INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

--------------------ANÁLISIS DE ALGORITMOS------------------

ACTIVIDAD

Análisis De Algoritmos No Recursivos

PROFESOR:

Franco Martínez Edgardo Adrián

ALUMNO:

Meza Vargas Brandon David – 2020630288

GRUPO:

3CM13

índice

Código 01

```
func SumaCuadratica3Mayores(A,n)
{
    if(A[1] > A[2] && A[1] > A[3])
        m1 = A[1];
        if (A[2] > A[3])
            m2 = A[2];
            m3 = A[3];
        else
            m2 = A[3];
            m3 = A[2];
    else if(A[2] > A[1] && A[2] > A[3])
        m1 = A[2];
        if (A[1] > A[3])
            m2 = A[1];
            m3 = A[3];
        else
            m2 = A[3];
            m3 = A[1];
    else
        m1 = A[3];
        if (A[1] > A[2])
            m2 = A[1];
            m3 = A[2];
        else
            m2 = A[2];
            m3 = A[1];

    i = 4;


    while(i<=n)
        if(A[i] > m1)
            m3 = m2;
            m2 = m1;
            m1 = A[i];
        else if (A[i] > m2)
            m3 = m2;
            m2 = A[i];
        else if (A[i] > m3)
            m3 = A[i]

        i = i + 1;

    return = pow(m1 + m2 + m3,2);
```

Si analizamos el primer if anidado nos damos cuenta que será O(1), esto lo vemos rápidamente ya que todos los ifs que están dentro siempre tiene instrucciones constantes, siendo todo este bloque O(1).

Pasando a los ifs dentro del ciclo podemos ver que de igual forma sus instrucciones son constantes, siendo estos O(1).

Por último el ciclo while es de cota O(n), siendo la cota superior ajustada del algoritmo:

O(n)

```
func SumaCuadratica3Mayores(A,n){
    if(A[1] > A[2] && A[1] > A[3])
        m1 = A[1];                  -- O(1)
        if(A[2] > A[3])
            m2 = A[2];              -- O(1)
            m3 = A[3];              -- O(1)
        else
            m2 = A[3];              -- O(1)
            m3 = A[2];              -- O(1)
    else if(A[2] > A[1] && A[2] > A[3])
        m1 = A[2];                  -- O(1)
        if(A[1] > A[3])
            m2 = A[1];              -- O(1)
            m3 = A[3];              -- O(1)
        else
            m2 = A[3];              -- O(1)
            m3 = A[1];              -- O(1)
    else
        m1 = A[3];
        if(A[1] > A[2])
            m2 = A[1];              -- O(1)
            m3 = A[2];              -- O(1)
        else
            m2 = A[2];              -- O(1)
            m3 = A[1];              -- O(1)

    i = 4;                          -- O(1)

    while(i <= n)                   -- O(n)
        if(A[i] > m1)               -- O(1)
            m3 = m2;                -- O(1)
            m2 = m1;                -- O(1)
            m1 = A[i];              -- O(1)
        else if(A[i] > m2)          -- O(1)
            m3 = m2;                -- O(1)
            m2 = A[i];              -- O(1)
        else if (A[i] > m3)         -- O(1)
            m3 = A[i];              -- O(1)

        i = i +1;                   -- O(1)
}
```

Código 02

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

Para este algoritmo vemos que las instrucciones del if todas son O(1), siendo todo el if O(1).

El segundo for depende de i, pero i en su peor momento vale 0, siendo el peor caso de este for que se repita n-1 veces, por lo tanto es 0(n), ahora el segundo ciclo y el if son O(n).

Por último, el primer for se repite de igual forma n-1 veces, siendo O(n), multiplicando el orden del for externo y el bloque interno nos queda **O(n^2):**

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

O(1)

O(N)

O(N^2)

Código 03

```
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;
```

Analizando este algoritmo nos damos cuenta que las instrucciones dentro del ciclo mientras tienen O(1), esto pasa de igual forma con las instrucciones fuera de este ciclo, así tenemos que la cota dependerá de la cota que tenga el ciclo.

Y el ciclo while se repite hasta que el residuo sea 0, en esta parte podemos considerar que el ciclo requiere de n repeticiones para reducir el residuo a 0, pero esta reducción no va de uno en uno, es decir, no es línea, es exponencial ya que de igual forma va reduciendo n veces.

