



Instituto Politécnico Nacional
Escuela Superior de Cómputo



ANÁLISIS DE ALGORITMOS NO RECURSIVOS

Ejercicio 03

Análisis de Algoritmos

M. en C. Edgardo Adrián Franco Martínez

Grupo: 3CM3

Fecha: 20 /Abril/2018



Alumno:

Calva Hernández José Manuel

2017630201

Índice

Algoritmo 1 2

Algoritmo 2 2

Algoritmo 3 3

Algoritmo 4 3

Algoritmo 5 4

Algoritmo 9 4

Algoritmo 10 5

Algoritmo 11 6

Algoritmo 12 6

Algoritmo 13 7

Algoritmo 14 8

Algoritmo 15 9

Conclusión..... 9

Algoritmo 1

```
for(i = 1; i < n; i++){  
    for(j=0; j < n-1; j++){  
        temp = A[j];  
        A[j] = A[j+1];  
        A[j+1] = temp;  
    }  
}
```

$O(n)$ $O(n^2)$

Se consideran como operaciones constantes todas las que están dentro del for más anidado, por tanto, no las consideramos para el análisis. Partiendo de ahí, comenzaremos el análisis del for más anidado, que podemos ver que va de 0 a $n-1$, lo que implicaría que hará $n-1$ operaciones, pero dado que estamos analizando por cotas, reducimos a $O(n)$ las operaciones. De forma similar el for externo realiza $n-1$ operaciones, pero lo acotamos a n operaciones, que combinándolas con el for interno, nos dará una cota total de $O(n^2)$ para todo el algoritmo.

Algoritmo 2

```
polinomio = 0;  
for (int i = 0; i <= n; ++i)  
{  
    polinomio = polinomio * z + A[n-i];  
}
```

$O(n)$

El algoritmo es sencillo, lo que tenemos dentro y fuera del for se consideran operaciones constantes al tratarse de asignaciones y operaciones aritméticas, por lo tanto, el resultado del análisis del for será la cota total del algoritmo. Podemos observar que el algoritmo realizará $n+1$ operaciones, sin embargo, podemos acotar esto a $O(n)$ que será el total de nuestro análisis.

Algoritmo 3

```
for i = 1 to n do
{
  for j = 1 to n do{
    c[i,j] = 0;
    for k = 1 to n do{
      c[i,j] = c[i,j] + A[i,k] * B[k,j]; } O (n) } O (n2) } O (n3)
    }
  }
}
```

Iniciando en el for más anidado, observamos que su operación la consideramos constante al ser aritmética, por tanto, será irrelevante para el análisis, a partir de ello, observamos que la iteración recorrerá todos los valores posibles de n , por lo tanto, lo acotaremos como $O(n)$. El siguiente for más anidado tiene una asignación que será irrelevante para el análisis al considerarse constante, entonces analizaremos el for que realizará la misma iteración sobre los valores posibles de n , que tomando en cuenta el for que tiene dentro, consideramos su cota como $O(n^2)$. Por último, el for externo nuevamente recorre todos los valores posibles de n , y al considerar la cota interna de $O(n^2)$, obtenemos que la cota total del algoritmo es de $O(n^3)$.

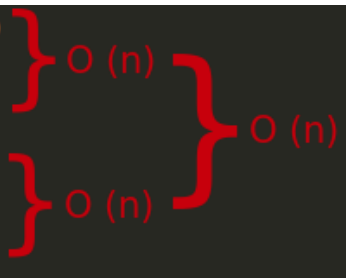
Algoritmo 4

```
anterior = 1;
actual = 1;
while(n>2){
  aux = anterior + actual; } O (n)
  anterior = actual;
  actual = aux;
  n = n-1;
}
```

Consideraremos las operaciones tanto dentro como fuera del ciclo while como irrelevantes al tomarse como constantes, por lo tanto, la cota dependerá de cómo se comporte la n dentro de la iteración. Si analizamos las operaciones internas, observamos que la n irá decrementando de unidad en unidad hasta terminar el ciclo, por lo tanto, podemos decir que realizamos $n-2$ operaciones, y esto si lo acotamos nos da el resultado de $O(n)$.

