Exercice 1 :

1. Un arbre binaire de recherche est un arbre binaire A dont les nœuds étiqueté par des éléments d'un ensemble E totalement ordonné, tel que pour tout nœud Noeud(G, x, D) de A, x est strictement plus grand que toute étiquette de G et strictement plus petit que toute étiquette de D.

   Ceci revient à dire que les étiquettes prises dans l'ordre préfixe forme un tuple strictement croissant.

```
2 let rec mini a = match a with
  | Vide -> failwith "arbre vide"
  | Noeud(Vide, x, _) -> x
4 | Noeud(gauche, _, _) -> mini gauche
```

3. (a) Supposons que x n'est pas une clé de a. Alors x n'est une clé d'aucun sous-arbre de a, et les seuls cas possibles lors de l'éxécution de `remove` sont la ligne 3, la ligne 5 et la ligne 6. Lors des appels récursifs ligne 5 ou ligne 6, l'appel vérifie encore la condition que x n'est pas une clé de l'ABR.

   Supposons maintenant que x est une clé de a et que l'unique nœud de a d'étiquette x a au plus un enfant. Alors les seuls cas possibles lors de l'exécution de `remove` sont les lignes 5, 6, 7 et 8. De plus, lors d'un appel récursif à la ligne 5 ou la ligne 6, la définition des ABR garantit que l'unique nœud de clé x est bein dans l'ABR sur lequel on fait l'appel récursif.

   Par ailleurs, les lignes 3, 7 et 8 ne contiennent qu'un nombre constant d'opérations atomiques. Les lignes 5 et 6 contiennent un nombre constant d'opérations atomiques et un appel récursif sur un des sous-arbres de a. Les tests effectués sont également de coût constant.

   Donc si on note $C_1(h)$ la complexité de `remove` dans le pire cas avec l'hypothèse que x n'est pas une clé de a, ou bien que x est une clé de a et que l'unique nœud de a d'étiquette x a au plus un enfant, on a $C_1(h) \leqslant K_1 + C_1(h-1)$, avec K_1 constante. Donc $C_1(h) = \mathcal{O}(h)$.

   (b) Il suffit essentiellement de montrer que si on est dans le cas de la ligne 9, on a une complexité $\mathcal{O}(h)$.

   Soit $a = \text{Noeud}(g, x, d)$, avec g et d tous deux non vides.

   `remove a x` va d'abord effectuer un nombre constant d'opérations atomiques pour les tests, puis appeler `mini` sur d. La complexité de cette opération est $\mathcal{O}(h)$.

   Il y a ensuite un appel récursif `remove a z`. Or z est le minimum de d : c'est donc l'étiquette du nœud situé le plus à gauche de d. En particulier, le nœud d'étiquette z n'a pas d'enfant gauche, et on est donc dans le cas précédent, de complexité $C_1(h) = \mathcal{O}(h)$.

   On a donc deux appels de fonction de coûts linéaire en h : il existe une constante K_2 telle que `remove a x` fasse au plus $K_2 h$ opérations atomiques lorsque x est l'étiquette de la racine de a.

   En généralisant à tous les cas, comme précédemment on a que les lignes 3, 7 et 8 terminent en temps constant, les lignes 5 et 6 font un appel récursif sur un arbre de hauteur au plus $h-1$, et la ligne 9 termine en temps au plus $K_2 h + K_3$.

   Donc si $C(h)$ est la complexité dans le pire cas pour un arbre de hauteur h, on a que pour $h \geqslant 1$, $C(h) \leqslant \max(K_1 + C(h-1), K_2 h + K_3)$. Donc on a bien $C(h) \leqslant \max(K_1, K_2)h + \max(C(0), K_3)$, donc $C(h) = \mathcal{O}(h)$.

```
1 let rec remove a x =
2   match a with
  | Vide -> Vide
4 | Noeud (gauche, y, droit) ->
      if y < x then Noeud (gauche, y, remove droit x)
6     else if x < y then Noeud (remove gauche x, y, droit)
      else (* y = x *) if gauche = Vide then droit
8     else (* y = x *) if droit = Vide then gauche
      else (* y = x et les deux sous-arbres sont non vides *)
10      let z = mini droit in
        Noeud (gauche, z, remove droit z)
```

