiner un arbre binaire contenant les valeurs $6, 18, 10, 9$ et 17 telle que cet arbre ne soit *pas* un tas et que l'étiquette de chaque noeud soit plus petite que les étiquettes de ses fils (lorsqu'elles existent).
5. Rappeler l'algorithme d'extraction du minimum d'un tas et l'appliquer à T_1 et donner l'arbre obtenu après extraction.
6. Rappeler l'algorithme d'insertion dans un tas et l'appliquer à T_2 pour insérer la valeur 28 et donner l'arbre obtenu après cette insertion.

❑ Exercice 12 : *Tri par tas*

1. Rappeler la définiton d'un tas binaire
2. Rappeler les relations liant l'indice d'un parent à ceux de ses fils (lorsqu'ils existent) lorsqu'on représente un tas binaire par un tableau
3. Rappeler le principe de l'insertion d'un nouvel élément dans un tas binaire
4. Rappeler le principe de la suppression de l'élément minimal d'un un tas binaire
5. Détailler les étapes et le fonctionnement de l'algorithme du tri par tas pour trier le tableau `[7; 15; 3; 4; 19; 11]`
6. On rappelle ci-dessous la définition du type représentant un tas binaire en OCaml :
   ```
   type 'a heap = {mutable size : int; data : 'a array};;
   ```
   Ecrire une fonction `insere` de signature `'a -> 'a heap -> unit` qui permet d'insérer une nouvelle valeur dans un tas binaire.