**FICHE TD : Analyse d’algorithme et complexité**

**Exercice 01**

Soit la fonction suivante qui fait la recherche par dichotomie d’un entier dans un tableau trié.

Fonction  recherche Dichotomique(**t[]: entier**,  x : entier): entier;

Début   
       m, g, d, cmp: entier;  
  
      g = 0; d = N-1;  
      tant que (g <= d) faire  
          m = (g+d)/2;  
         **si**(t[m] == x)  
              **retourner** m  
          **sinon** **si**(t[m] < x) alors  
              d = m-1  
          **sinon**  
              g = m+1;  
Fin

Analyser cette fonction (Terminaison, correction et complexité).

**Exercice 02**

|  |  |
| --- | --- |
| Soit l’algorithme suivant :  Algorithme exer ;  Entrée : un entier n et un tableau T alloué ;  Sortie : un tableau T et un entier i ;  n2=n ;  i=0 ;  Tant que ( n2>0) faire  T[i]= n2 mod2 ;  n2=n2/2 ;  i=i+1 ;  Fin tant que ;  Retourner i ;  Fin. | **Question :**  Analyser cet algorithme. |

**Exercice 03**

Calculer la complexité pour la fonction factorielle récursive suivante :

int fac(int n)

{

if(n>1) return n\*fact(n-1) ;

else return (1);

}

**Solution**

**Exercice 01 :**

Analyser cette fonction (Terminaison, correction et complexité).

1. Terminaison

Il existe un convergent g qui prouve que l’algorithme va se terminer à un moment donné.

1. Correction :

Il existe une assertion m = (g+d)/2 qui est vrai avant le traitement et reste vraie à chaque passage de la boucle et reste vraie à la sortie de traitement.

1. Complexité

Fonction  rechercheDichotomique(**t[]: entier**,  x : entier): entier;

Début   
       m, g, d, cmp: entier;  
  
      g = 0; 1op  d = N-1 2 op;   
      tant que (g <= d (n fois (n-1)-0+1)) faire  
          m = (g+d)/2; 3 op   
         **si**(t[m] == x) 1 op   
              **retourner** m 1 op   
          **sinon** **si**(t[m] < x 1 op) alors  
              d = m-1 2 op  
          **sinon**  
              g = m+1; 2 op  
Fin

**Cout Global :**

**Partie initialisation : 3 op**

**Partie traitement plus résultat :10 n**

**Cout global = 10n+3 donc la complexité est de O(n) donc cette fonction est rapide.**

**Exercice 02**

|  |  |
| --- | --- |
| Soit l’algorithme suivant :  Algorithme exer ;  Entrée : un entier n et un tableau T alloué ;  Sortie : un tableau T et un entier i ;  n2=n ; 1op  i=0 ;1 op  Tant que ( n2>0 n fois) faire  T[i]= n2 mod2 ; 2 op  n2=n2/2 ; 2 op  i=i+1 ; 2 op  Fin tant que ;  Retourner i ; 1 op  Fin. | **Convergent = n2**  **Invariant =T[i]=** n2 mod2.  Calcul de la complexité  Partie initialisation : 2 op  Partie traitement : 6 n  Partie résultat : 1 op  Cout global : 6n+3  Complexité= **est de O(n) donc cet algorithme est rapide.** |

**Exercice 03**

Calculer la complexité pour la fonction factorielle récursive suivante :

int fac(int n)

{

if(n>1) return n\*fact(n-1) ;

else return (1);

}

**Cette fonction est récursive, on ne peut pas utiliser les formules de calcul de la complexité comme dans le cas d’un algorithme itératif :**

**On prend des exemples :**

**Pour n=1 le nombre d’appel =1**

**Pour n= 2 le nombre d’appel récursif =2**

**Pour n= 3 le nombre d’appel récursif =2**

**. . ……**

**Pour n=n le nombre d’appel =n la complexité augmente en fonction de n donc elle est égale à de O(n) donc cette fonction est rapide.**