UNIVERSITE IBN TOFAIL
FACULTE DES SCIENCES
DEPARTEMENT D'INFORMATIQUE
-KENITRA-

TD1

Exercice1 :

Déterminez les informations suivantes pour les procédures ci-dessous :

1- Calculez le nombre d'appels à **faire**() pour n=100 ;
2- Déterminer le nombre d'appels de **faire**() en fonction de **n** ;
3- Déterminer la complexité temporelle asymptotique en notation **O(f(n))** en supposant que la complexité temporelle asymptotique de **faire()** est O(1).

```
Procedure proc1(Entier n)
{
   POUR a ALLANT_DE 0 A n-1
        faire();
   FIN_POUR

   POUR b ALLANT_DE 0 A n-1
        faire();
   FIN_POUR
}
```

```
Procedure proc2(Entier n)
{
   POUR a ALLANT_DE 0 A n-1
        POUR b ALLANT_DE 0 A n-1
            faire();
        FIN_POUR
   FIN_POUR

}
```

```
Procedure proc3(Entier n)
{
    a=0;
    TANT_QUE (a*a<=n) FAIRE
       faire();
       a<-a+1;
     FIN_TANT_QUE
 }
```

```
Procedure proc4(Entier n)
{
   POUR a ALLANT_DE 1 A 100
        POUR b ALLANT_DE 1 A n*n-1
            POUR c ALLANT_DE 1 A n-1
                faire();
            FIN_POUR
        FIN_POUR
   FIN_POUR
}
```

```
Procedure proc5(Entier n)
{
   POUR a ALLANT_DE 1 A n
        POUR b ALLANT_DE 1 A a
            faire();
        FIN_POUR
   FIN_POUR
}
```

```
Procedure proc6(Entier n)
{
   TANT_QUE (n>=1) FAIRE
       faire();
       n <- n/2;
   FIN_TANT_QUE
}
```

Exercice2 :

Déterminez les informations suivantes pour les méthodes ci-dessous :

1- Calculer le nombre d'appels à **faire()** pour n=3 ;
2- Déterminer le nombre d'appels à **faire()** en fonction de n ;
3- Déterminer la complexité temporelle asymptotique (notation O) en supposant que la complexité temporelle asymptotique de **faire()** est O(1).

```
FONCTION func1 (Eniter n)
{
    faire();

    SI (n <= 1 ) ALORS
        RENVOYER n;
    SINON
        RENVOYER n + func1(n-1);
    FIN_SI
}
```

```
Procedure proc2 (Eniter n)
{
    faire();

    SI (n > 1 ) ALORS
       proc2(n-1);
       proc2(n-1);
    FIN_SI
}
```

Exercice 3:

La suite de Fibonacci est une suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent.

Notée (F_n), elle est définie par $F_0 = 0$, $F_1 = 1$ et $F_n = F_{n-1} + F_{n-2}$ **pour** $n \geq 2$ **.**

1- Écrivez La fonction ***fibRecursiv(Entier n)*** qui renvoie le nième nombre de Fibonacci de manière récursive ;
2- Écrivez La fonction ***fibIterativ(Entier n)*** qui renvoie le nième nombre de Fibonacci de manière Itérative ;
3- En calculant la complexité de chacune, Déterminez le comportement d'exécution des deux fonctions ***fibRecursiv(Entier n)*** et ***fibIterativ(Entier n)***?

Exercice 4: Tri par insertion

Le tri par insertion est le tri du joueur de cartes. On fait comme si les éléments à trier étaient donnés un par un. Le premier élément constituant, à lui tout seul, une liste triée de longueur 1. On range ensuite le second élément pour constituer une liste triée de longueur 2, puis on range le troisième élément pour avoir une liste triée de longueur 3 et ainsi de suite...

Le principe du tri par insertion est donc d'insérer à la nième itération le nième élément à la bonne place.

1- Ecrire l'algorithme ***Tri_insertion*** permettant de faire un tri par insertion sur un tableau ;
2- Déterminer la complexité de l'algorithme ***Tri_insertion;***

Exercice 5 : Tri Rapide

Le principe de ce tri est d'ordonner un vecteur T de taille n en cherchant dans celui-ci une clé pivot autour de laquelle réorganiser ses éléments. L'idée est de choisir un élément au hasard dans le tableau et de s'en servir comme pivot. On va placer d'un côté toutes les valeurs inférieures au pivot, et de l'autre les valeurs supérieures à celui-ci. Puis, il faudra réitérer cela pour chaque partie du tableau, jusqu'à que les partitions soient composées d'un seul élément.

A chaque fois, on permute les éléments du tableau de façon à ce que pour un indice j particulier tous les éléments dont la valeur est inférieure à pivot se trouvent dans T[0]...T[j] et tous ceux dont la valeur est supérieure se trouvent dans T[j+1]...T[n]. On place ensuite le pivot à la position j.

1- Ecrire l'algorithme ***Tri_Rapide*** permettant de faire un tri rapide sur un tableau ;
2- Déterminer la complexité de l'algorithme **Tri*_Rapide ;***
3- Faites une comparaison de l'efficacité des algorithmes de tri étudiés (dans le TD 1 et dans le Cous).