UNIVERSIDAD NACIONAL

AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Algoritmos y Estructuras de Datos I (2016)

Ing. Adrian Ulises Mercado Martínez

Análisis de Algoritmos Recursivos

Hernández Rojas Mara Alexandra

Número de cuenta 317206876

Ciudad de México a 07 de junio de 2020

Tarea Análisis Algoritmos

a)X(n) = X(n-1) + 5 para n>1 x(1) = 0

[X(n-2)+5]+5

[X(n-3)+5]+5+5

Patron X(n-i)+5i

sustituyendo i=n-1, X(1)=0

(n-(n-1))+5(n-1)

(1) + 5n-5

5n-4

X(n) es de orden n

b)X(n) = 3X(n-1) para n>1, x(1)=4

3[3X(n-2)]

3**2[3X(n-3)]

Patron 3**i[X(n-i)]

Sustituyendo i=n-1 X(1)=4

3**(n-1) [X(n-n+1)]

3**(n-1) [x(1)]

3**(n-1) 4

n**3

X(n) es de orden n3**

c)X(n)= X(n-1)+n para n>0 X(0)=0

[X(n-2)+n]+n

[X(n-3)+n]+2n

Patron X(n-i)+in

Sustituyendo i=n, X(0)=0

X(n-n) + n*n

X(0) + n**2

n**2

X(n) es de orden n2**

d)X(n)=X(n/2)+2 para n>1, X(1)=1

Resolver para n=2**k

X(n/4)+2+2

X(n/8)+2+2+2

Parton X(n/2**i)+2**i

Susutituyendo i = log_2(n) y X(1)=1

X(n/2**(log_2(n))+2**(log_2(n))

X(n/n) +n

X(1) + n

1+n

X(n) es de orden n

ALgoritmo misterioso(n):

```
INICIO
    S<-0                    //O(1)
    Para i<-1 Hasta n Hacer //Se repite n veces
        S<-S+ i*i           //O(1)
        //i<-i+1
    Fin Para
    Devolver s              //O(1)
Fin
```

¿Que calcula el algoritmo?

iteracion	S
Inicio	0
1	1
2	5
3	14
4	30
5	55
6	105

La suma de términos cuadrados de 1 hasta n

¿Cual es la operación básica?

La suma de la multiplicacion i*i

¿Cuantas veces se ejecuta la operación básica en el caso base?

1

¿Cual es la eficiencia del algoritmo?

x(n) = X(n)+1 para n>0

=[X(n-1)+1]+1=X(n-1)+2

=[X(n-2)+1]+2=x(n-2)+3

=[X(n-3)+1}+3=x(n-3)+4

Patron x(n-i)+(i+1)

Sustituyendo i=(n-1) y x(1)=1

x(n-n+1)+(n-1+1)

x(1) +n

1 + n **X(n) es de orden n**

¿Cuáles son los pasos para analizar Algoritmos recursivos?

1.Decidir el parámetro n, que indica el tamaño de la entrada

2.Identificar la operación básica del algoritmo (caso base)

3.Determinar si el número de veces que la operación básica es ejecutada puede variar para diferentes entradas del mismo tamaño (analizar el peor caso, caso medio y el mejor caso)

4.Expresar somo una relación de recurrencia(el número total de operaciones de la relación básica) con una condición inicial el número total de operaciones de la operación básica

5.Resolver la relación de recurrencia para encontrar la fórmula Explícita para el término genérico que satisfaga la recurrencia Y la condición inicial o probar que no existe.

Factorial de un número

```
Algoritmo factorial(n):
        SI n=0 Entonces
                devolver 1
        SINO
                devolver factorial(n-1)*n:
        Fin SI
FIN
```

Solución
1. n es el número del cual se calcula el factorial
2.Operación básica es la multiplicación factorial(n-1)*n : M(n)
3. No hay variaciones
4. M(0) = 0 para n = 0
M(n) = M(n-1) + 1 para n>0
F(n-1) factorial(n-1)*1
EL 1 representa la multiplicación de Factorial(n-1)*1
EL F(n-1) representa la llamada a factorial

5. M(n) = M(n-1)+1 sustituir M(n-1) por M(n-2) + 1
= [M(n-2)+1]+1
= M(n-2)+2 sustiruir M(n-2) por M(n-3) + 1
= [M(n-3)+1]+1
= M(n-3)+3
Patron = M(n) = M(n-i)+1
Condicion inicial M(0) = 0
i=n
M(n) = M(n-i) + i
=M(n-n) + n
=M(0) + n
= 0 + n
= n

Este algoritmo es de orden lineal O(n)

Torres de Hanoi

Algoritmo Hanoi (n):
hanoi (inicial, destino, auxiliar):
hanoi(n-1, inicial, auxiliar, destino)
mover(incial, destino)
hanoi(n-1, auxiliar, destino, origen)

