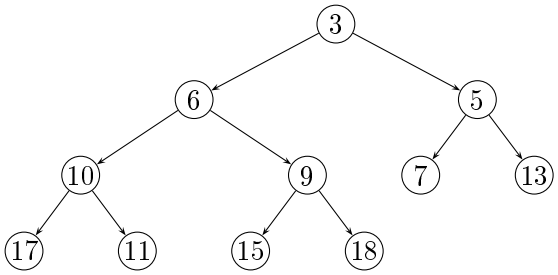
❑ **Exercice 1** : *Insertion dans un tas*

1. Rappeler la défintion d'un tas binaire (min)
2. On suppose qu'un tas est représenté par un tableau $t = (t_0, \dots t_{n-1})$. Lorsqu'ils existent quels sont les indices des fils de t_i ?
3. Quel est l'indice (lorsqu'il existe) du père de t_i ?
4. Rappeler le principe d'insertion d'un nouvel élément dans un tas binaire
5. Vérifier que l'arbre binaire suivant possède bien la structure de tas :



6. Détailler les étapes de l'insertion de 4 dans le tas précédent.
7. On rappelle les structures de données vues en cours et permettant de représenter un tas :
   — En langage C :

```c
struct heap
{
    int size;
    int capacity;
    int *tab;
};
typedef struct heap heap;
```

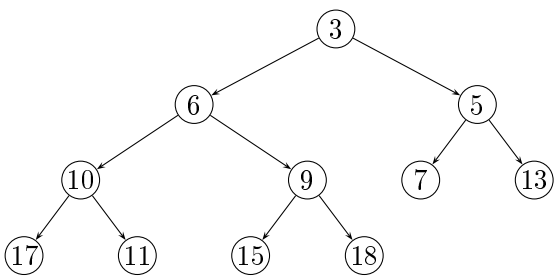
   — En OCaml :

```ocaml
type heap_int = {mutable size : int; data : int array};;
```

   Dans le langage de votre choix, implémenter l'algorithme d'insertion dans un tas.

❑ **Exercice 2** : *Extraction du minimum dans un tas*

1. Rappeler la défintion d'un tas binaire (min)
2. On suppose qu'un tas est représenté par un tableau $t = (t_0, \dots t_{n-1})$. Lorsqu'ils existent quels sont les indices des fils de t_i ?
3. Quel est l'indice (lorsqu'il existe) du père de t_i ?
4. Rappeler le principe d'extraction du minimum d'un tas binaire
5. Vérifier que l'arbre binaire suivant possède bien la structure de tas :



6. Détailler les étapes de l'extraction du minimum dans le tas précédent.
7. On rappelle les structures de données vues en cours et permettant de représenter un tas :
   — En langage C :

```c
struct heap
{
    int size;
    int capacity;
    int *tab;
};
typedef struct heap heap;
```

— En OCaml :

```ocaml
type heap_int = {mutable size : int; data : int array};;
```

Dans le langage de votre choix, implémenter l'algorithme d'extraction du minimum dans un tas.

❑ **Exercice 3** : *Ensemble inductif*

Soit X l'ensemble construit par induction à partir :

— des axiomes $X_0 = 0$

— des règles d'inférence $r_1 : n \mapsto n+5$, $r_2 : n \mapsto n+2$ et $r_3 : n \mapsto -n$

1. En donnant une suite de règle permettant de l'obtenir, montrer que $11 \in X$.
2. La définition de X est-elle ambigüe ?
3. Prouver par induction structurelle que $X \subset \mathbb{Z}$.
4. Montrer que $X = \mathbb{Z}$.
5. On définit en OCaml le type :

```ocaml
type nb =
| Z
| R1 of nb
| R2 of nb
| R3 of nb;;
```

Ecrire une fonction `res nb -> int` qui prend en renvoie l'entier associée à la suite d'applications des règles d'inférences. Par exemple `res (R3 (R1 (R2 Z)))` renvoie -7.

❑ **Exercice 4** : *Recherche d'un pic par la méthode diviser pour régner*

On dit qu'un tableau t de taille n présente un *pic* en position p si et seulement si :

— toutes les valeurs de t sont distinctes

— le sous tableau $t_0, \dots t_p$ est trié par ordre croissant

— le sous tableau $t_p, \dots t_{n-1}$ est trié par ordre décroissant.