Algoritmo 5

```
for (int i = n-1, j = 0; i >= 0; --i, j++)  
{  
    s2[j] = s[i];  
}  
for (int i = 0; i < n; ++i)  
{  
    s[i] = s2[i];  
}
```

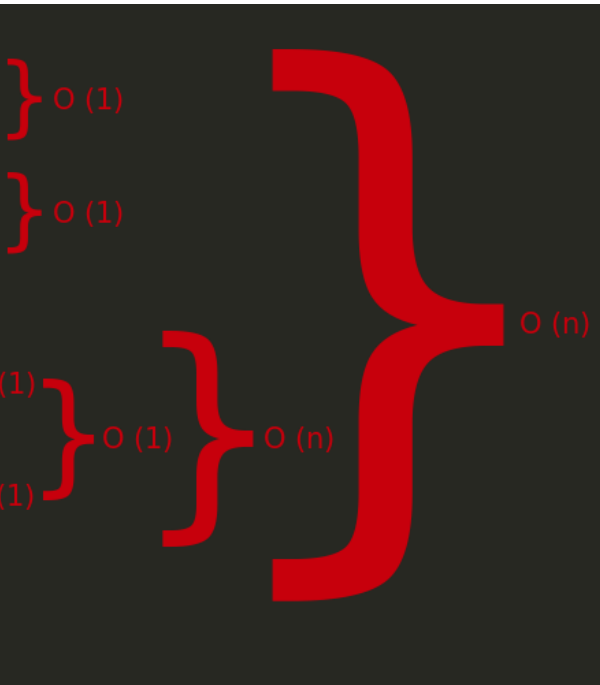


Podemos observar que las operaciones en ambos ciclos for son constantes, por lo tanto, serán irrelevantes para el análisis. Así, tomamos el primer for y es apreciable que la i se inicia en $n-1$, e irá decrementando hasta llegar a -1 , por lo tanto, se realizarán n iteraciones dado que la j es irrelevante para el ciclo. En lo que respecta al segundo for, es más sencillo de observar que nuevamente realizará n iteraciones.

Como resultado obtendríamos $2n$ operaciones al sumar ambos resultados, pero esto se acota a $O(n)$ dado que esto describe su comportamiento para valores de n muy grandes.

Algoritmo 9

```
func Producto2Mayores(A,n){  
    if (A[1] > A[2])  
    {  
        mayor1 = A[1];  
        mayor2 = A[2];  
    }  
    else{  
        mayor2 = A[1];  
        mayor1 = A[2];  
    }  
    i = 3;  
    while(i<=n){  
        if (A[i]>mayor1)  
        {  
            mayor2 = mayor1;  
            mayor1 = A[i];  
        }  
        else if (A[i]>mayor2)  
        {  
            mayor2 = A[i];  
        }  
        i++;  
    }  
    return = mayor1 * mayor2;  
}  
fin
```



Iniciando con las operaciones más anidadas, que son las comparaciones dentro del while, podemos considerarlas constantes al realizar únicamente las asignaciones correspondientes al evaluar las comparaciones. Por lo tanto, pasaremos a observar el comportamiento del while que vendrá dado por un incremento unitario en la i hasta alcanzar la n , sin embargo, hay que tomar en cuenta que la i se inicializó en 3, lo que resulta en que haremos $n-3$ operaciones, que acotándolo resultará en una cota de $O(n)$. Las operaciones fuera del ciclo while son constantes, por lo tanto, al compararlas con la cota que acabamos de encontrar, podemos concluir que la cota de esta función es de $O(n)$.

Algoritmo 10

```
func OrdenamientoIntercambio(a,n){  
  for (int i = 0; i < n-1; ++i)  
  {  
    for (int j = i; j < n; ++j)  
    {  
      if (a[j]<a[i])  
      {  
        temp = a[i];  
        a[i] = a[j];  
        a[j] = temp;  
      }  
    }  
  }  
}  
fin
```

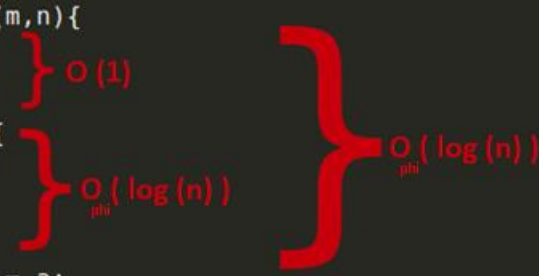
The diagram illustrates the complexity analysis of the provided code. It uses red curly braces to group different parts of the code and their corresponding time complexities:

- A small brace groups the `if (a[j]<a[i])` statement and the three assignment lines (`temp = a[i];`, `a[i] = a[j];`, `a[j] = temp;`) inside the innermost loop, labeled as $O(1)$.
- A medium brace groups the entire inner `for` loop (from `for (int j = i; j < n; ++j)` to its closing brace), labeled as $O(n)$.
- A large brace groups the entire function body (from the first `for` loop to the final closing brace), labeled as $O(n^2)$.

