❑ Exercice 1 : *Notation O*

1. Déterminer un O des suites de terme général :
   a) $2023n^2$
   b) $n^2 + 10^9 n$
   c) $3n + 7\log n$
   d) $2^{n+7} + n^{10}$
   e) $\sqrt{19n^2 + 3}$
2. Montrer que si $u_n = O(v_n)$ et $v_n = O(w_n)$ alors $u_n = O(w_n)$.
3. Montrer que $O(u_n + v_n) = O(\max(u_n, v_n))$.
4. Montrer que si $u_n = O(a_n)$ et $v_n = O(b_n)$ alors $u_n v_n = O(a_n b_n)$.
5. Déterminer un O (le « meilleur » possible) des expressions suivantes :
   a) $O(n^4) + O(n^2)$
   b) $O(n^5) + O(n^5)$
   c) $O(n^3) + O(\log(n))$
   d) $O(n^4) \times O(n^3)$
   e) $O(n^4) \times O(\sqrt{n})$

❑ Exercice 2 : *Vérification du tri*

1. Ecrire un algorithme permettant de vérifier qu'un tableau est trié par ordre croissant.
2. Prouver que votre algorithme est correct.
3. Déterminer sa complexité.

❑ Exercice 3 : *multiplier en additionnant*

```c
int mult(int n, int p){
    int prod = 0;
    while (p>0){
        prod = prod + n;
        p = p -1;}
    return prod;}
```

1. En supposant $p > 0$ montrer la terminaison.
2. Prouver que cette fonction renvoie $p \times n$.
3. Déterminer sa complexité.

❑ Exercice 4 : *exponentiation rapide*

On rappelle la fonction d'exponentiation rapide dans sa version récursive :

```c
float expo(float a, int n){
    float cp = a;
    float res = 1;
    while (n!=0){
        if (n%2==1){
            res = res*cp;}
        cp = cp*cp;
        n=n/2;}
    return res;}
```

1. Prouver que cet algorithme termine.
2. Prouver qu'il est correct.
   En notant n_0 la valeur initiale de n, on pourra considérer l'invariant suivant : `res * cp`$^{\texttt{n}}$$=a^{n_0}$
3. Donner sa complexité.