Instituto Politecnico Nacional

ESCOM “ESCUELA SUPERIOR DE CÓMPUTO”

ANÁLISIS DE ALGORITMOS

EJERCICIO 4: ALGORITMOS NO RECURSIVOS

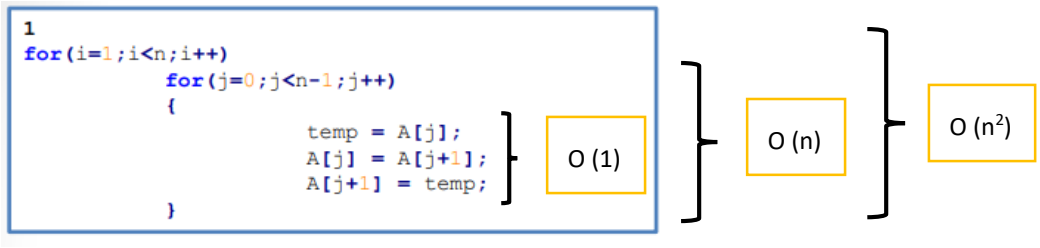
PROFE: Edgardo Adrián Franco Martínez

ALUMMNO: Rojas Alvarado Luis Enrique

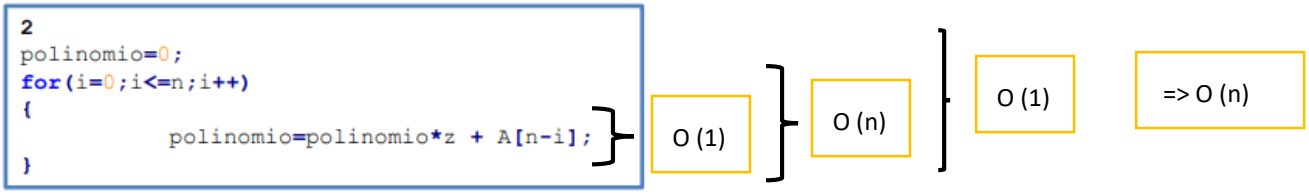
GRUPO: 3CM4

ANALISIS DE LOS ALGORITMOS NO RECURSIVOS

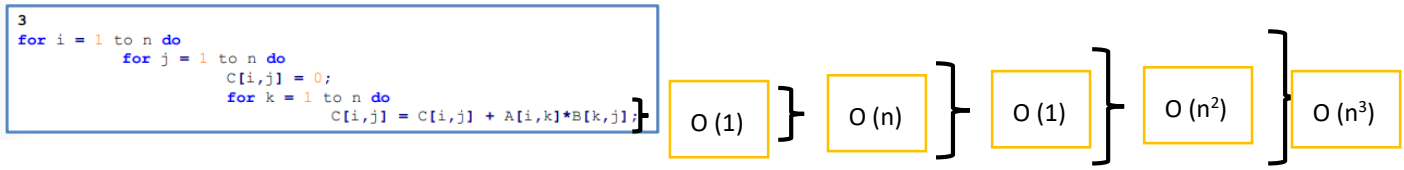
Se requiere analizar los siguientes algoritmos y determinar su cota O().



Analizando desde dentro hacia afuera se puede decir que hay solo asignaciones y operaciones aritméticas por lo que es de orden constante O(1). Posteriormente como las asignaciones están dentro de un ciclo for el cual se repetirá n-1 veces se puede decir que toda la función es de orden n u O(n) y esto está contenido en un segundo for anidado, el cual se repetirá n-1 de igual manera que el anterior. Como entrará n veces al ciclo for anidado se puede decir que éste algoritmo es de orden n^2. Puesto que O (n^2) es el mayor orden en el algoritmo, se puede decir que toda la función es de éste orden,



Analizando el algoritmo de dentro hacia afuera, tenemos operaciones aritméticas y asignaciones, por lo que el orden de ésta linea es constante. Posteriormente hay un ciclo for que se repetirá n-1 veces, siendo el orden del algoritmo O(n), por último hay una asignación por fuera del ciclo, siendo de complejidad constante. La complejidad de todo el algoritmo es de O(n) puesto que es la complejidad más alta.



