**Instituto Politécnico Nacional**

**Escuela Superior de Cómputo**

**Análisis de Algoritmos**

**Ejercicio 2:** Complejidad temporal y análisis de casos.

**Sampayo Hernández Mauro**



**Grupo:** 3CM2

**Profesor:** *Edgardo Adrián Franco Martínez*

**Fecha de entrega:** 6 de septiembre de 2018

**Ejercicio 2: Complejidad temporal y análisis de casos.**

A) Para los siguientes 5 algoritmos determine la función complejidad temporal y espacial en términos de n. Aclarando que se han tomado, para los 5 casos, las operaciones de: Asignación, Aritméticas, Condicionales, Saltos implícitos.

A.1

**for(**i**=**1**;**i**<**n**;**i**++) ->1+(n-1)+n** (asignación + (comparaciones) +aritméticas)

**->n** (Saltos implícitos true (n-1) + salto false)

**for(**j**=**0**;**j**<**n**;**j**++) ->1+(n-1)+(n-1)** (Asignación + comparaciones + aritméticas)

**{ ->n+3** (Asignaciones y n-1 saltos false + 1 salto true)

temp**=**A**[**j**];**

A**[**j**]=**A**[**j**+**1**];**

A**[**j**+**1**]=**temp**;**

**}**

Quedando así, reduciendo

A.2

int polinomio**=**0**; ->1** (Asignación)

**for(**i**=**0**;**i**<=**n**;**i**++) ->1+(n+2)+(n+1)** (Asignación + (Comparaciones)+ (Aritméticas))

**{**

**->n+2** (Saltos true + 1 salto false)

polinomio**=**polinomio**\***z**+**A**[**n**-**i**];-> 3** (Aritméticas)

**}**

Reduciendo:

A.3

**for** i**=**1 to n **do ->1+n** (Asignación + Comparación)

**->n+1** (Saltos implícitos true (n-1) + salto false)

**for** j**=**1 to n **do ->1+n** (Asignación + Comparación)

**->n+1** (Saltos implícitos true (n-1) + salto false)

C**[**i**,**j**]** **=** 0**;**

**for** k**=**1 to n **do ->1+n** (Asignación + Comparación)

**->n+1** (Saltos implícitos true (n-1) + salto false)

C**[**i**,**j**]=**C**[**i**,**j**]+**A**[**i**,**k**]\***B**[**k**,**j**]; ->3** (Aritméticas)

Quedando así, reduciendo

A.4

anterior **=** 1**; ->1** (Asignación)

actual **=** 1**; ->1** (Asignación)

**while** **(**n**>**2**)**

**{ ->n-1** (Saltos implícitos true (n-1) + salto false)

aux **=** anterior **+** actual**; ->2** (Asignación)

anterior **=** actual**; ->1** (Asignación)

actual **=** aux**; ->1** (Asignación)

n **=** n **-** 1**; ->2** (Asignación)

**}**

Quedando así, reduciendo

A.5

**for** **(**i **=** n 1**;**j**=**0**;**i**>=**0**;**i**--,**j**++)->1+1+(n+1)+n+n** (asignación + asignación + (comparaciones) + aritméticas + aritméticas)

**->n+1** (Saltos implícitos true (n-1) + salto false)

s2**[**j**]=** s**[**i**]; ->1** (Asignación)

**for** **(**i **=** 0**;**i**<**n**;**i**++) ->1+(n+1)+n** (asignación + (comparaciones) +aritméticas)

**->n+1** (Saltos implícitos true (n-1) + salto false)

s**[**i**]=** s2**[**i**]; ->1** (Asignación)

Quedando así, reduciendo

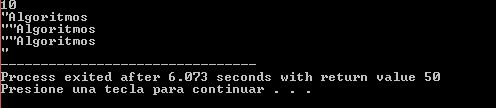
B) Para los siguientes 3 algoritmos determine el número de veces que se imprime la palabra “Algoritmos”. Determine una función lo más cercana a su comportamiento para cualquier n y compruébela codificando los tres algoritmos (Adjuntar códigos). De una tabla de comparación de sus pruebas para n’s igual a 10, 100, 1000, 5000 y 100000 y demostrar lo que la función encontrada determina será el número de impresiones.