Exercice 2 : **Extrait de E3A 2017**

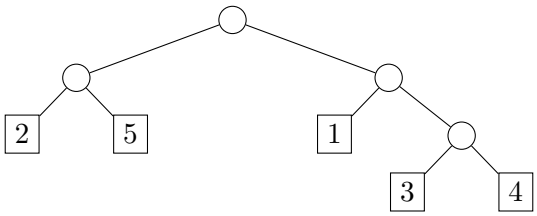

FIGURE 1 – Un peigne à cinq feuilles

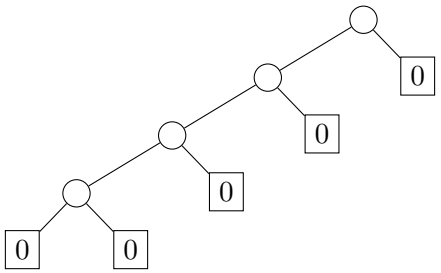

FIGURE 2 – Un peigne rangé à cinq feuilles

1. On représente un tel peigne sur la figure 2.
2. Montrons par récurrence sur $n \geqslant 1$ qu'un peigne rangé à n feuilles est de hauteur $n-1$:
   - C'est par définition vrai pour $n = 1$: le peigne réduit à une feuille est de hauteur 0.
   - Soit $n \geqslant 2$, on suppose la propriété vraie pour $n-1$. Considérons A un peigne rangé de taille n. Alors ce peigne n'est pas réduit à une feuille, donc il est de la forme $\text{Noeud}(A_g, F)$, où F est une feuille.

     Tous les nœuds internes de A_g sont aussi des nœuds internes de A, donc A_g est un peigne rangé, qui a $n-1$ feuilles (toutes les feuilles de A, sauf F).

     Donc par HR $h(A_g) = n-2$.

     Or $h(A) = 1 + \max(h(A_g), h(F)) = 1 + \max(n-2, 0) = n-1$ car $n \geqslant 2$.

```
let rec est_peigne_range a =
  match a with
  | Feuille _  -> true
  | Noeud (gauche, Feuille _) -> est_peigne_range gauche
  | _ -> false
```

```
let rec est_peigne_strict a =
  match a with
  | Feuille _  -> true
  | Noeud (gauche, Feuille _) -> est_peigne_strict gauche
  | Noeud (Feuille _, droit) -> est_peigne_strict droit
  | _ -> false
```

```
let est_peigne a =
  match a with
  | Feuille _ -> true
  | Noeud (gauche, droit) -> est_peigne_strict gauche && est_peigne_strict droit
```

5. Le résultat est décrit dans la fgure 3.

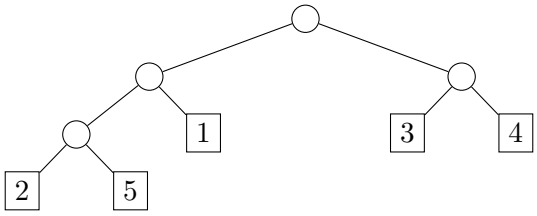

FIGURE 3 – Rotation gauche du peigne de la figure 1

6. Soit $A = \text{Noeud}(G, D)$ un peigne non réduit à une feuille, dont le sous-arbre droit D n'est pas une feuille. G et D sont des peignes stricts, par la définition des peignes.
   Alors D est de la forme $\text{Noeud}(A_2, F)$ ou $\text{Noeud}(F, A_2)$, où F est une feuille et A_2 un peigne strict.
   La rotation gauche de A est, dans les deux cas, $A' = \text{Noeud}(\text{Noeud}(G, F), A_2)$. Les nœuds internes de A' sont :
   - les nœuds internes de G et de A_2, qui ont nécessairement au moins une feuille parmi leurs enfants, car G et A_2 sont des peignes stricts ;
   - le nœud $\text{Noeud}(G, F)$, qui a la feuille F parmi ses enfants.
   - La racine, sur laquelle il n'y a pas de contrainte.

