EA4 – Éléments d'algorithmique
TD n° 3 : complexité et récursivité

Exercice 1 : classement

Classer les fonctions suivantes en fonction de leur ordre de grandeur dans les classes Θ_1 à Θ_7 : les fonctions appartiennent à la même classe Θ_i si et seulement si elles sont du même ordre de grandeur, et les classes Θ_i sont rangées en ordre croissant.
Liste des fonctions à traiter (où log désigne le logarithme en base 2) :

$$n^2 + n^4, \quad n^2 + 4^n, \quad n^2 \times 4^n, \quad n^2 + \log n, \quad n^2 + \log(4^n),$$

$$\log(\sqrt{n}), \quad \log(n^4), \quad (\log n)^4, \quad 4^{n-1}, \quad 4^{2n}, \quad 2^{2n}, \quad 4^{\log n}$$

Θ_1	Θ_2	Θ_3	Θ_4	Θ_5	Θ_6	Θ_7

Exercice 2 : algorithme de Karatsuba

Dérouler à la main l'algorithme de Karatsuba vu en cours pour le calcul du produit 3210×1203. Tracer l'arbre des appels récursifs et prendre le produit de chiffres décimaux comme cas de base.

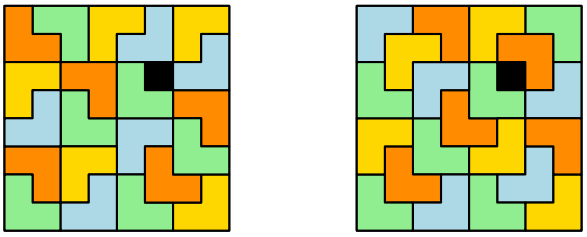
Exercice 3 : complexité d'algorithmes récursifs

Pour chacun des algorithmes `somme_`i`_` suivants, donner une relation de récurrence satisfaite par le nombre $A_i(n)$ d'additions effectuées pour une entrée de taille n, et en déduire l'ordre de grandeur de $A_i(n)$.

```
def somme_1_(T) :
  return 0 if len(T) == 0 else somme_1_(T[2:]) + 1

def somme_2_(T) :
  m = len(T)//2
  return 0 if len(T) == 0 else somme_2_(T[:m]) + 1

def somme_3_(T) :
  return 0 if len(T) == 0 else somme_1_(T) + somme_3_(T[1:])

def somme_4_(T) :
  m = len(T)//2
  return 0 if len(T) == 0 else somme_4_(T[:m]) + somme_4_(T[m:])
```

Exercice 4 : pavage avec des L

On considère une grille carrée de côté n dans laquelle une case i, j est « interdite » (avec $1 \leqslant i \leqslant n$ et $1 \leqslant j \leqslant n$). On suppose que n est une puissance de 2 ($n = 2^k$ pour un certain $k \geqslant 1$).
On veut paver la grille (excepté la case interdite) avec des tuiles de trois cases en forme de L, orientées dans n'importe quel sens.
Voici deux solutions, pour une grille de côté $8 = 2^3$ dont la case interdite est la case noircie :



Proposer une méthode basée sur la stratégie « diviser pour régner » en ramenant la résolution du problème de départ à la résolution de plusieurs sous-problèmes analogues sur des données de taille inférieure.
Évaluer ensuite le nombre d'appels récursif effectués, d'abord en fonction de k et ensuite en fonction de n.