1. Justifier rapidement que si un tableau présente un pic alors ce pic est unique.
2. Ecrire dans le langage de votre choix, une fonction itérative de complexité linéaire qui prend en entrée un tableau présentant un pic et renvoyant la position de ce pic.
3. On veut maintenant utiliser une méthode diviser pour régner.
   a) Donner le cas de base et sa solution
   b) Si on divise le tableau en deux moitiés, montrer qu'on peut déterminer dans laquelle se trouve le pic.
   c) Donner l'implémentation de la méthode diviser pour régner dans le langage de votre choix.
   d) Donner sa complexité.

❑ **Exercice 5** : *Conjecture de Sloanne*

En multipliant entre eux les chiffres d'un entier n, on obtient un nouvel entier n', on reitère ce processus sur n' aussi longtemps que le produit obtenu contient plus d'un chiffre.

1. Tester les entiers 524 et 6 787 832 en précisant le nombre d'étapes.
2. Prouver que ce processus termine.

3. Ecrire en OCaml une fonction `nb_to_lst : int -> int list` prenant comme paramètre un entier et renvoyant la liste de ses chiffres.
4. Ecrire une fonction `vol` prenant un paramètre `n` entier et renvoyant le nombre d'itérations.
5. Que pensez-vous de la conjecture : « le nombre d'itérations est inférieur à 9 ».

❑ Exercice 6 : *Rotation d'arbre*

1. Rappeler la définition d'un arbre binaire de recherche. On rappelle l'implémentation des arbres en OCaml utilisée en cours :

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

2. Représenter l'ac abr `sept` définie par :

```
let quatre = Noeud(Vide,4,Vide) in
let six = Noeud(Vide,6,Vide) in
let cinq = Noeud(quatre,5,six) in
let deux = Noeud(Vide,2,Vide) in
let trois = Noeud(deux,3,cinq) in
let neuf = Noeud(Vide,9,Vide) in
let sept = Noeud(trois,7,neuf) in
```

3. Rappeler la définition de la rotation droite et proposer une implémentation. Par exemple la rotation droite de l'arbre `sept` doit renvoyer `Noeud (Noeud (Noeud (Vide, 2, Vide), 3, Noeud (Noeud (Noeud (Vide, 4, Vide), 5, Noeud (Vide, 6, Vide)), 7, Noeud (Vide, 9, Vide))`
4. Proposer une implémentation de la rotation gauche
5. Avec un backtracking, proposer une implémentation de la fonction complet prenant comme en paramètre un abr et renvoyant un abr complet. Pour choisir la bonne voie on minimisera la hauteur.

❑ Exercice 7 : *Recherche de carrés magiques en retour sur trace*

Pour un entier n quelconque, le but de l'exercice est de programmer en C une recherche en *backtracking* d'une solution au problème du placement des entiers $1, \dots, n^2$ entiers dans un carré de côté n afin de former un carré magique.

1. Vérifier que dans le cas $n = 3$ le carré suivant est une solution :

2	7	6
9	5	1
4	3	8

2. Quelle serait la somme de chacune des lignes, colonnes ou diagonales dans le cas $n = 4$?
3. Donner l'expression de la somme de chacune des lignes, colonnes, ou diagonales pour un entier n quelconque.

   Afin d'implémenter la résolution en C, on propose de linéariser le carré en le représentant par un tableau à une seule dimension. Le carré ci-dessus est par exemple représenté par :
   `int carre[9] = {2, 7, 6, 9, 5, 1, 4, 3, 8}`
4. Donner l'expression de l'indice d'un élément situé en ligne i, colonne j dans le tableau linéarisé.
5. Inversement, donner la ligne et la colonne dans le carré initial d'un élément situé à l'indice i dans le tableau linéarisé.

   On représente par l'entier 0 une case non encore rempli du tableau, par exemple :
   `int carre[9] = {1, 8, 4, 0, 0, 0, 0, 0, 0}` représente un carré ayant simplement la première ligne complète
6. Ecrire une fonction `bool valide_ligne(int carre[], int lig, int size, int somme)` qui vérifie que la ligne `lig` d'un carré en cours de construction est encore valide, c'est le cas si cette ligne contient un zéro (car elle est alors incomplète) ou si elle ne contient aucun zéro et que sa somme est égale à la variable `somme` fournie en argument.

On suppose écrite les fonctions correspondantes pour les colonnes et les lignes de même qu'une fonction : `bool valide_carre(int carre[], int size, int somme)` qui vérife qu'un carré en cours de construction est encore valide.

7. Ecrire la fonction permettant de rechercher par backtracking un carré magique solution du problème posé. On pourra passer en argument à cette fonction un tableau de booléens de taille n^2 dont l'élément d'indice i indique si l'entier i a déjà été placé ou non dans le carré.