Si consideramos dos números consecutivos de la serie de Fobonacci, tenemos que las veces que se repite el ciclo mientras es la cantidad de números de la serie de Fibonacci antes del número menor.

Si tenemos m = 21 y n = 34, las veces que se repite el ciclo son 7.

Para encontrar este 7 recurrimos a la fórmula de Binet, la cual tiene una aproximación a :

$$f_N = \emptyset^N$$

Donde fn es el n-esimo numero de la serie de Fibonacci y N es la posición en la serie, de esta forma ya podemos obtener cuantas veces se repite el ciclo while, tan solo tenemos que aplicar $log\emptyset$ en ambos lados de la ecuación y sustituyendo fn por el valor más pequeño entre m y n:

$$\log_{\emptyset}(\min(n, m)) = \log \emptyset \, \emptyset^N$$

$$\log_{\emptyset}(n, m) = N$$

De esta forma tenemos que la cota será:

$$\mathbf{O}(\mathbf{log}_{\emptyset}(\boldsymbol{n},\boldsymbol{m}))$$

```
func MaximoComunDividor(m, n){
    a = max(n,m);              -- O(1)
    b = min(n,m);              -- O(1)
    residuo = 1;
    mientras(residuo > 0){    --O(log(min(n,m)))
        residuo = a mod b;    -- O(1)
        a = b;                -- O(1)
        b = residuo;          -- O(1)
    }

    MaximoComunDividor = a; -- O(1)
    return MaximoComunDividor;

}
```

O(LOGØ(MIN(N,M)))

Código 04

```
func SumaCuadratica3MayoresV2(A,n)
{
    for(i=0;i<3;i++)
    {
        for (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;
            }
        }
    }
    r=A[n-1] + A[n-2] + A[n-3];
    return pow(r,2);
```

Analizando el if interno, vemos que cada operación tiene O(1), por lo tanto ese if es O(1).

El segundo if depende de la i del primer for, pero esta i en su peor caso vale 0, por lo tanto este ciclo se repite n veces, siendo O(n).

Por último el primer for se repite tan solo 3 veces, podemos decir que su cota es de 0(3), sin embargo la cota mayor es la del for interno que resulto ser O(n) de esta forma la cota superior ajustada del algoritmo será:

O(n)

```
func sumaCuadratica3MayoresV2(A,n){

    for(i=0; i<3, i++)                -- O(3)
        for(j=0; j<n-1-i; j++)        -- O(n)
            if(A[j] > A[j+1]){
                aux = A[j];           -- O(1)
                A[j] = A[j+1];        -- O(1)
                A[j+1] = aux;         -- O(1)
            }

    r = A[n-1] + A[n-2] + A[n-3]; -- O(1)
    return pow(r, 2)
```

O(I)

O(N)

O(N)

Código 05

```
Procedimiento BurbujaOptimizada(A,n)
    cambios = "Si"
    i=0
    Mientras i< n-1 && cambios != "No" hacer
        cambios = "No"
        Para j=0 hasta (n-2)-i hacer
            Si(A[j] < A[j]) hacer
                aux = A[j]
                A[j] = A[j+1]
                A[j+1] = aux
                cambios = "Si"
            FinSi
        FinPara
         i= i+1
    FinMientras
ocedimiento
```

Al igual que en los algoritmos pasados, aquí tenemos que el if dentro de los ciclos tiene cota O(1), pues sus instrucciones cuestan 1.

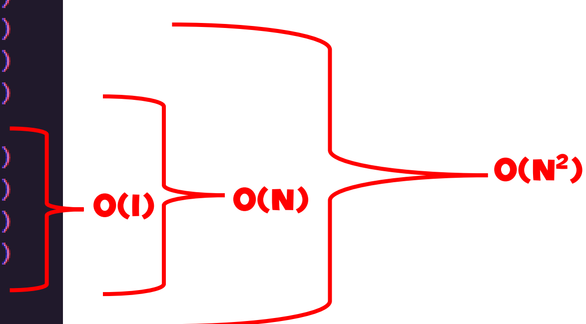
Procedemos con el ciclo para, este depende de i, pero vemos que i en su peor caso vale 0 por lo tanto es ciclo se repite n-1 veces, siendo su cota 0(n).