Haciendo un análisis desde dentro hacia afuera, se tiene que hay asignaciones y operaciones aritméticas siendo la complejidad de ésta línea O(1), ésta línea se encuentra dentro de un ciclo for que se repetirá n-1 veces, siendo la complejidad O(n), arriba del ciclo se encuentra una asignación siendo la omplejidadad O(1), sabiendo que se encuentra dentro de otro ciclo for con n-1 repeticiones la complejidad se convierte en O(n^2) y todo lo engloba otro ciclo terminando en O(n^3).

```
4
anterior = 1;
actual = 1;
while (n>2)
{

   aux = anterior + actual;
   anterior = actual;
   actual = aux;
   n = n - 1;
}
```

O (1)

O (n)

O (1)

=> O (n)

Haciendo un análisis desde dentro hacia afuera, dentro del ciclo while se tienen 4 asignaciones y 2 operaciones aritméticas, siendo la complejidad de O(1), analizando el ciclo while, se nota que se repetirá n-2 veces siendo la complejidad de O(n). Al inicio de todo el algoritmo hay 2 asignaciones, siendo la complejidad O(1). Así la complejidad de todo el algoritmo es de O(n) puesto que es la más grande jerarquicamente.

```
5
for (i = n - 1,j=0;i>=0;i--,j++)
            s2[j]= s[i];

for (i = 0,i<n;i++)
            s[i]= s2[i];
```

O (1)

O (1)

O (n)

O (n)

=> O (n)

Haciendo un análisis de los dos ciclos, se puede notar que dentro de los ciclos hay una asignación, siendola complejidad de O(1) y los mismos ciclos se repiten n-1 veces, siendo la complejidad de ambos ciclos O(n), lo que implica que la complejidad de todo el método es de O(n).

```
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
```

O (1)

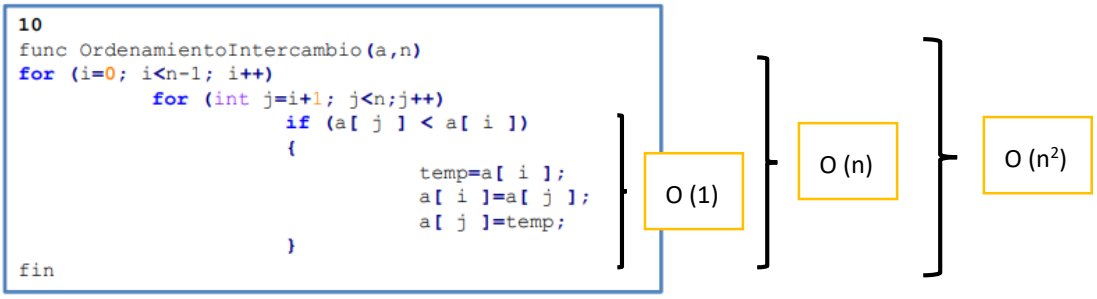
O (1)

O (n)

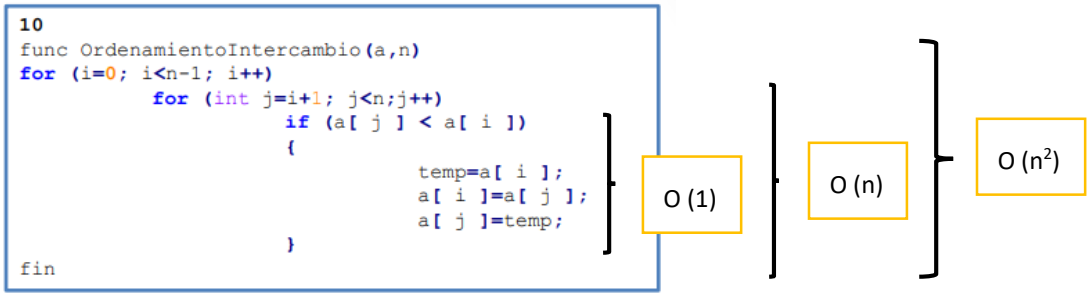
O (1)

=> O (n)