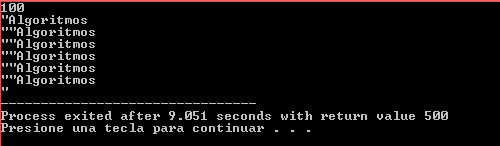
B.6

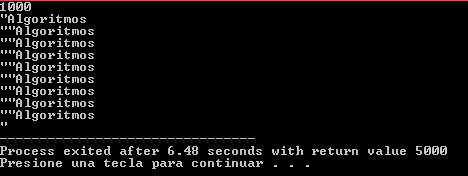
**for(**i**=**10**;**i**<**n**\***5**;**i**\*=**2**)**

printf**(**"\"Algoritmos\n\""**);**

|  |  |  |
| --- | --- | --- |
| n | # Impresiones reales | #Impresiones según la función |
| 10 | 3 | 3 |
| 100 | 6 | 6 |
| 1000 | 9 | 9 |
| 5000 | 12 | 12 |
| 100000 | 16 | 16 |







B.7

**for(**j**=**n**;**j**>**1**;**j**/=**2**) -> j se iguala a n, y se dividirá sobre 2 cada que se repita el ciclo**

**{**

**if(**j**<(**n**/**2**))**

**{**

**for(**i**=**0**;**i**<**n**;**i**+=**2**) -> n/2**

**{**

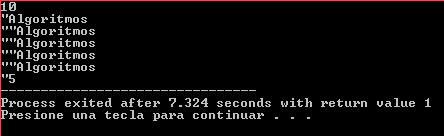
printf**(**"\"Algoritmos\n\""**);**

**}**

**}**

**}**

|  |  |  |
| --- | --- | --- |
| n | # Impresiones reales | #Impresiones según la función |
| 10 | 5 | 5 |
| 100 | 200 | 200 |
| 1000 | 3500 | 3500 |
| 5000 | 25000 | 25000 |
| 100000 | 700000 | 700000 |

****

****

****

B.8

i**=**n**; -> i se iguala a n**

**while(**i**>=**0**) -> Realiza el ciclo infinitamente debido a que i no cambia de valor**

**{**

**for** **(**j**=**n**;**i**<**j**;**i**-=**2**,**j**/=**2**) -> j se iguala a n. Como i no cambia de valor, la condición i<j nunca se cumplirá, por lo que nunca entrará al ciclo**

**{**

printf**(**"\"Algoritmos\n\""**);**

**}**

**}**

**return** 0**;**

|  |  |  |
| --- | --- | --- |
| n | # Impresiones reales | #Impresiones según la función |
| 10 | 0 | 0 |
| 100 | 0 | 0 |
| 1000 | 0 | 0 |
| 5000 | 0 | 0 |
| 100000 | 0 | 0 |

C) Para los siguientes algoritmos determine las funciones de complejidad temporal, para el mejor caso, peor caso y caso medio. Indique cual(es) son las condiciones (instancia de entrada) del peor caso y cual(es) la del mejor caso.

C.9

func Producto2Mayores**(**A**,**n**)**

**if(**A**[**1**]** **>** A**[**2**])**

mayor1 **=** A**[**1**];**

mayor2 **=** A**[**2**];**

**else**

mayor1 **=** A**[**2**];**

mayor2 **=** A**[**1**];**

i **=** 3**;**

**while(**i**<=**n**)**

**if(**A**[**i**]** **>** mayor1**)**

mayor2 **=** mayor1**;**

mayor1 **=** A**[**i**];**

**else** **if** **(**A**[**i**]** **>** mayor2**)**

mayor2 **=** A**[**i**];**

i **=** i **+** 1**;**

**return** **=** mayor1 **\*** mayor2**;**

fin

**Mejor Caso:** Los dos primeros son mayores

**Peor Caso:** Los números son crecientes o el primer número es el más grande y el resto de manera creciente.

**Caso Medio:**

# C.10

func OrdenamientoIntercambio**(**a**,**n**)**

**for** **(**i**=**0**;** i**<**n**-**1**;** i**++)**

**for** **(**int j**=**i**+**1**;** j**<**n**;**j**++)**

**if** **(**a**[** j **]** **<** a**[** i **])**

**{**

temp**=**a**[** i **];**

a**[** i **]=**a**[** j **];**

a**[** j **]=**temp**;**

**}**

fin

Operaciones básicas:

