6. COMPLEJIDAD COMPUTACIONAL

Esta sección quiere mostrar la importancia de estimar un algoritmo en términos de los recursos físicos que va a utilizar antes que el algoritmo sea codificado en un lenguaje específico, y por tanto antes de considerar una maquina para su ejecución.

El rendimiento o complejidad, está ligado únicamente al algoritmo. En ningún momento con la velocidad del computador o con las facilidades de eficiencia que presenta un lenguaje de programación, ni con el estilo hábil del programador.

Para calcular este rendimiento se tiene en cuenta dos pasos principales.

1. Caracterizar las operaciones básicas del algoritmo. Operaciones básicas son las que constituyen el trabajo realizado para resolver el problema. Si el algoritmo ejecuta una estructura repetitiva (ciclo), lo básico en el proceso no es la actualización del índice para la satisfacción de la condición.

2. Debe considerarse sobre quien va a recaer la operación. Se considera entonces los datos. De los datos debe considerarse el tamaño y la configuración.

El tamaño, esto es el número de datos; sea D_n={datos de tamaño n}, para un algoritmo cualquiera.

Un problema simple. Establecer si un elemento Q pertenece a un vector (lista), depende del número de elementos del vector. Para un D_n conocido existirán múltiples instancias posibles.

La configuración I, I en D_n, encontrar un Q en una configuración específica sería bien característico para cada una de ellas. Q puede ser el primero de I, puede estar en un lugar intermedio, o simplemente puede no estar. Según sea el caso, el tiempo requerido para solucionar el problema es distinto.

Análisis del comportamiento promedio.

Sea W un vector de n elementos, si un valor Q esta en el vector, el siguiente algoritmo reporta su posición, sino esta, reporta 0.

```
POSI(W,Q)
        k = 1
        MQ (k<=n, W(k)><Q)
                k=k+1
        FMQ
        SI k>= n TH k=0
                ESCRIBIR(k)
FINPOSI()
```

Eventualmente, el ciclo se repite n veces. Dentro del ciclo, la operación básica es la comparación W(k) ><Q. La actualización de k, así como la impresión pueden estar ligadas con el lenguaje de programación.

Para una ubicación cualquiera de Q, que se llamará i, tendremos un tiempo asociado $t(I_i)=i$, siendo I_i una configuración de las varias posibles, en las que Q esta en la posición i.

Sea p_i=probabilidad de que Q este en I.

De este modo, $p(I_i)=p_i/n$ para cualquier lugar i, $1<=i<=n$.
Para establecer que Q no aparece en W se requieren n comparaciones. Todo esto conduce a:

$tcp = [SUM_{i=1}^{n} p(I_i)t(I_i)] + (1-p_i)n$
$tcp = [SUM_{i=1}^{n} (p_i/n)i] + (1-p_i)n$
$tcp = (p_i/n)[SUM_{i=1}^{n} i] + (1-p_i)n$
$tcp = (p_i/n)[n(n+1)/2] + (1-p_i)n$
$tcp = p_i(n+1)/2 + (1-p_i)n$.

El peor caso es en que X está en la ultima posición, o no aparece.
$tcp = max\{t(li)\} = n$, para n=1,2,...,n

El análisis computacional de un problema requiere:

a) una medida de esfuerzo (operaciones básicas)
b) una medida del tamaño (sobre quien va a recaer el esfuerzo)

Se pueden clasificar los problemas que se someten al computador para ser solucionados desde el punto de vista de tamaño y desde el punto de vista de esfuerzo.

Clasificación según tamaño:

a. búsqueda, hallar un valor X en una entrada, tal que X satisfaga la condición Y;
b. estructuración, transformar la entrada para que satisfaga la propiedad Y;
c. construcción, construir X para que satisfaga la propiedad Y;
d. optimización, encontrar XC que satisfaga mejor la propiedad Y;
e. decisión, decidir si la entrada satisface la propiedad Y;
f. adaptativos, mantener la propiedad Y a través del tiempo.

Clasificación según esfuerzo:

a. conceptualmente difíciles, no hay algoritmo asociado por que no se comprende el problema cabalmente;
b. analíticamente difíciles, existe algoritmo para resolver el problema pero no se sabe analizar cuanto tiempo tomará solucionar cada instancia;

c. computacionalmente difíciles, hay algoritmo y se sabe analizar, pero el análisis indica que instancia relativamente pequeñas tomaran millones de años para ser resueltas;
d. computacionalmente imposible, se ha demostrado que no existe algoritmo alguno para hallar la solución del problema.

Ejemplos de algoritmos recursivos e iterativos:

1. Factorial

Por definición n!=1 si n=0
n!=n*(n-1) si n>0

Algoritmo recursivo

```
FACREC (n)
        SI (n=0) TH FACREC = 1
                SN FACREC = n*FACREC(n-1)
        FSI
FIN()
```

Algoritmo iterativo

```
FACITE (n)
        FACITE = 1
        SI (n>1) TH PARA i= 1, n
                FACITE = FACITE * i
                FPARA
        FSI
FIN()
```

2. Fibonacci

Por definición F(n) = n si n=0,1
F(n) = F(n-1) + F(n-2) si n>1

Algoritmo recursivo

```
FIBREC (n)
        SI (n=0 o n=1) TH FIBREC (n) = n
                SN  FIBREC (n) = FIBREC (n-1) + FIBREC (n-2)
        FSI
FIN()
```

Algoritmo iterativo

```
FIBITE (fib,n)
   SI (n=0  0  n=1)  TH  fib = n
                SN  fiba = 0
                   fibb = 1
                   i=2
                   MQ  (I<=n)
                      fib = fibb + fiba
                      fiba = fibb
                      fibb = fib
                       i=i+1
                   FMQ
   FSI
FIN()
```

3. Capital

M=monto inicial, N=# períodos, x= % interés por período

$C_n = M$ si $n = 0$
$= (1+x)*C_{n-1}$ si $n > 0$

Algoritmo recursivo

```
CAPREC(M,N,x)
// calcula el capital que se tendra al invertir un monto inicial M
// durante N años con una tasa anual de x%
        SI N=0 TH CAPREC(N) = M
                SN CAPREC(N) = (1+x)*CAPREC(M,N-1,x)
        FSI
FIN()
```

Algoritmo iterativo

```
CAPITE(M,N,x)
        C=M, i=1
        MQ i <= N
                C=(1+x)*C
                i<- i+1
        FMQ
        Escribir C
FIN()
```

4. Torres Hanoi

```
HANOI(N,ORIG,DEST,aux)
// el número de movimientos esta dado por la formula NM = 2N-1
        SI N=1 TH ORIG = DEST
                SN  HANOI(N-1,ORIG,AUX,DEST)
        FSI
FIN
```