Travaux dirigés n^o 3
Grands entiers

Nous allons analyser tout au long de ce TD des algorithmes qui prennent en entrée de grands entiers x, y, N, a, b ... représentés en mémoire sur n bits. Clairement, les opérations arithmétiques usuelles ne se font plus en temps constant $O(1)$.

Exercice 3.1 — Multiplication

Vous avez vu en cours un algorithme itératif pour multiplier deux grands entiers x et y. Voici un algorithme pour effectuer la même multiplication en récursif. Évaluez sa complexité en fonction de $n = \log_2(x) = \log_2(y)$.
Indication : considérez que la taille réelle de l'entrée m égale la somme des tailles des deux entiers.

```
   Multiply (entier x, entier y) {
1    si (y = 0) {
2      retourner 0
     }
3    z ← Multiply (x, y div 2)
4    si (y mod 2 = 0) {
5      retourner 2 * z
     }
     sinon {
6      retourner x + 2 * z
     }
   }
```

Exercice 3.2 — Calcul de la factorielle

Voici un algorithme récursif pour le calcul de la factorielle d'un entier N. Quelle est la complexité de cet algorithme en fonction de N ? En déduire la complexité en fonction de la taille n de N.

```
   Fact_rec (entier N) {
1    si (N ≤ 1) {
2      retourner 1
     }
3    r ← Fact_rec(N−1)
4    retourner N * r
   }
```

Exercice 3.3 — Plus grand commun diviseur

Voici une version modernisée de l'algorithme d'Euclide calculant le PGCD de deux entiers a et b, on suppose que $a \geq b$. Quelle est la complexité de cet algorithme en fonction de la taille n commune à ces deux entiers ? On pourra utiliser (et montrer pour les plus rapides d'entre vous ...) que si a est supérieur ou égal à b alors a mod $b < a/2$.

```
   Euclide  (entier a, entier b) {
1    si (b = 0) {
2      retourner a
     }
3    retourner Euclide (b, a mod b)
   }
```

Exercice 3.4

La suite de Fibonacci est définie comme suit : $\begin{cases} Fib(0) = 0 \\ Fib(1) = 1 \\ \forall N \geq 0,\ Fib(N+2) = Fib(N) + Fib(N+1) \end{cases}$

La suite croît très rapidement compte tenu qu'une approximation du terme $Fib(N)$ est $2^{0{,}694N}$. L'algorithme suivant calcule le $N^{\text{ème}}$ nombre de Fibonacci pour tout $N \geq 0$, l'appel initial étant $Fib\_rec(N, 0, 1)$.
Quelle est, en fonction de N, la complexité de l'algorithme récursif suivant ?

```
   Fib_rec (entier N, entier j, entier i) {
1    si (N = 0) {
2      retourner j
     }
3    retourner Fib_rec(N-1, j+i, j)
   }
```

Un algoritme naïf aurait consisté à effectuer deux appels récursifs. Évaluez sa complexité en fonction de N puis comparez-la à celle de l'algorithme précédent.