* Comparar elementos del arreglo
* Asignaciones al arreglo

**Mejor Caso:** Que estén ordenados de menor a mayor

**Peor Caso:** Que estén de mayor a menor o que el mayor sea el primero y lo demás creciente.

**Caso Medio:**

# C.11

func MaximoComunDivisor**(**m**,** n**)**

**{**

a**=**max**(**n**,**m**);**

b**=**min**(**n**,**m**);**

residuo**=**1**;**

mientras **(**residuo **>** 0**)**

**{**

residuo**=**a mod b**;**

a**=**b**;**

b**=**residuo**;**

**}**

MaximoComunDivisor**=**a**;**

**return** MaximoComunDivisor**;**

**}**

Operaciones básicas consideradas:

-Realización del módulo.

Con esto en consideración, se tienen los siguientes casos:

**Mejor Caso:** Que ambos números sean primos.

Es decir, posee un costo constante, dado a que el Máximo Común Divisor entre ambos números siempre será uno.

**Peor Caso:** Que se pretenda calcular el Máximo Común Divisor entre dos números consecutivos de la Serie de Fibonacci.

Para este caso, el tamaño de problema *n* es representado por la cantidad de bits necesaria para representar ambos números. Así, la función de complejidad temporal para el peor caso está dada por:

**Caso Medio:** Para cuando ninguna de las dos condiciones anteriores se cumpla, la función de complejidad temporal para el algoritmo está representada por:

# C.12

ProcedimientoBurbujaOptimizada(A,n)

cambios=“No”

i=0

Mientras i< n-1 && cambios !=“No” hacer

cambios=“No”

Para j=0 hasta (n-2)-i hacer

Si(A[i]<A[j]) hacer

aux=A[j]

A[j]=A[i]

A[i]=aux

cambios=“Si”

FinSi

FinPara

i= i+1

FinMientras

fin Procedimiento

Operaciones básicas consideradas:

* Comparación de los elementos del arreglo.

Bajo este criterio, podemos tener los siguientes casos:

**Mejor Caso:** Los elementos ya se encuentran en un orden creciente.

**Peor Caso:** Los elementos se encuentran ordenados en orden decreciente.

**Caso Medio:** Los elementos del vector se encuentran consistentemente aleatoriamente distribuidos. Sólo es ligeramente menor al del caso peor, con lo que puede considerarse como:

# C.13

ProcedimientoBurbujaSimple(A,n)

parai=0hastan-2hacer

paraj=0hasta(n-2)-i hacer

si(A[j]>A[j+1]) entonces

aux=A[j]

A[j]=A[j+1]

A[j+1]=aux

fin si

fin para

fin para

fin Procedimiento

Operaciones básicas:

* Comparar elementos del arreglo
* Asignaciones al arreglo

**Mejor Caso:** Que estén ordenados de menor a mayor

**Peor Caso:** Que estén de mayor a menor o que el mayor sea el primero y lo demás creciente.

**Caso Medio:**

# C.14

ProcedimientoOrdena(a,b,c)

{

if(a>b)

if(a>c)

if(b>c)

salida(a,b,c);

else

salida(a,c,b);

else

salida(c,a,b);

else

if(b>c)

if(a>c)

salida(b,a,c);

else

salida(b,c,a);

else

salida(c,b,a);

}

Operaciones básicas:

* Comparar los elementos
* La salida de los elementos

**Mejor Caso:** Que se los numero este invertidos de posición(c > b >a) o que c > a > b

**Peor Caso:** Que se cumplan los demás casos

**Caso Medio:**

# C.15

ProcedimientoSeleccion(A,n)

Para k=0hasta n-2 hacer

p=k

para i=k+1hasta n-1 hacer

si A[i]<A[p] entonces

p=i

fin si

fin para

temp=A[p]

A[p]=A[k]

A[k]=temp

fin para

fin Procedimiento

Operaciones básicas:

* Comparar elementos del arreglo
* Asignaciones al arreglo

En este algoritmo, siempre se va a pasar por las operaciones básicas, por lo tanto, no van a existir casos diferentes, aunque el tamaño del arreglo sea el que sea.