Instituto Politécnico Nacional
Escuela Superior de Cómputo

Análisis de Algoritmos

Semestre: 2022-1

Grupo: 3CV11

Práctica 4

Fecha: 16 de noviembre de 2021

Práctica 4: Divide y Vencerás

Brayan Ramirez Benitez.
*bramirezb*1901@*alumno.ipn.mx*

Resumen: En esta práctica realizamos el análisis a priori y posteriori para dos algortimos que permitan rotar una imagen textual 90°, haciendo uso del paradigma de divide y vencerás. Para esto, los algoritmos se implementarán mediante el uso del lenguaje de programación C, además llevamos a cabo el análisis a posteriori mediante el uso de una Calculadora Grafica del sitio Web Desmos.

Palabras Clave: Rotar imagen, Divide, Vence, Combina, Matriz cuadrada.

1 Introducción

Cuando hablamos del término Divide y vencerás, nos referimos a la técnica de diseño de algoritmos, esta técnica es una de las mejores maneras de abordar un problema desde una perspectiva diferente, algo que todo programador deberia conocer, ya que son una parte fundamental de la concurrencia y los subprocesos múltiples.

Divide y vencerás tiende a resolver de manera satisfactoria problemas complejos, como la torre de Hanoi, una especie de rompecabezas matemático. Sin duda alguna, es un desafío resolver problemas de una complejidad alta para los cuales no tenemos una idea básica, pero mediante el uso de esta técnica podemos disminuir este esfuerzo, ya que dividimos el problema principal en subproblemas y luego los resolvemos de forma recursiva. Este tipo de algoritmos suelen ser más rápidos que otros algoritmos y utilizan eficientemente la memoria caché sin utilizar mucho espacio puesto que resuelve los subproblemas simples dentro de la memoria caché en lugar de acceder a la memoria principal más lenta, incluso es mejor que la técnica de fuerza bruta su contraparte.

Un algoritmo muy conocido que hace uso de Divide y vencerás es Búsqueda binaria, el cual es un algoritmo de búsqueda que funciona comparando el valor objetivo con el elemento intermedio existente en una matriz ordenada, si el valor no es igual, descarta la mitad de la matriz que no puede contener el objetivo y continuará la busqueda en la otra mitad, este proceso sigue repitiéndose hasta encontrar el objetivo o por el contrario determinar que el objetivo no se encuentra. El algoritmo de karatsuba para multiplicación támbien hace uso de está técnica, es uno de los algoritmos de multiplicación más rápidos de la época tradicional, inventado por Anatoly karatsuba a finales de 1960 y publicado en 1962. Multiplica dos números de n dígitos de tal manera que reduce esto a un solo dígito.

2 Conceptos Básicos

En esta sección mostraremos todos los conceptos necesarios para el desarrollo de la práctica.

Divide y vencerás: Es un paradigma de diseño de algoritmos que tiene como base la recursividad. Un algoritmo que hace uso de divide y vencerás funciona descomponiendo recursivamente un problema en varios subproblemas del mismo tipo, pero de menor tamaño, hasta que estos se vuelven suficientemente simples como para tener una solución trivial. Luego, las soluciones a estos subproblemas se combinan para obtener una solución al problema original.

Los algoritmos que hacen uso de divide y vencerás estan compuestos de tres partes:

1. Divide: Planteamos el problema de tal manera que podamos descomponerlo subproblemas de menor tamaño, pero del mismo tipo.

2. Vence: Resolvemos los subproblemas mediante recursividad. En caso de que los subproblemas son suficientemente pequeños se pueden resolver de manera trivial.

3. Combina: Combinamos las soluciones de los subproblemas para formar la solución al problema principal.

Primer Algoritmo: El algoritmo consiste en rotar una imagen 90 grados mediante una función implementada haciendo uso de divide y vencerás. La función recibe dos matrices A y C de tamaño nxn, una variable para las filas f, una variable para las columnas c, el tamaño del subarreglo n y el tamaño completo de la matriz v, la cual no regresa nada. Iniciamos preguntando por el caso base mediante un if si n es igual dos, si esto se cumple rotamos la matriz de 2x2 90 grados luego asignamos las posiciones del arreglo A al arreglo C de tal manera que la matriz en C sea el resultado de rotar 90 grados la matriz original A. Si la condición no se cumple, partimos la matriz en cuatro partes cada una de tamaño n/2 y llamamos recursivamente a la función RotarArreglo con cada una de las partes de la matriz, cuando termine de ejecutarse la matriz estará rotada 90 grados.