   A' est donc bien un peigne.

```
let rotation a =
  match a with
  | Feuille _ -> a
  | Noeud (_, Feuille _) -> a
  | Noeud (a1, Noeud (Feuille n, a2)) -> Noeud (Noeud (a1, Feuille n), a2)
  | Noeud (a1, Noeud (a2, Feuille n)) -> Noeud (Noeud (a1, Feuille n), a2)
  | _ -> failwith "ce n'est pas un peigne"
```

On note que la fonction ne vérifie pas en profondeur que a est un peigne : c'est seulement si son enfant droit à comme enfant deux nœuds internes qu'on déclenche une erreur.

```
let rec rangement a =
  match a with
  | Feuille _ -> a
  | Noeud (gauche, Feuille n) -> Noeud (rangement gauche, Feuille n)
  | _ (* l'enfant droit de a n'est pas une feuille *) -> rangement (rotation a)
```

Remarque : la fonction est de complexité dans le pire cas linéaire en la taille du peigne, même si ce n'est pas immédiatement évident à prouver (on pôurrait croire que la complexité est quadratique si on ne tient pas compte du fait que `rotation` renvoit toujours un peigne dont l'enfant gauche a une feuille comme enfant droit).

Il n'était évidemment pas demandé de prouver la complexité.

9. La présence de la rotation fait qu'on ne peut pas procéder par induction structurelle sur les arbres : en effet on n'a pas une relation d'ordre entre un peigne et sa rotation gauche.
   On va donc procéder par une double récurrence, d'abord $n \geqslant 1$ le nombre de feuilles de a

- *Init :* Si $n = 1$, a est réduit à une feuille et est déjà rangé.
- *Hérédité :* Soit $n \geqslant 2$. On suppose que `rangement` est correcte sur tout peigne avec $n-1$ feuilles (HR1).
  Montrons par récurrence sur $m \geqslant 1$ le nombre de feuilles de son sous-arbre droit que `rangement` est correcte sur tout peigne a à n feuilles dont m dans le sous-arbre droit :
  - *Init :* Si $m = 1$, alors a est de la forme Noeud(g, f) avec f une feuille. g a $n-1$ feuilles.
    Donc par lhypothèse de récurrence (HR1) `rangement` est correcte sur g et renvoie g' un peigne rangé, et `rangement a` renvoie donc Noeud(g', f), qui est bien un peigne rangé.
  - *Hérédité :* Soit $m \geqslant 2$. On suppose que `rangement` est correcte sur tout peigne avec n feuilles dont $m-1$ dans le sous-arbre droit (HR2).
    Soit a un peigne a n feuilles dont m dans le sous-arbre droit. Alors son sous-arbre droit n'est pas une feuille car $a \geqslant 2$, donc `rangement a` renvoie `rangement (rotation a)`.
    Or `rotation a` a n feuilles, dont $m-1$ dans le sous-arbre droit, puisque l'une d'entre elles a été déplacée. Donc par l'hypothèse de récurrence HR2, `rangement` est correcte sur cet arbre et donc correcte sur a.

Ceci conclut la preuve de la correction de `rangement`

Exercice 3 : **Extrait de E3A 2018**

1. Soit $a_m a_{m-1} \dots a_1 a_0$ l'écriture binaire de $n \in \mathbb{N}^*$. On a $n = \sum_{i=0}^{m} a_i 2^i$.
   Soit v le plus petit entier tel que $a_v \neq 0$ (c'est à dire, $a_v = 1$). v existe car n est non nul. Alors :

$$\begin{aligned} n &= \sum_{i=0}^{v-1} 0 \times 2^i + 2^v + \sum_{i=v+1}^{m} a_i 2^i \\ &= 2^v + 2^{v+1} \sum_{i=v+1}^{m} a_i 2^{i-v-1} \\ &= 2^v \left(1 + 2 \sum_{i=v+1}^{m} a_i 2^{i-v-1}\right) \end{aligned}$$

   Or $k = \left(1 + 2 \sum_{i=v+1}^{m} a_i 2^{i-v-1}\right)$ est impair car tous les 2^{i-v-1} sont entiers, donc on a bien $n = 2^v k$ avec k impair.
   Donc la valuation de n est le rang du 1 de poids le plus faible dans son écriture binaire (alternativement : le nombre de 0 à la fin de son écriture binaire), et son résidu est le nombre dont l'écriture binaire s'obtient en enlevant les 0 de poids faible dans l'écriture binaire de n.
2. 11000000 se termine par six 0, donc la valuation de 192 est 6.
   Son résidu est le nombre qui s'écrit 11 en binaire, c'est à dire 3.

