IUT de Montpellier **TD1**
M3103 Algorithmique avancée

Pour tous les exercices, on détaillera la spécification des algorithmes (action, prérequis), et on vérifiera que les appels respectent les prérequis.

1 Exercices de base sur les entiers

Exercice 1. Factorielle Ecrire un algorithme récursif `int factorielle(int n)` qui pour tout $n \in \mathbb{N}^*$ calcule $n!$.

Exercice 2. Pair Ecrire un algorithme récursif `boolean pair(int n)` qui pour tout $n \in \mathbb{N}$ retourne vrai ssi n est pair.

Exercice 3. Somme impairs Ecrire un algorithme récursif `int sommeImpairs(int n)` qui pour tout $n \geq 1$, n impair, calcule $\sum_{i=0}^{\frac{n-1}{2}} (2i+1)$.

Exercice 4. Puissance Ecrire un algorithme récursif `int puiss(int x, int n)` qui pour tout entier x et pour tout $n \in \mathbb{N}$ calcule x^n. Nous écrirons une autre version de cet algorithme de type "diviser pour régner" plus tard.

2 Exercices de base sur les tableaux

Exercice 5. Nombre d'occurences

Question 5.1.
Ecrire un algorithme récursif `int nbOccAux(int x, int []t, int i)` qui pour tout entier x, tableau t non vide et pour tout i **tel que** $0 \leq i < t.length$ calcule le nombre d'occurences de x dans le sous tableau $t[i..(t.length - 1)]$.

Question 5.2.
En déduire un algorithme (non récursif .. mais sans boucle !) `int nbOcc(int x, int []t)` qui calcule le nombre d'occurences de x dans t. Indication : appelez nbOccAux.

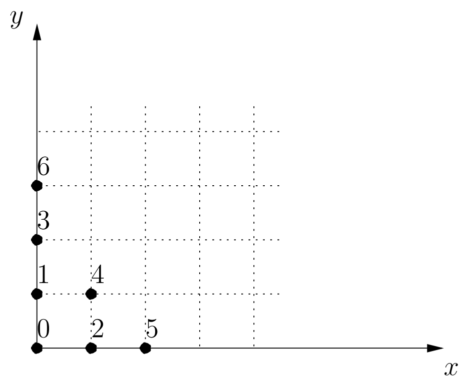
Question 5.3.
On remarque que pour l'instant nbOccAux ne supporte pas les tableaux vides (car il faudrait donner i tel que $0 \leq i < 0$). On va donc changer la spécification de nbOccAux. Ecrire un algorithme `int nbOccAux2(int x, int []t, int i)` qui pour tout entier x, tableau t et pour tout i **tel que** $0 \leq i \leq t.length$ calcule le nombre d'occurences de x dans le sous tableau $t[i..(t.length - 1)]$. Conclusion : nbOccAux2 supporte maintenant les tableaux vides, et on s'aperçoit que le cas de base était plus court, on est gagnant des deux côtés ! Il faudra donc s'habituder à ces spécifications où les indices ont le droit de sortir du tableau.

Exercice 6. Palindrome En s'inspirant de la démarche de l'exercice précédent (c'est à dire en écrivant un estPalindromeAux(.....) POUR LEQUEL VOUS INDIQUEREZ VOS PREREQUIS), écrire un algorithme `boolean estPalindrome(char []t)` qui détermine si t est un palindrome. Par exemple, $t = ['a', 'b', 'c', 'b', 'a']$ est un palindrome, $t = ['a', 'b', 'b', 'a']$ est un palindrome, mais $t = ['a', 'b', 'c', 'e', 'a']$ n'est pas un palindrome.

Exercice 7. Croissants En s'inspirant de la démarche de l'exercice précédent, écrire un algorithme `boolean croissants(int []t)` qui détermine si les éléments de t sont rangés par ordre croissant.

Exercices bonus

Exercice 8. Numérotation du plan On considère la fonction C qui à tout couple d'entiers (x, y) associe un entier comme indiqué sur le schéma.



Question 8.1.
Ecrire un algorithme récursif `int C(int x, int y)` qui pour tout $x \in \mathbb{N}$ et $y \in \mathbb{N}$ calcule $C(x, y)$

Question 8.2.
On admettra (voir dessin!) que C est une bijection, et donc que pour tout $n \in \mathbb{N}$ il existe unique couple (x, y) tel que $C(x, y) = n$. Un tel couple sera noté $C^{-1}(n)$. Ecrire un algorithme récursif `int[] invC(int n)` qui pour tout $n \in \mathbb{N}$ calcule le couple $C^{-1}(n)$ (sous la forme d'un tableau de taille 2 de la forme $[x, y]$)

Exercice 9. PGCD Nous allons écrire l'algorithme récursif d'Euclide pour calculer le PGCD de deux entiers naturels.

Question 9.1.
Soient a et b dans $\mathbb{N}$, avec $b > 0$. Soient q et r dans $\mathbb{N}$ tels que $a = bq + r$, avec $0 \leq r < b$. Montrer que pour tout x, (x divise a et x divise b) $\Leftrightarrow$ (x divise b et x divise r).

Question 9.2.
Soient a et b dans $\mathbb{N}$, avec $b > 0$. Montrer que $PGCD(a, b) = PGCD(b, r)$.

Question 9.3.
En déduire un algorithme récursif `int PGCD(int a,int b)` qui pour tout a et b dans $\mathbb{N}$ calcule le pgcd de a et b.

Exercice 10. Décomposition en facteurs premiers Le théorème fondamental de l'arithmétique nous dit que tout entier supérieur ou égal à 2 se décompose en un produit de nombres premiers. Par exemple :

- $18 = 2 \times 3 \times 3$
- $36 = 2 \times 2 \times 3 \times 3$
- $500 = 2 \times 2 \times 5 \times 5 \times 5$

Question 10.1.
Ecrire un algorithme `ArrayList<Integer> decompose(int x)` qui, étant donné un entier $x \geq 2$, retourne la liste de ses facteurs premiers dans la décomposition ci-dessus. Par exemple, `decompose(500)` doit retourner $[2, 2, 5, 5, 5]$ (l'ordre n'a pas d'importance).

On supposera que l'on dispose d'une fonction `int plusPetitDiv(int x)` qui, étant donné un entier $x \geq 2$, retourne le plus petit diviseur de x. On rappelle que l'on peut manipuler des listes ainsi :

- ArrayList<Integer> l = new ArrayList<Integer>();
- l.add(5);
- l.addAll(l2); //avec l2 une ArrayList<Integer>, va ajouter tous les éléments de l2 à la fin de l