El ciclo mientras inicio con i igual a 0 hasta n-1, repitiéndose n veces este ciclo.

Las instrucciones fuera e los ciclos tienen un costo de 1, por lo tanto la cota superior ajustada para este algoritmo será:

$$O(n^2)$$

```
Procedimiento burbujaOptimizada(A,n){
    cambios = "Si";                                 -- O(1)
    i = 0;                                          -- O(1)
    Mientras i < n-1 && cambios != "No" hacer       -- O(n)
        cambios = "No"                              -- O(1)
        Para j = 0 hasta (n-2)-i hacer              -- O(n)
            Si(A[j] < A[j+1]) hacer
                aux = A[j]                          -- O(1)
                A[j] = A[j+1]                       -- O(1)
                A[j+1] = aux                        -- O(1)
                cambios "Si"                        -- O(1)
            FINSI
        FinPara
        i = i+1                                     -- O(1)
    FinMientras
```



Código 06

```
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
    Procedimiento
```

Este algoritmo es muy parecido al anterior, pues la cota del if es 0(1), la del ciclo es m, pues se repite n-1 veces, al igual que el for del inicio, por lo tanto la cota superior ajustada es de:

$$O(n^2)$$

```
Procedimiento burbujaSimple(A,n){
    Para i = 0 hasta n-2 hacer                          -- O(n)
        Para j = 0 hasta (n-2)-i hacer                  -- O(n)
            Si(A[j] < A[j+1]) hacer
                aux = A[j]                              -- O(1)
                A[j] = A[j+1]                           -- O(1)
                A[j+1] = aux                            -- O(1)
            FINSI
        FinPara
    FinMientras
```

O(N)
O(I)
$O(N^2)$

Código 07

```
Proceso validarPrimo
    Leer n
    divisores<-0
    si n>0 Entonces
        Para i<-1 Hasta n Hacer
            si (n%i=0) Entonces
                divisores=divisores+1
            FinSi
        FinPara
    FinSi

    si divisores=2
        Escribir 'S'
    SiNo
        Escribir 'N'
    FinSi
FinProceso
```

El primer if solo tiene un camino con cota O(1).

El ciclo para inicio en i = 1 hasta n, por lo tanto se repite n-1 veces, teniendo una cota O(n).

El primer si comprende el bloque ya analizado el cual fue de O(n).

Las últimas dos condiciones también tienen una cota de 0(1), de esta forma la cota superior ajustada para este algoritmo será:

O(n)

```
Proceso validarPrimo(){
    Leer n                                  -- O(1)
    divisores <- 0                          -- O(1)

    si > 0 Entonces
        Para i <- i Hasta n Hacer           -- O(n)
            si (n % i = 0) Entonces
                divisores = divisores + 1   -- O(1)
            FinSi
        FinPara
    FinPara

    si divisores = 2
        Escribir 'S'                        -- O(1)
    SiNo
        Escribir 'N'                        -- O(1)
    FinSi
}
```

O(1) O(N) O(N) O(N)

O(1)

Código 08

```
Algoritmo FrecuenciaMinNumeros
	Leer n
	Dimension A[n]
	i=1
	Mientras i<=n
		Leer A[i]
		i=i+1
	FinMientras
	
	f=0
	i=1
	Mientras i<=n
		ntemp=A[i]
		j=1
		ftemp=0
		Mientras j<=n
			
			si ntemp=A[j]
				ftemp=ftemp+1
			FinSi
			j=j+1
		FinMientras
		
		si f<ftemp
			f=ftemp
			num=ntemp
		FinSi
		
		i=i+1
	FinMientras
	Escribir num
FinAlgoritmo
```

Primero analicemos el primer mientras que parece, este se repite n veces, por lo tanto su cota será O(n).

Ahora veamos el segundo ciclo, este tiene un if interno, este if solo tiene un camino y sus instrucciones cuestan 1, por lo tanto su cota es 0(1).

El mientras anidado se vuelve a repetir n veces, por lo tanto el bloque que comprende este ciclo y el if dentro de el tiene una cota de 0(n).

Ahora bien hay un if más, este de igual forma tiene un costo de 1, siendo su cota O(1).