3.
```
let rec residu n =
  if n = 0 || n mod 2 = 1 then n
  else residu (n / 2)
```

4.
```
let rec pgcd a b =
  if b = 0 then a
  else pgcd (min a b) (abs (a - b))
```

5. (a) a et b sont positifs. De plus, si $a > 0$, soit $b = 0$ et l'algorithme termine, soit $b > 0$ auquel cas un appel récursif se fait avec en premier argument $min(a, b) > 0$.
   Donc tout appel récursif de `pgcd` vérifie encore que $a > 0$.
   Soit $C(n)$ la complexité dans le pire cas en fonction de $n = \max(a, b)$.
   Si $a = b$, alors `pgcd a a = pgcd a 0 = a` termine en temps constant. De même si $b = 0$. Soit K_1 une majoration de la complexité dans ces deux cas.
   Dans tous les autres cas, on a $|b - a| < \max(a, b)$ (car $a > 0$ et $b > 0$) et $\min(a, b) < \max(a, b)$ (car $a \neq b$). Donc les deux arguments sont au plus $n - 1$. Soit K_2 une majoration de la complexité des opérations autre que l'appel récursif.
   Donc pour $n \geqslant 1, C(n) \leqslant \max(K_1, K_2 + C(n-1))$, où K_1 et K_2 sont des constantes. On a donc bien $C(n) = \mathcal{O}(n)$.

   (b) Soit $n \in \mathbb{N}^*$. On pose $a = 1$ et $b = n$.
   Alors le calcul donne `pgcd 1 n = pgcd 1 (n - 1)`. On a donc bien $C(n) = K + C(n-1)$ dans ce cas particulier.

6. Si $a = b$, on a $|a - b| = 0$ et donc à l'appel suivant de l'algorithme on renvoie $\min(a, b) = a = b$.
   Supposons désormais que a et b sont deux entiers pairs distincts, posons $a = 2^v k$ et $b = 2^{v'} k'$, avec v et v' entiers et k et k' entiers impairs.

   - **Cas 1 :** $v \neq v'$. On suppose sans perte de généralité que $v' > v$.
     Alors $|a - b| = 2^v |k - 2^{v'-v} k'|$, donc le résidu de $|a - b|$ est $|k - 2^{v'-v} k'|$, qui est impair. D'après l'hypothèse, l'algorithme calcule donc le pgcd de $\min(a, b)$ et de $|k - 2^{v'-v} k'|$.
     $\text{pgcd}(a, b) = 2^v pgcd(k, k')$, avec $pgcd(k, k')$ impair : c'est le résidu du pgcd.
     Or $\text{pgcd}(2^v k, k - 2^{v'-v} k') = \text{pgcd}(2^{v'} k', k - 2^{v'-v} k') = \text{pgcd}(k, k')$ (car l'un des deux termes est impair donc tout diviseur commun est impair).
     Donc on calcule le résidu du pgcd de a et b.
   - **Cas 2 :** $v = v'$. Alors $|a - b| = 2^v |k - k'|$, mais $k - k'$ est pair et non nul.
     Supposons sans perte de généralité $a < b$ et donc $k < k'$ (car $v = v'$), et posons $k' - k = 2^w k''$, avec k'' impair.
     Pour les mêmes raisons que précédemment, on calcule le pgcd de $\min(a, b) = a = 2^v k$ et de k''.
     En simplifiant les facteurs pairs, on trouve qu'il s'agit de $\text{pgcd}(k, k'')$. Or $\text{pgcd}(k, k') = \text{pgcd}(k, k' - k) = \text{pgcd}(k, 2^w k'') = \text{pgcd}(k, k'')$.
     Donc encore une fois on calcule le pgcd de k et k', c'est à dire le résidu du pgcd de a et b.