Algorithm 1 *RotarArreglo*

Require: Char A[][], Char C[][], Int f, Int c, Int n, Int v
Ensure: Arreglo rotado 90 grados

```
1: if n == 2 then
2:     C[f-2][c-2] = A[c-2][f-2]
3:     C[f-2][c-1] = A[c-1][f-2]
4:     C[f-1][c-2] = A[c-2][f-1]
5:     C[f-1][c-1] = A[c-1][f-1]
6:     if f ≤ (v/2) then
7:         C[f-2][c-2] = A[c-2][v-f+1]
8:         C[f-2][c-1] = A[c-1][v-f+1]
9:         C[f-1][c-2] = A[c-2][v-f]
10:        C[f-1][c-1] = A[c-1][v-f]
11:    else
12:        C[f-2][c-2] = A[c-2][v-f]
13:        C[f-2][c-1] = A[c-2][v-f+1]
14:        C[f-1][c-2] = A[c-1][v-f]
15:        C[f-1][c-1] = A[c-1][v-f+1]
16:    end if
17: else
18:    RotarArreglo(A,C,f-(n/2),c-(n/2),n/2)
19:    RotarArreglo(A,C,f-(n/2),c,n/2)
20:    RotarArreglo(A,C,f,c-(n/2),n/2)
21:    RotarArreglo(A,C,f,c,n/2)
22: end if
```

Segundo Algoritmo: El algoritmo consiste en rotar una imagen 90 grados mediante una función implementada de manera iterativa moviendo columna por columna. La función recibe una matriz A de tamaño nxn y el tamaño del arreglo n, la cual no regresa nada. Iniciamos declarando tres variables donde j e i nos serviran como iteradores mientras que z nos servira como índice para mover las posiciones de la matriz, luego un arreglo de carácteres C que nos servira como auxiliar para guardar el resultado de rotar la matriz. Iniciamos un ciclo for con i igual a 0, la condición i menor que n e incrementaremos i de uno en uno, luego iniciamos otro ciclo for con j igual a 0, la condición j menor que n e incrementaremos j de uno en uno, dentro de el asignamos a la matriz auxiliar C las posiciones que le corresponden de la matriz A rotada, finalmente decrementamos z de uno en uno.

Algorithm 2 *RotarFxF*

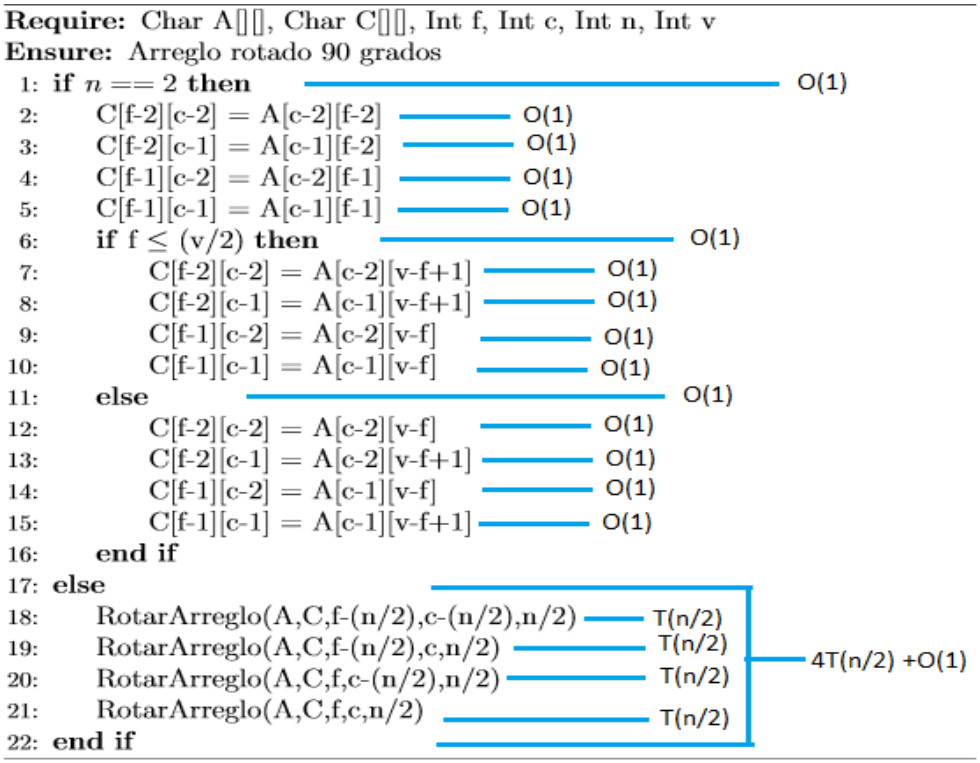
Require: char A[][], Int n
Ensure: Matriz rotada 90 grados

```
1: i
2: j
3: z = n − 1
4: for i = 0; i ≤ n i++; do
5:     for j = 0; j ≤ n j++; do
6:         C[i][j] = A[j][z]
7:     end for
8:     z − −
9: end for
```