El primer for se repite n veces, siendo su cota de O(n), si multiplicamos la cota del bloque dentro de este mientras y el mismo mientras tenemos la cota $O(n^2)$. Para sacar la cota resultante vemos que la mayor cota es la de $O(n^2)$ que involucra los ciclos anidados, por lo tanto la cota superior ajustada será:

$$\mathbf{O(n^2)}$$

```
Algoritmo FrecuenciaMinNumeros{
    Leer n                                      -- O(1)
    Dimension A[n]                              -- O(1)
    i = 1                                       -- O(1)
    Mientras i <= n                             -- O(n)
        Leer A[i]                               -- O(1)    O(N)
        i = i + 1                               -- O(1)
    FinMientras

    f = 0                                       -- O(1)
    i = 1                                       -- O(1)
    Mientras i <= n                             -- O(n)
        ntemp = A[i]                            -- O(1)
        j = 1                                   -- O(1)
        ftemp = 0                               -- O(1)              O(N²)
        Mientras j <= n                         -- O(n)
            si ntemp = A[j]
                ftemp = ftemp + 1               -- O(1)
            FinSi                                      O(1)  O(N)
            j = j + 1                           -- O(1)
        FinMientras                                              O(N²)

        si f < ftemp
            f = ftemp                           -- O(1)
            num = ntemp                         -- O(1)    O(1)
        FinSi

        i = 1 + 1                               -- O(1)
    FinMientras
    Escribir num
}
```

Código 09

```c
void search(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    for (int i = 0; i <= N - M; i++)
    {
        int j;

        for (j = 0; j < M; j++)
            if (txt[i + j] != pat[j])
                break;

        if (j == M)
            printf("Pattern found at index %d \n", i);
    }
```

Primeramente veamos el if que esta al final del primer for, este solo tiene una instrucción constante, por lo tanto su cota es O(1).

El for anidado tenemos que se repite M veces, de esta forma podemos decir que su cota superior ajustada es O(M).

Pasamos con el primero for, este se repite N-M veces, entonces podemos decir que la cota es O(N-M).

Para encontrar la cota de este bloque de código que comprende los for anidados hacemos a la multiplicación de las cotas, dando O(M*(N-M))

Por último, analizamos las primeras instrucciones, donde para sacar el tamaño de la cadena supones que será de cota O(M) y O(N) respectivamente.

Como la mayor de las cotas es la O(M*(N-M)) la cota superior ajustada resultante será:

O(M*(N-M))

```c
void search(char *pat, char* txt){

    int M = strlen(pat);                                -- O(M)
    int N = strlen(txt);                                -- O(N)

    for(int i = 0; i <= N-M; i++){                      -- O(M*(N-M))
        int j;
        for(j = 0; j < M; j++)                          -- O(M)
            if(txt[i + j] != pat[j])                            O(M)      O(M*(N-M))
                break;                                  -- O(1)

        if(j == M)
            printf("pattern found at index %d \n", i);  -- O(1)   O(1)
    }

}
```

Código 10

```
stack<int> sortStack(stack<int> &input)
{
    stack<int> tmpStack;

    while (!input.empty())
    {
        int tmp = input.top();
        input.pop();

        while (!tmpStack.empty() && tmpStack.top() > tmp)
        {
            input.push(tmpStack.top());
            tmpStack.pop();
        }

        tmpStack.push(tmp);
    }
    return tmpStack;
```

El ciclo de más adentro se repite n veces, donde n son las veces donde el elemento top de la pila es mayor al temporal, las instrucciones internas supondremos que son O(1).

El primer ciclo de igual forma se repite n veces, donde n son las veces que pasa sin que la pila esta vacía, de esta forma la cota es O(n).

Multiplicando ambas cotas tenemos que la cota superior ajustada para este algoritmo será:

$O(n^2)$

```
stack<int> sortStack(stack<int> &input){
    stack<int> tmpStack;                                    -- O(1)

    while(!input.empty()){                                  -- O(n)
        int tmp = input.pop();                              -- O(1)
        input.pop();                                        -- O(1)

        while(!tmpStack.empty() && tmpStack.top() > tmp){   -- O(n)
            input.push(tmpStack.top());                     -- O(1)
            tmpStack.pop();                                 -- O(1)
        }

        tmpStack.push(tmp);                                 -- O(1)
    }

    return tmpStack;
}
```

O(N)

O(N²)