7. Soit $m \in \mathbb{N}^*$ pair, a et b des entiers naturels impairs tels que $m = a + b$.
   On suppose sans perte de généralité $a \leqslant b$.
   L'appel récursif se fait sur a et le résidu de $b - a$. Or $b - a$ est pair, donc son résidu est au plus $\dfrac{b-a}{2}$.
   Donc la somme des arguments de l'appel récursif est au plus $a + \dfrac{b-a}{2} = \dfrac{m}{2}$.
   Comme toutes les opérations autres que l'appel récursif sont de coût au plus constant logarithmique en m **(le calcul du résidu n'est PAS de coût constant)**, il existe K tel que $\forall m \geqslant 4$, $C'(m) \leqslant \max\limits_{\substack{2 \leqslant k \leqslant \frac{m}{2} \\ k \text{ pair}}} C'(k) + K \log_2(m)$
   Mes excuses pour l'erreur dans le sujet qui laissait entendre qu'il fallait faire quelque chose de plus compliqué....

8. Posons $K' = \max(K; C'(2))$ et montrons par récurrence sur $m \geqslant 2$ que $\forall m \geqslant 4$ pair, $C'(m) \leqslant K' \log_2^2(m)$.
   Erreur d'énoncé
   Initialisation : Par définition de K', $C'(2) \leqslant K' = K' \log_2^2(2)$.
   Hérédité : Soit $m \geqslant 4$. Supposons $\forall m' \in [\![2; n-2]\!]$ pair, $C'(m') \leqslant \log_2^2(m')$.
   On a $C'(m) \leqslant \max\limits_{\substack{2 \leqslant k \leqslant \frac{m}{2} \\ k \text{ pair}}} C'(k) + K \log_2(m)$

Or tous les $k \in [\![2; \frac{m}{2}]\!]$ sont couverts par l'hypothèse de récurrence. Donc :

$$\begin{aligned}
C'(m) &\leqslant \max_{\substack{2\leqslant k\leqslant \frac{m}{2} \\ k \text{ pair}}} C'(k) + K\log_2(m) \\
&\leqslant \max_{\substack{2\leqslant k\leqslant \frac{m}{2} \\ k \text{ pair}}} (K'\log_2^2(k)) + K\log_2(m) \\
&\leqslant K'\log_2^2\left(\frac{m}{2}\right) + K\log_2(m) \qquad \text{par croissance de } \log_2 \\
&\leqslant K'(\log_2(m) - K'\log_2(2))^2 + K\log_2(m) \\
&\leqslant K'\log_2(m) - (2K' - K)\log_2(m) + 1 \\
&\leqslant K'\log_2(m)
\end{aligned}$$

On a prouvé l'initialisation et l'hérédité, donc on a bien $\forall m \geqslant 2$ pair, $C'(m) \leqslant K'\log_2^2(m)$. En pariculier, $C'(m) = \mathcal{O}\left(\log^2(m)\right)$.

À noter que comme $\max(a,b) \geqslant \dfrac{a+b}{2}$, ceci s'applique également à la complexité en fonction de $n = \max(a,b)$ (mais la preuve est plus longue).

```
9 let pgcd_chinois a b =
2   let k1 = residu a in
    let k2 = residu b in
4   let facteur_pair = min (a / k1) (b / k2) in
    let rec pgcd_aux a b =
6     if b = 0 then a
      else pgcd (min a b) (residu (abs (a - b)))
8   in
    facteur_pair * pgcd_aux k1 k2
```

On a justifié plus haut que l'appel à `pgcd_aux` était de complexité $\mathcal{O}\left(\log^2(k_1+k_2)\right) = \mathcal{O}\left(\log^2(a+b)\right)$ car les deux sont impairs.

Le calcul du facteur pair nécessite deux calculs de résidu, de complexité au plus logarithmique.

Donc la complexité de la fonction est au plus $\mathcal{O}\left(log^2(a+b)\right)$

À noter : on peut éliminer le facteur carré mais ceci complique en réalité les choses. Je ne retirerai pas de point si vous avez (à tort) compté la recherche du résidu comme une opération en temps constant, ce qui permet d'arriver à la valeur recherchée en gardant le $\dfrac{m}{2}$ au lieu du $\dfrac{3m}{4}$ inutile.