Universidad Nacional Mayor de San Mar os
Facultad de Ingeniería de Sistemas

# Algoritmica III

## Guía 2

## Inducción y recursión

Mg Augusto Cortez Vásquez

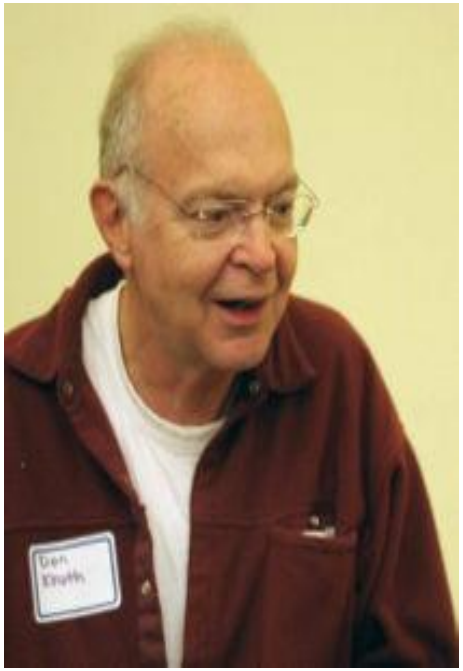
# Objetivos

- Conocer las técnicas de induccion matematica
- Conocer las técnicas de recursividad y recurrencias
- Conocer la tecnica Divide y Venceras

*Tres clases hay de ignorancia: no saber lo que debiera saberse, saber mal lo que se sabe, y saber lo que no debiera saberse.*

Faccois de la Rochefoucald

*(1613-1680) Escritor francés*



**Donald Ervin Knuth** Nacido el 10 de enero de 1938, es uno de los más renombrados científicos de la computación, profesor emérito de la Universidad de Stanford. Es conocido como autor de múltiples volúmenes: El arte de la programación de computadoras, considerado como una referencia en el área de ciencias de la computación, prácticamente fue el creador del análisis de algoritmos y contribuyo significativamente a varias ramas de teoría de las ciencias de la computación

## INDUCCION MATEMATICA

Consideremos que la proposición P(n) definida en un dominio. Se desea demostrar que P(n) es verdadera para todo n en el dominio. Si verificamos que:

1 P(k) es verdadera, para k un entero (positivo, negativo o cero) fijo.

**2** Si P(h) es verdadera para todo h > k llamada

(**hipótesis inductiva)** implica que P(h+1) es verdadera.

3 P(n) es verdadera para todo n en su dominio.

Quiere decir que si se cumple los pasos 1 y 2 entonces por el principio de inducción afirmamos que se cumple el paso 3, es decir P(n) es verdadera para todo n.

## *Ejemplo 1*

$$\textbf{Si} = 1 + 2 + 3 + \ldots n = \frac{n \cdot (n+1)}{2}$$

sea el enunciado $P(n) = 1 + 2 + 3 + \ldots n = \frac{n \cdot (n+1)}{2}$

**1** para n = 1 $\quad P(1) = 1 = \frac{1 \cdot 2}{2} = 1$

**2** supongamos que para n= h > 1, P(h) es verdadera **hip. inductiva**

es decir $P(h) = 1 + 2 + 3 + \ldots h = \frac{h.(h+1)}{2}$

tenemos que

**3** $P(h+1) = 1 + 2 + 3 + \ldots h + h+1$

$= (1 + 2 + 3 + \ldots h) + h+1$

$= \frac{h.(h+1)+h+1}{2}$

$= \frac{(h+1) \cdot (h+2)}{2}$

por el principio de inducción matemática se sigue que P(n) es verdadera para todo $n \geq 1$.

```
FUNCION CUADRADO(N)
Inicio
        B ——➢ 0
        S ——➢ 0
        MIENTRAS (B < N)
                S ——➢ S + N
                B ——➢ B + 1
        FIN MIENTRAS
        RETORNAR(S)
Fin
```