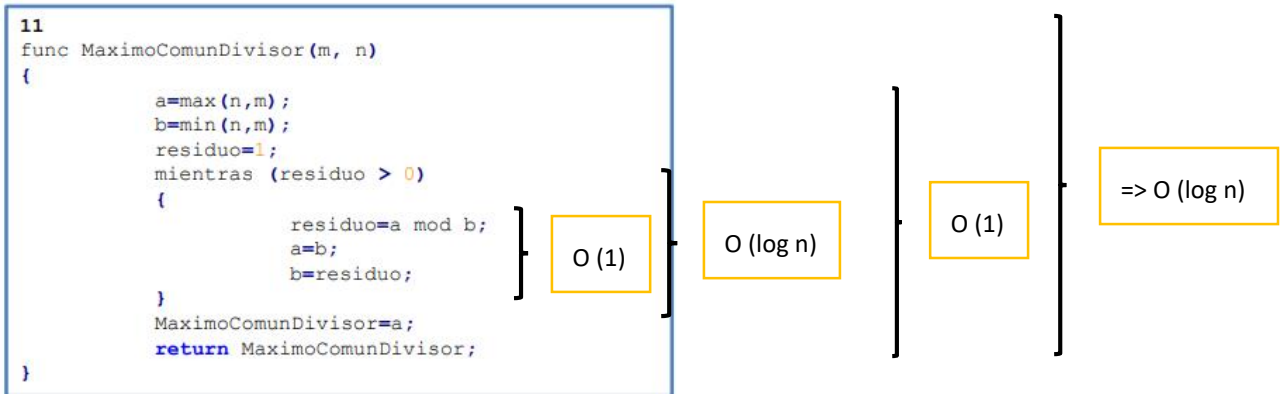
Primero se analizaron los condicionales, dado que las comparaciones y las asignaciónes dentro de los mismos se consideran de orden constante O(1), de la misma manera que la asignación de la variable i. Haciendo el análisis del while desde dentro hacia afuera, se tiene un condicional que de igual manera que los anteriores su complejidad es considerada como constante O(1). El while se repite n-2 veces y su complejidad es de O(n), hasta abajo hay un return y una asignacion con una operación aritmética, siendo su complejidad O(1). Como la complejidad más grande en todo el método es de O(n), todo el algoritmo es de ésta complejidad.



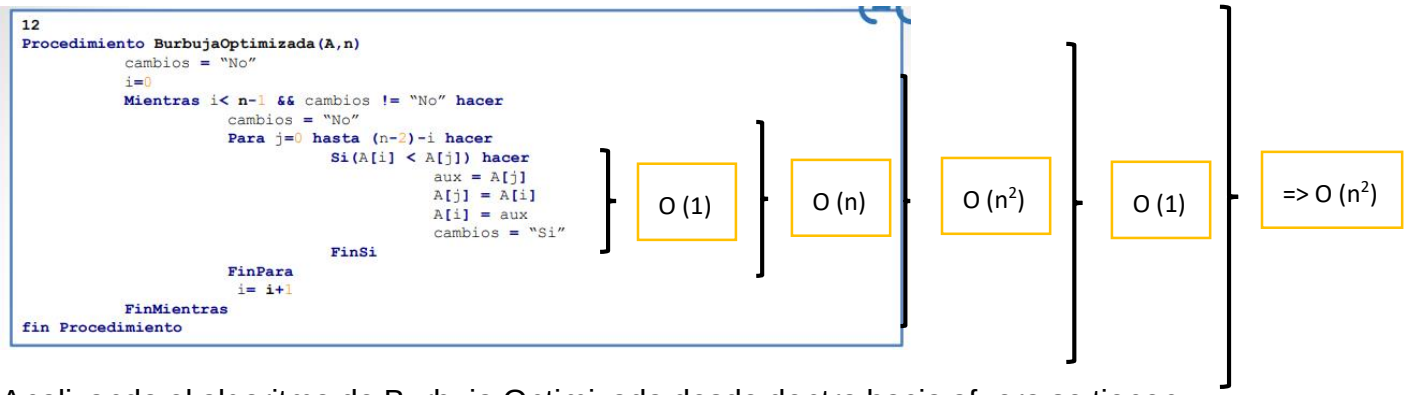
Analizando el algoritmo de ordenamiento de intercambio desde el interior hacia el exterior, se tienen 3 asignaciones dentro de un condicional por lo que se hace la complejidad de O(1), como éstas asignaciones están dentro de un ciclo for en donde se repetirá n-1 veces, la complejidad es de O(n), y está anidado con otro ciclo en el cual se va a repetir nuevamente n-1 veces haciendo la complejidad de todo el algoritmo de $O(n^2)$.



Analizando el algoritmo de Ordenamiento Intercambio desde dentro, hay 3 asignaciones dentro de un condicional, por lo que la complejidad es de O(1). Lo que contiene al condicional es un ciclo for donde se repetirá n-1 veces, haciendo la complejidad O(n), éste ciclo for está contenido por otro ciclo for anidado, el cual se repetirá de igual manera n-1 veces, haciendo la complejidad del algoritmo $O(n^2)$.

