Ce TD possède plusieurs objectifs.
Le premier est de vous entraîner à corriger seuls vos erreurs.
N'hésitez pas à poser quand même des questions, mais vous devez répondre vous même à la question «Pourquoi ça ne marche pas ? » (cf la feuille du même nom...).
L'autre objectif est de vous faire revenir sur d'anciens problèmes que vous n'auriez éventuellement pas eu le temps de résoudre et que nous estimons importants, afin de mettre en pratique la première partie.

1 Petit entraînement

Essayez **d'abord** de comprendre à quoi servent les procédures ci-dessous, puis corrigez la version « buggée » qui se trouve sur l'ordinateur. Il faut non seulement que Maple comprenne ce que vous lui définissez, mais également que la fonction obtenue fasse la même chose que l'algorithme : testez-là !

Algo1 :
Entrée : L, Liste de taille n
Pour i allant de 1 à n
Pour j allant de $i+1$ à n
Si L[i]=L[j]
Alors Renvoyer **vrai**
Renvoyer **faux**

(On rappelle que `Renvoyer` met fin à la procédure en cours).

Algo2 :
Entrée : n, entier
Si n=0
Alors Renvoyer 1
Sinon Renvoyer $n * Algo2(n-1)$

---

2 Applications

2.1 crible d'Eratosthène

Le crible d'Eratosthène est un algorithme pour trouver les nombres premiers inférieurs à N. La méthode est la suivante :

- on se donne une liste des entiers de 2 à N ;
- on prend le plus petit élément de la liste, et on raye tous ses multiples ;
- on prend l'entier non rayé suivant et on recommence.

Exercice 1 *En vous inspirant de cette méthode, écrivez une procédure* `Crible(N)` *qui renvoie la liste des nombres premiers inférieurs à N (aide : au lieu de "rayer une case", on peut remplacer l'élément concerné dans la liste par 0, et pour finir se servir d'une procédure éliminant les zéros d'une liste).*

2.2 Quelques suites célèbres

Exercice 2 (Nombres de Catalan) *On considère une suite finie de parenthèses, et on veut que cette suite soit correcte, autrement dit que chaque parenthèse fermante suive une parenthèse ouvrante.*
Par exemple $(((()(())))())$ est correcte - on dit que cette expression est bien parenthésée *-, mais $((\,$ ou bien $)((()))$ ne le sont pas.*
Le nombre d'expression bien parenthésées pour une suite de $2n$ parenthèses est noté c_n.
La suite c_n vérifie : $c_0 = 1$, $c_n = \sum_{k=0}^{n-1} c_k \cdot c_{n-k-1}$.

1. *Vérifiez cette forumule pour $n = 1, 2, 3$.*
2. *Ecrivez une procédure qui prend en argument n et retourne c_n.*
3. *Expliquez pourquoi une procédure récursive ne fonctionne que pour des n petits.*
4. *Modifiez votre algorithme en utilisant un tableau de taille n stockant au fur et à mesure les valeurs de c_k pour $k < n$.*

Exercice 3 (Le Problème de Syracuse) *Considérons la suite récurrente suivante :*

$$u_{n+1} = \begin{cases} u_n/2 & \text{si } u_n \text{ est un entier pair} \\ 3u_n + 1 & \text{si } u_n \text{ est un entier impair} \end{cases}$$

et où u_0 est un entier naturel quelconque.
Exemple : $7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, \ldots$
La conjecture de Syracuse (non démontrée à l'heure actuelle) est que l'on arrive toujours sur le cycle $4, 2, 1$. Notre but est de le vérifier pour quelques entiers.

1. *Définissez la fonction $f : u_n \mapsto u_{n+1}$ (pour tester la parité, utilisez* `mod`*).*
2. *Écrivez une procédure qui calcule les termes de la suite suivant l'entier u_0 que l'on s'est fixé et le nombre d'itérations n donné.*
3. *Modifiez cette procédure pour qu'elle donne le plus petit entier n tel que $u_n = 1$ - en supposant que celui-ci existe comme l'indique la conjecture.*

3 Algorithmes de Tri

Le but de cette section est de vous familiariser avec les principes algorithmiques liés aux différents tris usuels. Nous souhaitons donc programmer ici des algorithmes qui, étant donnée une liste d'entiers L, renvoient une nouvelle liste contenant les mêmes éléments triés par ordre croissant. Les algorithmes proposés sont ordonnés par difficulté croissante.