El algoritmo recibe como entrada N y devuelve S= $N^2$
Sea la proposición P(n) : $S_n = N * B_n$

donde $S_n$ y $B_n$ son los valores de S e B después de haber pasado por el ciclo MIENTRAS i veces.

para **n = 0**

$P(0) : S_0 = 0 = N \times B_0 = N * 0$

porque S = B no han pasado aún por el ciclo MIENTRAS

Para **n = h** > **0**

supongamos que P(h) : es verdadero para h> 0

$S_h = N \times B_h$ **hipotesis inductiva**

para **n = h+1** se tiene:

$S_{h+1} = S_h + N$ .....(a)

$B_{h+1} = B_h + 1$ .....(b)

De aquí se tiene que, reemplazando h.i. en (a)

$S_{h+1} = N * B_h + N = N * (B_h + 1)$

$= N \times B_{h+1}$ por (b)

luego, se cumple $S_{h+1} = N * B_{h+1}$

por lo cual P(n) es verdadero para todo n

# Recursión

La recursión es un método que, directa o indirectamente, se hace una llamada así mismo.

Esto puede  parecer un circulo vicioso: ¿cómo un método F puede resolver un problema llamándose a si mismo?

La clave esta en que el método F se llama así mismo pero en instancias diferentes, mas simples, en algún sentido adecuado

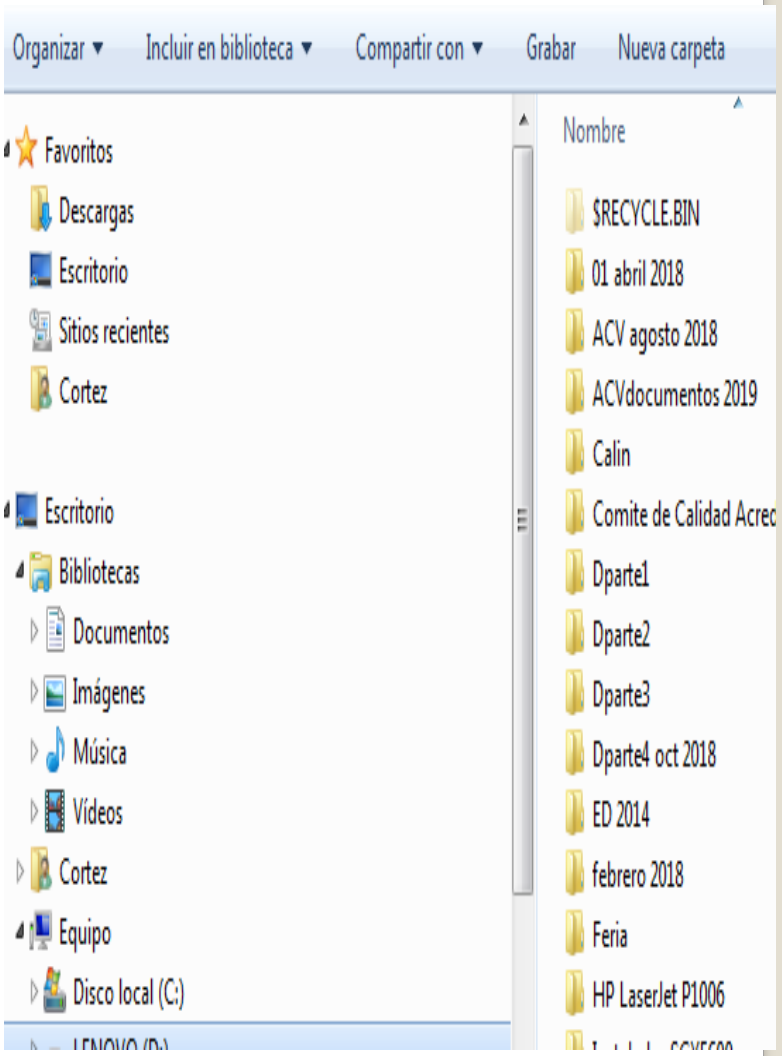
La recursión es una técnica que se utiliza en la vida cotidiana

La recursión es un concepto fundamental en matemáticas e informática. La definición mas sencilla es que una función se llama recursiva si se llama a si mismo.  Una función recursiva es aquella función que se define en términos de si mismo. Una programa o función recursiva requiere una condición de terminación que autorice al programa o función a dejar de llamarse a si mismo.