1. N es el número de discos
2. Operación básica mover un disco: M(n)
3. No existen variaciones en la cantidad de operaciones que se van a ejecutar con una entrada de n elementos
4. M(1) = 1
M(n) = M(n-1) + i + M(n-1)
M(n) = 2M(n-1) + 1 para n > 1
5. Sustituir
M(n) = 2M(n-1)+1 sust. M(n-1) con 2M(n-2)+1
= 2[2M(n-2)+1] +1
= 2**2 M(n-2) +2 + 1 sust. M(n-2) con 2M(n-3)+1
= 2**2 [2M(n-3)+1] +2 +1
= 2**3 M(n-3) +2**2 +2**1 +2**0

Patron M(n) = 2**i M(n-i) + 2**(i-1) + 2**(i-2) +...+ 2**1 + 2**0
= 2**i M(n-i) + 2**(i) - 1

Sustituyendo i = (n-1) y M(1)=1

M(n) = 2**(n-1) M(n-(n-1)) + 2**(n-1) -1
M(n) = 2**(n-1) M(n -n +1) + 2**(n-1) -1
M(n) = 2**(n-1) M(1) + 2**(n-1) -1
M(n) = 2**(n-1) +1 + 2**(n-1) -1
M(n) = 2**(n-1) + 2**(n-1)
M(n) = 2**(n) - 2**1

M(n) es de orden 2(n)**

Calcula un factorial elevado al cubo

Algoritmo S(n)
si n=1 Entonces
Devolver 1
Sino
Devolver S(n-1)*n*n*n
Fin si
FIN

Tabla

n	s
1	1
2	216
3	13824
4	

1. n representa el numero para calcular su fctorial elevado al cubo
2. Operación Básica: Multiplicacion M(n)
3. No hay variación
4. Condicioin inciao n = 1 M(1) = 0
M(n) = M(n-1)+3
M(n) = (M(n-2) + 3) + 3
M(n) = M(n-2) + 6

5. Sustituie M(1) = 0 e i= n-1
M(n) = M(n-(n-1)) +3(n-1)
= M(1) + 3n -3
= 3n -3
M(n) es de orden n
EL algoritmo es lineal.

Busqueda Binaria

Parte de que un arreglo esta ordenado de tal forma que todos los numeros que se encuentran a la derecha del elemento central son mayores y todos los que estan a su izquierda son menores entonces la busqueda binaria va a decir quieres buscar "X" parte el arreglo en dos partes a partir del elemento central, entonces checa si X es menor o mayor que ele elemento central y busca en la derecha o la izquierda, saca un nuevo elemento central y pregunta si el elemento X es mayor o menor que el para decidir que rama se queda y la vuelve a partir hasta que lo encuentra. Lo que hace la búsqueda binaria es buscar en la mitad de los datos proporcionados, pretende ser más rápida que la búsqueda lineal, es casi la misma idea que utiliza el algoritmo de mergesort. Algoritmo donde n es el elemento a buscar:

```
INICIO
BusquedaBinaria(n, a, primero, ultimo)
  central<-(primero+ultimo)/2
  Si a[central] = n Entonces
      devolver verdadero
  Sino Si a[central]>n Entonces
        busqueda_binaria(n,a,primero,central
-1)
  Sino
        Busqueda_Binaria(n,a,central,ultimo)
  Fin SI
Fin
```

Solución
1. n: a es tamaño del arreglo
2. Busqeda_Binaria representada por B(n)
3. Si hay variaciones hay que analizar peor, medio y mejor
4. B(1)=0
B(n) = B(n/2)+1

5. Sustitir B(n/2) = B(n/4)+1
B(n) = B(n/2)+1 = B(n/4)+2
Sustiruir B(n/4) = B(n/8)+1= B(n/8)+3
Patron = B(n/2**i)+i
Buscamos b(1) = B(n/n)
2**i = n operacion contraria i = log_2(n)
= B(n/2**log_2(n)) + log_2(n)
=B(1) + log_2(n)
= 0 + log_2(n)
= log_2(n)

6. La eficiencia es de orden Log_2(n)

Mergesort

Algoritmo de ordenamiento

```
mergesort(a, primero, ultimo){
        Si primero < ultimo Entonces
                Central <- (Primero+ultimo)/2
                mergesort(a,primero,central)
                mergesort(a,central+1,ultimo)
                mezcla(a, primero, central,
ultimo)
        Fin si
fin
```

1. n: el tamaño de a
2. La operacion básica: Mezclar M(n)
3. No hay variaciones
4. Condicion incial M(1)=0
M(n) = M(n/2) + M(n/2) +1
=2M(n/2) + 1
5.M(n)=2M(n/2) sustituyendo M(n/2) = 2M(n/4)+1
=2[2M(n/4)+1]+1
=2**2[M(n/4)+2+1 Susut M(n/4) = 2 M(n/8)+1
=2**2[2M(n/8)+1]+2+1
=2**3M(n/8)+4+2+1
Patron = 2**i M(n/2**i) + 2**(i-1)

Con i=log2(n)
=2**(log2(n)) M(n/2**log2(n)) +2**log2(n)-1
=n*M(n/n)+n-1
=n*1+n-1
=2n-1
Merge sin considerar a mezcla es de orden lineal O(n)

El proceso de mergesort con la función mezcla es de tipo n*log(n)