Complexité d'un algorithme : (à passer en première lecture)
La notion de complexité ne sera pas définie formellement ici. Intuitivement, nous dirons qu'un algorithme a pour complexité la fonction $f(n)$ si, étant donnée une entrée de taille n (ici une liste de longueur n), le nombre d'opérations que l'algorithme doit faire pour retourner le résultat est dans la classe $O(f(n))$. Les seules opérations que nous compterons ici seront les lectures des éléments de la liste. Exemple :

```
Entreé : L
Variables locales : m, l, i
l := longueur(L)
m := L[1]
Pour i entre 2 et l faire
      si m<L[i] alors m := L[i] fin si
fin boucle
Renvoyer m
Fin
```

Pour cet algorithme (que fait-il au fait ?), la complexité est $f(n) = 2n$. Comme c'est en fait la classe de complexité qui nous intéresse, nous retiendrons simplement que cet algorithme est de complexité $O(n)$ et dirons que c'est un algorithme linéaire.

Nous allons voir dans la suite que les algorithmes naïfs de tri sont en $O(n^2)$. Nous présenterons à la fin un algorithme en $O(n \log n)$. Il est en fait possible de montrer que cette borne de complexité est optimale, mais c'est un peu plus dur...

Exercice 4 (Sélection) *Ce tri est basé sur une idée simple : il est facile de déterminer la position du plus petit élément de L, et sa nouvelle position dans la liste triée est connue puisqu'il doit être en première position ! Nous pouvons donc* **échanger** *ces deux éléments (le premier et celui contenant le minimum de la liste). Il suffit ensuite d'itérer cette idée avec la suite de la liste. Ainsi, après i itérations, la sous liste $L[1..i]$ contient les i plus petits éléments de L rangés par ordre croissant, et la sous-liste $L[i+1..n]$ les éléments restant à trier.*

1. *Commencez par faire quelques exemples à la main pour bien comprendre l'algorithme.*
2. *Écrivez une procédure* `Indice-min(L)` *qui étant donnée une liste d'entiers retourne l'indice du plus petit élément de cette liste.*
3. *Écrivez une procédure* `Echange(L,i,j)` *qui étant donnés une liste* `L` *et deux positions* `i` *et* `j` *dans cette liste, renvoie une nouvelle liste définie de la manière naturelle...*
4. *Écrivez enfin une procédure* `Selection(L)` *qui étant donnée une liste renvoie la liste triée associée obtenue par le tri par sélection.*
5. *(facultatif) Évaluez la complexité de cet algorithme dans le pire cas et dans le meilleur cas.*

Exercice 5 (Insertion) *Ce tri est censé être le tri naturel pour trier un jeu de cartes. Le principe du tri est le suivant :*
On va ajouter successivement à leur place dans un tableau initialement vide tous les éléments de L. Il est en effet facile d'insérer à sa place un élément dans une liste déjà triée : il suffit d'avancer dans cette liste jusqu'à trouver un élément plus grand que l'élément à insérer.

1. *Commencez par faire quelques exemples à la main pour bien comprendre l'algorithme.*
2. *Écrivez une procédure* `Inserer(L,a)` *qui étant donnée une liste d'entiers* `L` *supposée triée et un entier* `a` *retourne la liste obtenue en insérant* `a` *à sa place dans* `L`.
3. *Écrivez enfin une procédure* `Insertion(L)` *qui étant donnée une liste renvoie la liste triée associée obtenue par le tri par insertion.*
4. *(facultatif) Évaluez la complexité de cet algorithme dans le pire cas, et dans le meilleur cas.*

Exercice 6 (Fusion) *La particularité de ce tri est d'utiliser une méthode* ***récursive****. En effet, on découpe la liste en deux sous-listes, puis on trie récursivement les deux sous-listes. Ensuite vient une phase de recomposition, appelée* **fusion***. Celle-ci consiste, étant données deux listes supposées triées, à fusionner ces deux listes en une liste triée.*

1. *Écrivez la procédure* `Fusion(L1,L2)` *définie ci-dessus.*
2. *Écrivez enfin une procédure* `Tri-fusion(L)` *qui étant donnée une liste renvoie la liste triée associée obtenue par le tri par fusion.*
3. *Évaluez la complexité de cet algorithme dans le pire cas et dans le meilleur cas.*

Exercice 7 (Quick Sort) *À nouveau, ce tri est récursif. Il consiste à choisir un élément arbitraire du tableau, appelé* pivot*. Il s'agit ensuite de réorganiser la liste L de sorte que tous les éléments inférieurs (resp. supérieurs) au pivot soient à gauche (resp. à droite) du pivot. Ainsi, le pivot est à sa place dans la future liste triée. On réapplique l'enfin l'algorithme aux deux sous-listes des éléments inférieurs et supérieurs au pivot. Pour fixer les idées, on dira que le pivot est le premier élément de la liste.*

1. *Écrivez les procédures* `Organise(L,a)` *et* `QuickSort(L)` *comme définies ci-dessus.*
2. *Evaluez la complexité de cet algorithme dans le pire cas et dans le meilleur cas. Décrivez les cas où sont atteints ces valeurs de la complexité.*