Iniciando por el if más anidado, observamos que tanto su comparación como las asignaciones dentro de él son constantes, por lo tanto, no será relevante para el análisis. El siguiente for que está anidado podemos observar que en el peor de los casos recorrerá todos los valores posibles de n , por lo tanto, su cota será de $O(n)$. El for externo siempre hará un recorrido por los $n-1$ valores de n , sin embargo, esto podemos acotarlo a n , que aunado a la cota que acabamos de encontrar interna, nos resulta en una cota de $O(n^2)$ para toda la función.

Algoritmo 11

```
fun MaximoComunDivisor(m,n){  
  a = max(n,m);  
  b = min(n,m);  
  residuo = 1;  
  while(residuo > 0){  
    residuo = a%b;  
    a = b;  
    b = residuo;  
  }  
  MaximoComunDivisor = a;  
  return MaximoComunDivisor;  
}  
fin
```



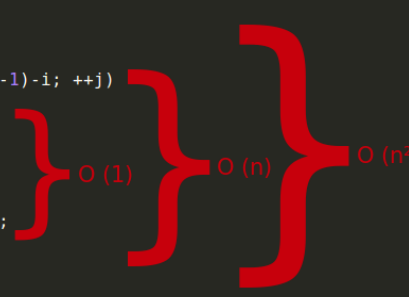
Complexity analysis for Algorithm 11:

- The initialization of `a` and `b` is $O(1)$.
- The `while` loop body is $O(\log_{\phi}(n))$.
- The overall complexity of the `while` loop is $O(\log_{\phi}(n))$.

Comenzaremos considerando como constantes las operaciones dentro del ciclo `while` que es el más anidado, por lo tanto, continuaremos analizando el comportamiento de la iteración, que en el peor de los casos se ha dicho que serán dos números consecutivos de la serie de Fibonacci, que lleva asociada una complejidad relacionada con el número áureo. Por lo que podemos expresar la cota del `while` como $O(\log_{\phi}(n))$ donde n será el máximo valor entre n y m , es decir, el número k_{m+1} de la serie de Fibonacci. Las demás operaciones se consideran constantes al tratarse únicamente de asignaciones, por lo que la cota de la función estará dada por $O(\log_{\phi}(n))$.

Algoritmo 12

```
Procedimiento BurbujaOptimizada(A,n){  
  cambios = true;  
  i = 0;  
  while(i < n-1 && cambios){  
    cambios = false;  
    for (int j = 0; j < (n-1)-i; ++j)  
    {  
      if (A[i] < A[j])  
      {  
        aux = A[j];  
        A[j] = A[i];  
        A[i] = aux;  
        cambios = true;  
      }  
    }  
    i++;  
  }  
}  
fin
```



Complexity analysis for Algorithm 12:

- The `if` block inside the `for` loop is $O(1)$.
- The `for` loop is $O(n)$.
- The `while` loop is $O(n^2)$.

Se tomarán como constantes todas las operaciones realizadas dentro del bloque `if`, así como la comparación que realiza éste, por lo tanto, la cota de ese bloque se considera como $O(1)$. Yendo al `for` anidado, podemos observar que, en el peor caso, es decir, la primer iteración, se realizarán las operaciones sobre todos los posibles valores de n , por lo que su cota será $O(n)$. Siguiendo con el `while` externo, sus dos operaciones se consideran constantes

así que tomaremos como relevante la cota $O(n)$ del ciclo for, pasaremos a analizar el comportamiento del ciclo while, que podemos observar que va de 0 a $n-1$, que acotando nos resulta en n operaciones, que, al considerarlo junto con la cota interna, resulta que el ciclo while en total tiene una cota de $O(n^2)$. Por último, las dos operaciones iniciales se consideran constantes y son irrelevantes para el análisis, por lo que la cota general del algoritmo será de $O(n^2)$.