**Los lenguajes de programación mas conocidos permiten la implantación de algoritmos recursivos. Esta es una metodología que se emplea para la solución de problemas de computación y en muchos casos facilita resolverlos.**

## Ejemplo 1

Los ficheros en un computador se almacenan generalmente en directorios. Los usuarios pueden crear directorios, que a su vez almacenan mas ficheros y directorios. Suponga que deseamos examinar cada fichero de un directorio D, incluyendo todos los ficheros de sus subdirectorios ( y sus subdirectorios, y así sucesivamente). Esto se puede hacer examinando recursivamente los ficheros de cada subdirectorio junto con todos los ficheros en el directorio D.



# Ejemplo 2

Definición recursiva de tren

TREN : Locomotora + Vagones

Vagones : Vagón + Vagones / Vagón

# Ejemplo 3

**Sea S(N) : suma de los N primeros números. S(N) puede definirse :**

**Forma recursiva**

S(1) = 1
S(N) = S(N-1) +N

**Forma explicita**

S(N) = N*(N+1) /2

```
Entero Suma()
Inicio
        Leer N
        Escribir S(N)
Fin
```

Algoritmo iterativo

```
Entero S(Entero N)
Inicio
    S=0
    Para i desde 1 hasta N
                S = S +i
    FinPara
    Retornar ( S)
 Fin
```

**Algoritmo recursivo**

```
Entero S(Entero N)
Inicio
    Si N = 1
        Retornar 1
    Sino
        Retornar S(N-1)+N
    Fin si
Fin
```

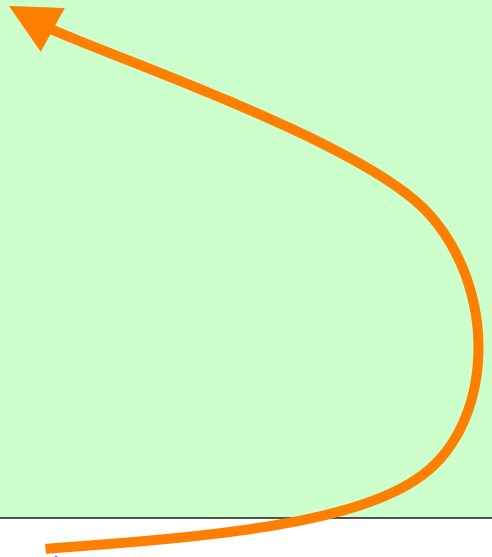
La función Suma lee N y luego escribe S(N) que es la suma de los N primeros números

Debemos tener en cuenta que la recursión no siempre es apropiada. En ocasiones las llamadas recursivas consumen tiempo y limitan el valor de N para el cual se puede ejecutar el programa. No es conveniente, por ejemplo usar la recursión para sustituir un simple bucle.

```
Funcion Potencia(a:entero; n:natural)    dev (p:entero)
    Caso        n=0                 retornar 1
    Caso        n >0                return a * Potencia(a,n-1)
Fin



Accion Principal()
Inicio
    Leer a,n
    Escribir Potencia(a,n)
Fin
```



## *Ejemplo 4*

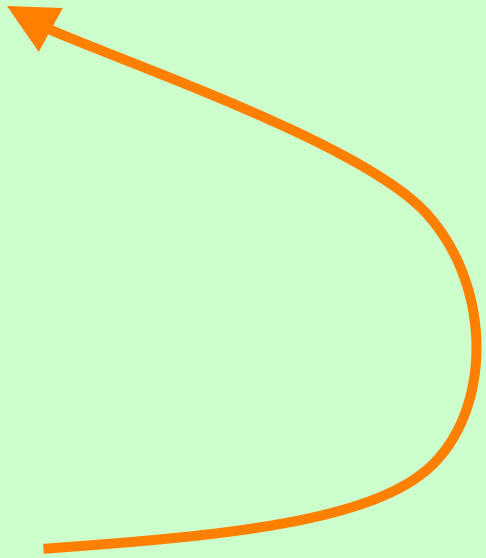
la secuencia de fibonacci son 1, 1, 2, 3, 5, 8, ... los cuales se determinan por la función:

Fib(n) = Fib(n-1) + Fib(n-2) si n>1,
Fib(n)=1 si n <= 1

```
fib(int n)
{
        if(n==0 || n==1) return 1;  /* condición básica  */
        else
                return (fib(n-1)+fib(n-2));   /* doble llamada
*/
```

```
Funcion Fibo(a:entero; n:natural)    dev (p:entero)
    Caso        n=0 o n=1       retornar 1
    Caso        n  > 1                fib(n-1)+fib(n-2)
Fin


Accion Principal()
Inicio
    Leer n
    Escribir Fibo(n)
Fin
Fin
```



# Relación de recurrencia

La relación de recurrencia $a_{n+1} = 3\ a_n$ para $n \geq 0$ no define una única progresión geométrica, pues la secuencia:

7,21,63,189, también satisface la relación. Para distinguir una sucesión particular descrita necesitamos conocer uno del término de la sucesión. Por tanto

$a_{n+1} = 3\ a_n$ $n \geq 0$ donde $a_0 = 5$

define 5, 15, 45, 135….

Mientras que
$a_{n+1} = 3\ a_n$ $n \geq 0$ donde $a_0 = 3$

define 3, 19, 27, 81….

## Relación de recurrencia de primer orden

La ecuación $a_{n+1} = 3\ a_n$ es una recurrencia, ya que el valor de $a_{n+1}$depende de $a_n$ . Como cada elemento depende solo de su predecesor inmediato decimos que es de primer orden

## Relación de recurrencia homogénea lineal de primer orden

En la ecuación $a_{n+1} = K\ a_n$ , $a_n$ depende solo de su predecesor inmediato

Los valores $a_0$ y $a_1$ que se dan además de la relación de recurrencia se denominan condiciones de frontera. En la expresión $a_0=A$ , donde A es una constante, también se conoce como condición inicial.
La relación $a_{n+1} = K\ a_n$ se dice lineal porque, puede expresarse de la forma
$a_{n+1} - K\ a_n = 0$
Cada término con subíndice aparece elevado a la primera potencia.

# Recurrencia homogénea

Una recurrencia es homogénea con coeficientes constantes, si tiene la forma

$$a_0 t_n + a_1 t_{n-1} + a_2 t_{n-2} + \ldots a_k t_{n-k} = 0$$

**para todo $a_i$ constante**

en donde los $t_i$ son los valores que estamos buscando . La combinación lineal de los $t_{n-i}$ es igual a 0.

## *Ejemplo 5*

$a_n = a_{n-1} + a_{n-2}$

podemos reescribir $a_n - a_{-n-1} - a_{-n-2} = 0$

Rec. homogenea

la recurrencia corresponde a la serie de fibonacci.

Donde K = 2, $a_0 = 1$ $a_1 = a_2 = -1$

**Podemos remplazar** $t_n$ por $x^n$
**donde x es una constante desconocida por el momento**

$$\mathbf{a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} +\ldots a_k x^n = 0}$$

La ecuación se satisface si x = 0 siendo la solución trivial
en caso contrario la ecuación se satisface si existe k tal que

$$\mathbf{a_0 x^k + a_1 x^{k-1} + a_2 x^{k-2} +\ldots a_k = 0}$$ **ecuación característica**

$$a_0 x^k + a_1 x^{k-1} + a_2 x^{k-2} + \ldots a_k = 0$$

**ecuación característica**

$$P(x) : a_0 x^k + a_1 x^{k-1} + a_2 x^{k-2} + \ldots a_k$$

**polinômio característico**

Las raíces de esta ecuación están en alguno de los tres casos siguientes:

$r_1$, $r_2$ son números reales distintos
$r_1$, $r_2$ son números complejos conjugados
$r_1$, $r_2$ son números reales iguales

en todos los casos $r_1$, $r_2$ son las raíces características

## *Ejemplo 6*

$a_n + a_{n-1} - 6a_{n-2} = 0$ donde $n \geq 2$ $a_0 = 1$ $a_1 = 2$

**$P(x) : x^2 + x - 6$** **polinômio característico**

Cuya solución esta dada por (x+3)(x-2)

Con raíces $r_1 = 2$ y $r_2 = -3$

La solución general es de la forma

$a_n = c_1 r_1^{\,n} + c_2 r_2^{\,n} = c_1 2^{\,n} + c_2 (-3)^{\,n}$

cuando n = 0

$a_0 = 1 = c_1 2^0 + c_2 (-3)^0$

$= c_1 + c_2$ (a)

cuando n = 1

$a_1 = 2 = c_1 2^1 + c_2 (-3)^1 = 2c_1 - 3 c_2$ (b)

resolviendo (a) y (b)

$c_1 = 1$ y $c_2 = 0$

por tanto $a_n = 2^n$ para $n \geq 0$

**Es la única solución de la recurrencia dada.**

# Recurrencia no homogénea

Una recurrencia es no homogénea cuando la combinación lineal no es igual a cero, es decir, no es cierto que toda la combinación lineal de las soluciones sea una solución

$$a_0 t_n + a_1 t_{n-1} + a_2 t_{n-2} + \ldots a_k t_{n-k} = b^n P(n)$$

**para todo b, $a_i$ constante**

**P(n) es um polinomio en n**

## *Ejemplo 7*

$a_n - 2\,a_{n-1} = 3^n$ b=3 P(n) = 1 (a)

multiplicando (a) por 3

$3a_n - 6\,a_{n-1} = 3^{n+1}$ (b)

Sustituyendo n por n-1 en (b)

$3a_{n-1} - 6\,a_{n-2} = 3^n$ (c)

Restando (a) y (c)

$a_n - 5\,a_{n-1} + 6\,a_{n-2} = 0$ (d)

(d) es una ecuación homogénea y se resuelve con la técnica antes expuesta.

## Solución de una recurrencia mediante remplazos sucesivos

***Ejemplo 8***

Consideremos la siguiente serie  S : 2, 5, 8, 11, 14...

Se quiere hallar la suma de los N primeros números de la serie

S puede definirse de la siguiente forma

$a_n = a_{n-1} + 3$ para $n > 1$ $\quad a_1 = 2$

Asi

| N | 1 | 2 | 3 | 4 | 5 | | |
|---|---|---|---|---|---|---|---|
| $a_n$ | 2 | 5 | 8 | 11 | 14 | | |

```
Accion Principal()
Inicio
        Leer N
        Escribir SUMA(N)

Fin
```

```
Accion SUMA(K)
Inicio
        S=2
        V=5
        Para i desde 1 hasta K-1
                S=S+V
                V=V +3
        FinPara
        Retornar S
Fin
```

## Solución recursiva

```
Accion Principal()
Inicio
        Leer N
        S=0
        Para i desde 1 hasta N
                S = S + SUMA(i)
        FinPara
        Escribir S
Fin
```

```
Accion SUMA(K)
Inicio
        Si k=1
                Retornar 2
        Sino
                Retornar SUMA(K-1) + 3
FinSi
```

¿ De que orden es el algoritmo ?