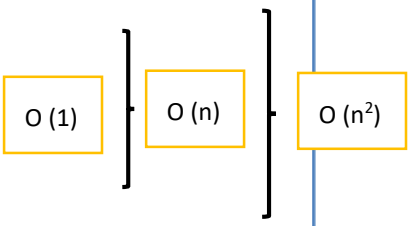


Analizando el algoritmo del máximo común divisor empezando desde dentro del while, se tienen 3 asignaciones. Como el cociente es por lo menos 1 para cualquier entrada, siempre vamos a obtener 1, se reducen al mínimo 2 números en cada iteración, por lo que la complejidad es $\log_2(n)$, fuera del ciclo se tienen 3 asignaciones en la parte superior y en la parte inferior una asignación y un return, por lo que su complejidad es de O(1). Dado que la complejidad más grande es de O(log n), ésta es la complejidad de todo el algoritmo.



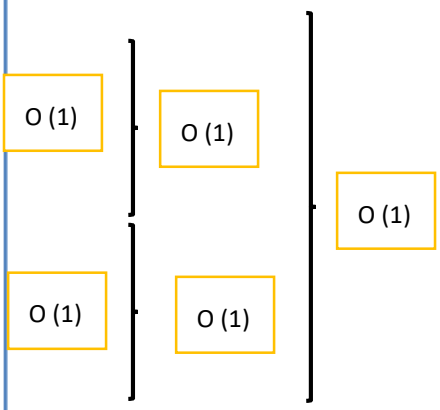
Analizando el algoritmo de Burbuja Optimizada desde dentro hacia afuera se tienen 4 asignaciones dentro de un condicional, por lo que la complejidad de ésta parte es de O(1), el ciclo for que contiene este condicional se repetirá n-1 veces haciendo el orden del algoritmo O(n) hasta el momento, posteriormente hay una asignación arriba del for y una más cuando termina el for pero es descartable puesto que el while que se repetirá n-1 veces hace que la complejidad lineal O(1) se vea opacada por la jerarquía de complejidad, haciéndola $O(n^2)$. Hay 2 asignaciones antes de que empiece el ciclo while O(1). Por lo que la complejidad de todo el algoritmo de burbuja optimizada es de $O(n^2)$.

```
13
Procedimiento BurbujaSimple(A,n)
          para i=0 hasta n-2 hacer
                    para j=0 hasta (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
```



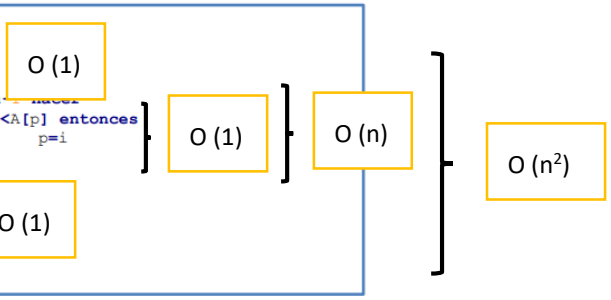
Analizando el algoritmo de Burbuja Simple desde dentro hacia afuera, se tiene un condicional con 3 asignaciones, haciendo su complejidad O(1). Posteriormente existe un ciclo for que se repetirá n-1 veces, haciendo la complejidad O(n) y puesto que está anidado con otro ciclo for haciendo la complejidad de todo el algoritmo Burbuja Simple la complejidad de $O(n^2)$.

```
14
Procedimiento Ordena(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);
}
```



Analizando el algoritmo de Ordena desde dentro hacia afuera se tienen 3 condicionales if anidados dentro de ellos solo imprime ordenados las entradas que se le dieron al algoritmo, siendo la complejidad de todo el algoritmo O(1).

```
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
```



Analizando el algoritmo de selección desde dentro hacia afuera, se tiene un condicional con una asignación dentro de él, por lo que la complejidad es lineal O(1), éste condicional está dentro de un ciclo for, haciendo la complejidad O(n), fuera de

éste ciclo for en la parte de arriba se tiene una asignación y abajo tiene 3 asignaciones, haciendo la complejidad de O(1) y siendo opacada por la complejidad O(n). Todo lo anterior está contenido en un for que se repetirá n-1 veces, haciendo la complejidad de todo el algoritmo de selección $O(n^2)$.

CONCLUSION

Éste tipo de análisis se facilita mucho saber la complejidad de cualquier algoritmo sin necesidad de hacer un análisis más profundo, nos guiamos por los peores casos y sólo se utilizan las complejidades lineal, constante, log n, etc. También éste tipo de análisis nos dice que tan eficiente es de acuerdo con la complejidad determinada y cómo se comporta de manera general.

La ventaja de hacer éste tipo de análisis es que coincide perfectamente con el análisis detallado que hicimos anteriormente en el ejercicio 2, siendo el análisis más fácil y rápido de las complejidades temporales.