Algoritmo 13

```
Procedimiento BurbujaSimple(A,n){  
  for (int i = 0; i < n-1; ++i)  
  {  
    for (int j = 0; j < (n-1)-i; ++j)  
    {  
      if (A[j]>A[j+1])  
      {  
        aux = A[j];  
        A[j] = A[j+1];  
        A[j+1] = aux;  
      }  
    }  
  }  
}  
fin
```

The diagram illustrates the complexity analysis of the `BurbujaSimple` algorithm. Red curly braces are used to group the code blocks and their corresponding time complexities:

- The innermost `if` block is labeled $O(1)$.
- The inner `for` loop is labeled $O(n)$.
- The outer `for` loop is labeled $O(n^2)$.

Iniciaremos considerando como constante todo el bloque `if` interno, siguiendo el mismo razonamiento del algoritmo anterior, posteriormente continuaremos con el `for` anidado que va en el peor de los casos por los $n-1$ valores posibles, que acotando nos resulta en una $O(n)$ interna. Siguiendo con el `for` externo, observamos que recorrerá siempre los $n-1$ valores posibles, que acotaremos a n , y que, aunado a la cota resultante del bloque interno, nos dará una cota $O(n^2)$ para todo el algoritmo.

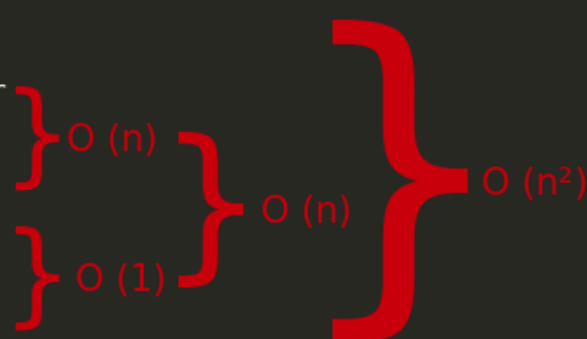
Algoritmo 14

```
Procedimiento Ordena(a,b,c){  
  if (a>b)  
  {  
    if (a>c)  
    {  
      if (a>c) { salida(a,b,c); } O(1)  
    }  
    else { salida(a,c,b); } O(1)  
  }  
  else { salida(c,a,b); } O(1)  
}  
else {  
  if (b>c)  
  {  
    if (a>c) { salida(b,a,c); } O(1)  
  }  
  else { salida(b,c,a); } O(1)  
}  
else { salida(c,b,a); } O(1)  
}  
}  
fin
```

Este algoritmo es sencillo de analizar, ya que al desconocer lo que hace la función “salida”, consideramos que es constante, y dado que las comparaciones de los if / else son consideradas constantes también, podemos concluir que el costo total del algoritmo es constante, o bien, estrictamente hablando diremos que el costo dependerá de la función salida.

Algoritmo 15

```
Procedimiento Seleccion(A,n)
  Para k=0 hasta n-2 hacer
    p = k
    Para i=k+1 hasta 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
```



Iniciaremos el análisis en el ciclo más anidado, en este caso es el que irá desde $k+1$ hasta $n-1$, en primer lugar, diremos que sus operaciones son constantes dado que se tratan de asignaciones y comparaciones únicamente. Dicho esto, observamos que el ciclo en el peor caso irá desde 1 hasta $n-1$, por lo que realizará $n-1$ operaciones que podemos acotar como $O(n)$. A continuación, analizaremos el ciclo externo, tendremos en cuenta que el ciclo interno nos ha dado una cota de $O(n)$, y si miramos el resto del bloque, observamos que se trata de operaciones de asignación, por lo que todas serán consideradas constantes e irrelevantes para el análisis. Por lo tanto, nos centraremos en el comportamiento del ciclo, que nos dice que irá de 0 hasta $n-2$, que serán $n-1$ operaciones en todos los casos y que podremos acotar como n operaciones, que, aunado a la cota interna, podemos concluir que la cota para este algoritmo es de $O(n^2)$.

Conclusión

Tomando en cuenta el análisis realizado en el ejercicio 2, podemos observar concluir que no se variaría el resultado al establecer cotas, debido a que, para una n muy grande, no es perceptible la diferencia generada por el factor multiplicativo, mucho menos la generada por variables de menor grado o que tienen un factor de crecimiento menor llegado a esa magnitud. Por tanto, el análisis realizado en ambos casos es correcto y nos determinaría la misma cota.