# Algorithmes — Exercices

### Exercice 1 : sous-vecteur minimal

*Etant donné un vecteur $v = (v_1, \ldots, v_n)$ de $\mathbf{Z}^n$ ($n \geq 2$), il faut trouver un sous-vecteur minimal de $v$, c'est-à-dire deux indices $g$ et $d$, $1 \leq g \leq d \leq n$, tels que la somme $v_g + v_{g+1} + \cdots + v_d$ soit la plus petite possible.*

Pour $1 \leq j \leq n$, on pose :

$$s(j) = \min_{1 \leq i \leq j} \{v_i + \cdots + v_j\}$$
$$i(j) = \max \{i \ / \ v_i + \cdots + v_j = s(j)\}$$

Calculer $s(1)$ et $i(1)$ et écrire les formules de récurrence donnant $s(j)$ et $i(j)$ en fonction de $s(j-1)$ et $i(j-1)$ lorsque $2 \leq j \leq n$. En déduire un algorithme optimal en $\Theta(n)$ additions et comparaisons. Ecrire la procédure correspondante, et tester.

### Exercice 2 : puissance rapide

1. Programmer une procédure `puissance(x,n)` calculant $x^n$ de façon récursive, en remarquant que :
   - si $n$ est pair, alors $x^n = (x.x)^{n/2}$,
   - si $n$ est impair, alors $x^n = x.x^{n-1}$.

2. Calculer le nombre de multiplications effectuées au cours du calcul, en fonction de $\lfloor \log_2 n \rfloor$ et de la somme des chiffres de l'écriture binaire de $n$.

### Exercice 3 : calcul récursif de $\binom{n}{p}$

*On veut écrire une fonction qui, étant donnés deux entiers $n$ et $p$, $0 \leq p \leq n$, renvoie la valeur du coefficient du binôme $\binom{n}{p}$, grâce à la formule de récurrence :*

$$\binom{n}{p} = \binom{n-1}{p-1} + \binom{n-1}{p}$$

1. Programmer et tester la fonction récursive suivante :

```
> C:=proc(n,p)  # calcule C(n,p) en supposant 0<=p<=n
      if p=0 or p=n then 1 else C(n-1,p-1)+C(n-1,p) end if
  end proc;
```

2. Montrer que la complexité en espace de cet algorithme est au pire $\Theta(n)$.

3. Calculer, en fonction de $n$ et $p$, le nombre d'appels à la fonction $C$ que provoque le calcul de $\binom{n}{p}$, puis le nombre d'additions effectuées. En déduire que la complexité en temps, dans le pire des cas, est exponentielle.

4. Ajouter à la procédure l'option `remember`. Comparer les performances, et expliquer en étudiant les complexités en espace et en temps de cette deuxième version.

5. Programmer une troisième version, optimale, en utilisant une autre formule de récurrence.