Complexité de séquences itératives - Déterminer la complexité de chaque séquence d'instructions

Séquence -1 :

```
For i = 1 to N do
        Opération ;
Endfor
```

Séquence -2 :

```
For i = 1 to N do
        For j = 1 to i do
                Opération ;
        Endfor
Endfor
```

Séquence -3 :

```
While ( i < N) Do
        i = 2*i;
        Opération ;
Endwhile
```

Séquence -4 :

```
For i = 1 To N Do
        J=1;
        While (J < N ) Do
                J = 2 * J;
                Opération;
        Endwhile
Endfor
```

Séquence -5 :

```
i = 1;
While ( i < N ) Do
        i = 2*i;
        For j = 1 to i Do
                Opération;
        Endfor
Endwhile
```

Séquence -6 :

```
i = 1;
For j=1 To n do
      i = 2*i;
Endfor
      For j= 1 to i do
            Opération;
      Endfor
```

Séquence -7 :

```
i=1
For k = 1 To n  do
        i = 2*i;
        For L = 1 to i do
                For m = 1 to k do
                        Opération ;
                Endfor
        Endfor
Endfor
```

Séquence -8 :

```
For k = 1 to n do
        i = 1;
        For j = 1 to k do
                i = 2*i;
        Endfor
        For j = 1 to i do
                Opération;
        Endfor
Endfor
```

Séquence -9 :

```
For j=2 to n do
        clé = A[ j ]
        i= j-1
        while (i > 0 and A[i] > clé) do
        A[i+1]=A[i]
        i=i-1
        End_while
A[i+1]=clé
Endfor
```

Evaluer le coût unitaire et le coût total de chaque instruction et déduire la complexité de cette séquence.

Séquence -10 :

```
while(low <= high)
{
  mid = (low + high) / 2;
  if (target < list[mid])
    high = mid - 1;
  else if (target > list[mid])
    low = mid + 1;
  else break;
}
```

N.B. : On néglige les coûts de toutes les opérations non soulignées.

Exercice : Algorithme de recherche dichotomique

```
Fonction RechDicho(Tab :Tableau, borneinf :entier, bornesup :entier,
    elem :entier) : bool
        Si (borneinf<=bornesup) alors
            mil = (borneinf+bornesup) DIV 2 ;
            Si (Tab[mil]=elem) Alors
            retourner (vrai)
            Sinon
            Si (Tab[mil]>elem) Alors
            Retourner (RechDicho(Tab, borneinf, mil-1, elem))
        Sinon
            Retourner(RechDicho(Tab, mil+1, bornesup, elem))
            Fin Si
            Fin Si
            Sinon
            Retourner (Faux)
        FinSi
```

Analyser la complexité de cet algorithme.

$T(n) = T(n/2) + cte$

$T(n) = O(\log_2 n)$

Exercice : algorithme tour de Hanoi

```
Procédure Hanoi (n, départ, intermédiaire, destination)
    Si n > 0 Alors
        Hanoi (n-1, départ, destination, intermédiaire)
        déplacer un disque de départ vers destination
        Hanoi (n-1, intermédiaire, départ, destination)
    Fin Si
Fin
```

Faire la trace d'exécution et analyser la complexité de cet algorithme.

Faire la trace d'exécution pour n=3 (appel à Hanoi (3,1,2,3))

On compte le nombre de déplacements de disques effectués par l'algorithme Hanoi invoqué sur n disques.

On trouve :

- $T(n) = T(n-1) + 1 + T(n-1)$
- $T(n) = 2T(n-1) + 1$
- $T(n) = 2^n - 1$

Complexité exponentielle

Un petit résumé:

Growth Rate	Name	Code Example	description
1	Constant	a= b + 1;	statement (one line of code

log(n)	Logarithmic	while(n>1)
		{
		n=n/2;
		}
			Divide in half (binary search)

n	Linear	for(c=0; c<n; c++)
		{
		a+=1;
		}
			Loop

n*log(n)	Linearithmic	Mergesort, Quicksort, …	Effective sorting algorithms

n^2	Quadratic	for(c=0; c<n; c++)
		{
		for(i=0; i<n; i++)
		{
		a+=1;
		}
		}
			Double loop

n^3	Cubic	for(c=0; c<n; c++)
		{
		for(i=0; i<n; i++){
		for(x=0; x<n; x++){
		a+=1;
		}
		}
		}
			Triple loop

2^n	Exponential	Trying to braeak a password generating all possible combinations	Exhaustive search