TD - Preuves d'algorithmes
Terminaison - Complexité.

Exercice 1 (Échauffement - Notations de Landau).
Simplifier les écritures suivantes :

1. $O(n+1)$

2. $O(3n+3)$

3. $O(\frac{n(n+1)}{2})$

4. $O(n^2 e^{42} + 3 \times 2^{3n-1})$

5. $O(\log_2(n+1) + \ln(2n^2))$

Exercice 2 (Échauffement - Complexité des boucles).
Dans toutes les questions, n et $m < \frac{n}{2}$ désignent des entiers naturels. i, j, k sont des entiers préalablement déclarés. x est un entier préalablement défini. En détaillant vos calculs, déterminer la complexité temporelle de chacun des codes suivants.

```
// code 1
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    x = x + 1;

// code 2
for (i = 0; i < n; i++)
  for (j = 0; j < i; j++)
    x = x + 1;

// code 3
for (i = m; i < n-m; i++) {
  for (j = i-m; j < i+m; j++)
    x = x + 1;

// code 4
for (i = 0; i < n; i++)
  for (j = 0; j < i; j++)
    for (k = 0; k < j; k++)
      x = x + 1;
```

```
// code 5
int i = n;
while (i > 1) {
  x = x + 1;
  i = i / 2;
}

// code 6
i = n;
while (i > 1) {
  for (j = 0; j < n; j++)
    x = x + 1;
  i = i / 2;
}

# code 7
i = n;
while (i > 1) {
  for (j = 0; j < i; j++)
    x = x + 1;
  i = i / 2;
}
```

Exercice 3 (Multiplication russe).

1. Cours : réécrire l'algorithme de multiplication russe sous forme récursive **terminale** en OCaml.

2. Prouver la terminaison de cet algorithme.

3. Démontrer que :

$$\forall x \in \mathbb{R}, \forall q \in \mathbb{N}^*, \left\lfloor \frac{\lfloor x \rfloor}{q} \right\rfloor = \left\lfloor \frac{x}{q} \right\rfloor$$

4. En déduire une formule explicite pour le variant choisi.

5. Combien d'appels récursifs sont nécessaires pour que l'algorithme termine ?

6. Qu'en déduire sur la complexité temporelle de cet algorithme ?

Exercice 4 (Complexités récursives).
On considère l'algorithme d'exponentiation rapide **récursif**.

1. Écrire rapidement l'algorithme en OCaml
2. On appelle $C(n)$ la complexité (nombre de multiplications) pour un exposant n. Donner une relation de récurrence sur $C(n)$.
3. On se restreint au cas où n est une puissance de 2. Donner alors une expression de $C(n)$
4. On traite maintenant le cas général. Pour cela, on va traiter simultanément toutes les valeurs d'exposant n situées entre les mêmes puissances de 2 : $2^k \leq n < 2^{k+1}$. Conjecturer un encadrement de $C(n)$ pour ces valeurs de n.
5. Prouver cet encadrement pour toutes les valeurs d'exposant possibles
6. Que peut-on en conclure sur la complexité de l'algorithme d'exponentiation rapide ?

Exercice 5 (Recherche dichotomique).
On considère ci-dessous une implémentation possible de l'algorithme de recherche dichotomique.

```
let recherche_dichotomique v t =

  let l = ref 0 and
      r = ref ( Array.length t ) and
      trouve   = ref false and
      idx_loc = ref (-1) in

  while (l < r && !trouve = false) do

    let m = ((!r) + (!l) )/2 in
    let val_milieu = t.(m) in

    if (val_milieu = v) then
      (
        trouve := true;
        idx_loc := m
      )
    else
      (
        if (val_milieu > v) then
          r := m-1
        else
          l := m+1
      )
  done;
  !idx_loc
;;
```

1. Donner la complexité spatiale de cette fonction
2. Montrer que la complexité temporelle dans le pire des cas est un $O(n)$
3. Cette borne n'est cependant pas très serrée. On peut dire beaucoup mieux. Établir et démontrer une relation de récurrence sur la taille de la fenêtre de recherche.
4. En déduire une majoration de la taille de cette fenêtre de recherche au cours de l'algorithme
5. En déduire que la complexité dans le pire cas est en fait en $O(\log_2 n)$
6. On introduit la notation Θ. On dit qu'une fonction $g : \mathbb{N} \to \mathbb{R}^+$ est un Θ de $f : \mathbb{N} \to \mathbb{R}^+$, et l'on note $g(n) \in \Theta(f(n))$ s'il existe deux facteurs k_1, $k_2 \in \mathbb{R}^+$ et un rang $n_0 \in \mathbb{N}$ tels que

$$\forall n \geq n_0,\ k_1\, f(n) \leq g(n) \leq k_2\, f(n)$$

   Identifier un pire cas permettant de prouver que la complexité est en $\Theta(\log_2(n))$