TD N°3

Objectifs : savoir résoudre des problèmes récurrents

Exercice 1 : Calcul du PGCD par la méthode d'Euclide
Ecrire une fonction récursive PGCD_Euc qui retourne le PGCD de 2 entiers A et B en utilisant l'algorithme d'Euclide qui s'appuie sur les propriétés suivantes :

PGCD(A,B) = B **Si B est un diviseur de A**
PGCD(A,B) = PGCD (B,A mod B) **Si non**

Exemple : PGCD(36,20) = PGCD(20,16) = PGCD(16,4) = 4.

Exercice 2 : Palindrome
Ecrire une fonction récursive Palind qui vérifie si une chaîne de caractères est un palindrome ou non.
Formellement, une chaîne S de n caractères est un palindrome si et seulement si :

$$\forall\, i, 1 \leq i \leq n \; div \; 2,\; S[i] = S[n - i + 1]$$

Une condition nécessaire est que les caractères extrêmes soient identiques et que la sous chaîne privée des caractères extrêmes soit également un palindrome.

Exercice 3 : Suite de Fibonacci.
La suite de Fibonacci est définie mathématiquement par la formule si dessous:

Fib(n)=1 **si n=1 ou n=2**
Fib(n)=Fib(n-1)+Fib(n-2) **si non (somme des deux derniers termes si n>=3)**

1- Écrire un algorithme qui permet de :

- saisir un entier n >= 1
- de calculer et d'afficher Fib(n) en utilisant :
  a) une fonction récursive
  b) une fonction itérative

2- Avec n = 30, quel est le processus le plus rapide ?

Exercice 4 : Fonction d'Ackermann
1- Ecrire une fonction récursive Ack qui calcule Ack (n,m) selon la formule suivante :

Ack(0,m) = m + 1
Ack(n,0) = Ack (n-1,1)
Ack(n,m) = Ack (n-1,Ack (n,m-1))

2- Vérifier que : Ack (1, n) = n + 2, Ack (2,n) ≈ 2 * n, Ack (3,n) ≈ 2n.

Exercice 5 : Fonction de MacCarthy
1- Ecrire une fonction récursive f qui calcule f (n) selon la formule suivante :

f(n) = n - 10 **Si n > 100**
f(n) = f(f(n+11)) **Sinon**

2- Calculer f(96).

Exercice 6 :

Ecrire la fonction récursive Factorielle de façon terminale et non terminale.

Exercice 7 :

Ecrire la fonction récursive Fibonacci de façon terminale et non terminale.

Exercice 8:

Ecrire la fonction (en récursive terminale et non terminale) qui recherche le maximum d'un tableau et retourne l'indice de la valeur maximale.

Si le tableau de longueur 0, vous retourner -1

Exercice 9:

Ecrire une méthode récursive terminale permettant de retourner le nombre d'occurrences d'un élément X dans un tableau d'entiers T de taille n transmis en argument d'entrée.

Exercice 10 :

Dérécursiver les solutions des Exercices 6, 7, 8 et 9

Exercice 11 :

Ecrire une méthode récursive qui retourne la somme des carrés des x premiers entiers si x > 0 ; -1 sinon.

$$sommeCarre(x) = \begin{cases} \sum_{i=1}^{x} i^2 & \text{si } x \geq 0 \\ -1 & \text{si } x < 0 \end{cases}$$

Exemple : on prend x = 4, le résultat retournera la valeur 30.

Exercice 12 : Recherche dichotomique

L'algorithme de recherche dichotomique permet la recherche d'une information x dans une table **triée** par ordre croissant sur une caractéristique. Cet algorithme est basé sur le principe suivant:

A- On compare la valeur x par rapport au milieu de notre espace de recherche, si sa coïncidence avec cet élément, alors on s'arrête avec un résultat positif.
B- sinon on s'intéresse
B1) soit au sous tableau de gauche (si x < à l'élément du milieu)
B2) soit au sous tableau de droite (si x > à l'élément du milieu).
Et on respecte ces opérations jusqu'à le sous tableau de recherche soit vide.

1- Proposer une solution récursive pour cet algorithme de recherche.
2- Proposer une solution itérative pour cet algorithme.

Exercice 13 : Tri rapide

Le problème de tri consiste à trier (où à ranger) dans un ordre l'ordre croissant au sens large une suite d'éléments comparables (dotés d'une relation d'ordre total).
Il existe plusieurs algorithmes permettant de réaliser l'opération de tri.
Ces algorithmes sont classés en deux catégories :

- Algorithmes élémentaires : tri par sélection, tri par insertion, etc. (complexité n^2)
- Algorithmes évolués : tri par tas, tri rapide, etc. (complexité $n\log_2(n)$)

Le tri rapide (quick sort) est fondé sur la méthode de conception diviser pour régner en utilisant les deux étapes suivantes :
A- placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite. Cette opération s'appelle le partitionnement.
B- Pour chacun des sous-tableaux (sous tableau gauche et sous tableau droite), on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.

Écrivez un algorithme qui permet de remplir un tableau T par N réels, de le trier par le tri rapide et d'afficher le résultat.

Pour simplifier ce problème déclarez le 3 méthodes suivantes :

```
Procédure Permute(T: TabReel, i: entire, j:entier)
```

Cette méthode permet la permutation de deux éléments d'un tableau.

```
Fonction partition(T: TabReel, Gauche: entire, Droite:entier) :entier
```

Cette fonction fait le partitionnement par rapport à un pivot et qui renvoi le nouvel indice du pivot choisie.

```
Procédure tri_rapide(T: TabReel, Gauche: entire, Droite:entier)
```

Cette procédure utilise méthodes précédentes.