$a_n = a_{n-1} + 3 \qquad a_1 = 2$ (a)

esta recurrencia puede resolverse mediante la técnica de polinomio característico o mediante remplazos sucesivos

si remplazamos **n** con **n-1** en (a) tenemos

$a_{n-1} = a_{n-2} + 3$ (b)

sustituimos (b) en (a)

tenemos $a_n = a_{n-2} + 3 + 3 = \mathbf{a_{n-2} + 3 * 2}$ (c)

nuevamente si remplazamos **n** con **n-1** en (a) tenemos

$a_{n-2} = a_{n-3} + 3$ (d)

sustituimos (d) en (c)

$a_n = a_{n-3} + 3 + 3*2 = \mathbf{a_{n-3} + 3 * 3}$

en general

$a_n = a_{n-k} + k * 3$

si hacemos K = n-1 tenemos

$a_n = a_1 + (n-1) * 3$

y como $a_1 = 2$

tenemos finalmente

$\mathbf{a_n = 2 + 3 * (n-1)}$

Así decimos que el algoritmo con ecuación de recurrencia es de orden **O(3n)**

# Divide y vencerás

Consiste en dividir un problema en dos o más subproblemas, cada uno de los cuales es similar al original pero más simple en tamaño.
Para resolver cada subproblema se tienen dos alternativas:
El subproblema es suficientemente pequeño para dar una solución
o
El subproblema todavía es de gran tamaño el cual se debe dividir aplicando recursivamente la técnica.
Los algoritmos divide y vencerás son algoritmos recursivos que constan de dos partes:

Dividir: se resuelve recursivamente problemas mas pequeños (excepto, por supuesto los casos bases)
Vencer : La solución al problema original se consigue a partir de las soluciones a los subproblemas.

# Algoritmo

**DivideyVencerás( limite del problema)**

```
Inicio
        Si la condición  define un caso
                         suficientemente simple entonces
                         Dar la solución del caso simple
        Sino
                 //Dividir el problema en subproblemas
                 DivideyVencerás( límite del subproblema 1)
                 ……….
                 DivideyVencerás( límite del subproblema n)
        Fin_si
Fin
```

***Ejemplo 9***

Multiplicación de enteros grandes

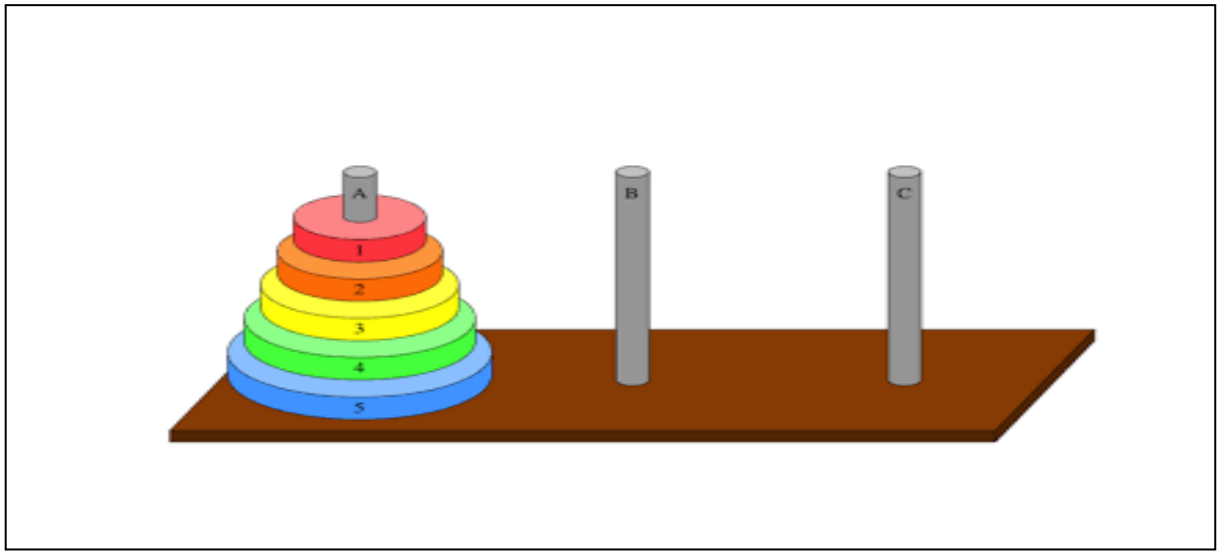
Sea X = 23435676 Y = 54871262

Podemos hacer $X = X_1X_2$ $Y = Y_1Y_2$

Entonces $X * Y = X_1*Y_1 * 10^8 + (X_1*Y_2 + X_2*Y_1) * 10^4 + Y_1*Y_2$

***Ejemplo 10***

Multiplicación de matrices




¿ Cual es la recurrencia?

$a_n$ = numero de veces que se utiliza (o ejecuta) la instrucción escribir en una llamada a Hanoi()

$$a_n = \begin{cases} 1 & n = 0 \\ 2a_{n-1} + 1 & \text{otro caso} \end{cases}$$

$$a_n - 2a_{n-1} - 1 = 0$$

```
int E(int n)
{
   if (n == 1)
      return 0;
   else
      return E(n/2) + 1;
}
```

$$T(n) = \begin{cases} 1, & n = 1 \\ 1 + T\left(\frac{n}{2}\right) & n > 1 \end{cases}$$

**T(n) es O ($\log_2$ n)**

T(n) = T(n/2) + 1
= (T(n/4) + 1) + 1 = T(n/4) + 2
= (T(n/8) + 1) +2 = T(n/8) + 3
...
= $T(n/2^i) + i$
...
= $T(n/2^{\log(n)}) + \log_2(n)$
= $T(1) + \log_2(n)$

Por tanto, T(n) es O(log n)

```
Accion Merge_Sort(S,i,j)
Inicio
Si i = j  retornar
          M=(i+j)/2
          Merge_Sort(S, i, m)
          Merge_Sort(S , m+1, j)
          Merge(S, i , m, j, C)
          Para k dese i hasta j
                    S_k = C_k
          FinPara
Fin
```

Tenemos que $T(n) = 2T(n/2) + n$ (a)

Sustituimos n por n/2

$2T(n/2) = 2\,(2T(n/4) + n) = 4\,T(n/4) + n$

Remplazamos en (a) $T(n) = 4T(n/4) + 2n$ (b)

Sustituimos n por n/4

$4T(n/4) = 4\,(2T(n/8) + n) = 8\,T(n/8) + n$

Remplazamos en (b)

$T(n) = 8T(n/8) + 3n$

Sucesivamente, tenemos que

$T(n) = 2^k T(n/2^k) + kn$ (c)

Hacemos $k = \log n$ luego $n = 2^k$

En (c)

$T(n) = nT(1) + n \log n = n + n\log n$

Por tanto el algoritmo es de orden $O(n\log n)$

# Ejercicios propuestos

1 **Proporcione dos ejemplos de proposiciones matemáticas. Demuéstrelas por el método de inducción matemática**

2 **Proporcione dos ejemplos de aplicación de recursión en la vida cotidiana**

3 Probar por el método de inducción que:

**FUNCION COMPARA(N, B; Y)**

```
INICIO
        Y = N
        A = B
        MIENTRAS A > 0
                Y = Y + N
                A = A - 1
        FIN MIENTRAS
RETORNAR
```

donde $A_n$ y $Y_n$ son los valores de A y Y después de haber pasado por el ciclo MIENTRAS n >=0 veces.

El algoritmo recibe como entrada N y B, y devuelve:
$Y = N + B^2$

**4 Defina los siguientes conjuntos en forma recursiva**

a) **El conjunto de todas las proposiciones del calculo proposicional.**

b) **El conjunto de todas las expresiones aritméticas**

c) **El conjunto de los números enteros positivos múltiplos de K**

**5 Construya un algoritmo recursivo para**

**a) Sumar los elementos de una lista enlazada**

**b) Invertir el orden de un vector**

**c) Comparar si dos listas son iguales o no**

**d) Hallar la suma de los elementos de un vector en forma recursiva dicotómica**

**6** Dada las siguientes sucesiones. Construya la recurrencia y resuelva por el método de polinomio característico y por el método de reemplazos sucesivos

a) 3, 8, 18, ....

b) 3, 5, 11, 21, 43...