3 Experimentación y Resultados

Análisis a priori para el primer algoritmo: Para el algoritmo **RotarArreglo** llevaremos a cabo el análisis mediante segmentos de código. Si observamos el pseudo-código podemos determinar que el peor caso y el mejor caso son iguales.

Se tiene que



De lo anterior tenemos que

$T(n) = 4T(n/2) + O(1)$

Sea $a = 4$, $b = 2$ y $f(n) = c$

Se tiene $n^{log_b a} = n^{log_2 4} = n^2$

$f(n) = c \in O(n^{log_b a - \epsilon}) = O \in n^{log_2 4 - \epsilon} = O \in (n^{2-\epsilon})$

Aplicando el teorema maestro, caso I

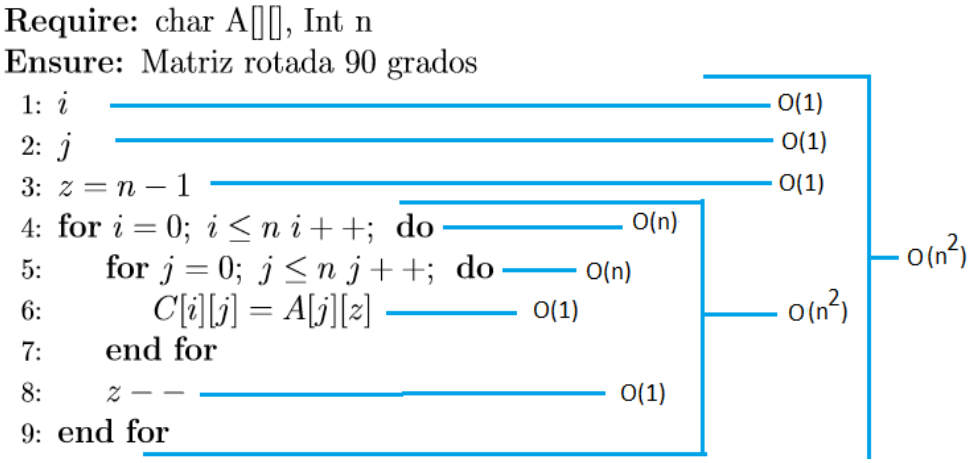
$T(n) = O(n^{log_b a}) = O(n^{log_2 4})$

$T(n) = O(n^2)$

$\therefore$ **RotarArreglo** $\in O(n^2)$

Análisis a priori para el segundo algoritmo: Para el algoritmo **RotarFxF** llevaremos a cabo el análisis mediante segmentos de código. Si observamos el pseudo-código podemos determinar que el peor caso y el mejor caso son iguales.

Se tiene que



De lo anterior concluimos que

La función **RotarFxF** $\in O(n^2)$

Análisis a posteriori para el primer algoritmo:

Comenzamos implementando el algoritmo como una función llamada **"RotarArreglo"** con base en el pseudo-código del **Algorithm 1 RotarArreglo**, mediante el Lenguaje de Programación en C.

```
void RotarArreglo(char A[][TAM], char C[][TAM], int f, int c, int n, int v){
    if(n == 2){//Vence
        int aux = A[f-2][c-2];//rotamos la matriz 2x2
        A[f-2][c-2] = A[f-2][c-1];
        A[f-2][c-1] = A[f-1][c-1];
        A[f-1][c-1] = A[f-1][c-2];
        A[f-1][c-2] = aux;

        if(f<=(v/2)){//Combina (Ordenamos el resultado de rotar la matriz de 2x2)
            C[f-2][c-2] = A[c-2][v-f+1];//ordena la mitad superior
            C[f-2][c-1] = A[c-1][v-f+1];
            C[f-1][c-2] = A[c-2][v-f];
            C[f-1][c-1] = A[c-1][v-f];
        }else{
            C[f-2][c-2] = A[c-2][v-f];//ordena la mitad inferior
            C[f-2][c-1] = A[c-2][v-f+1];
            C[f-1][c-2] = A[c-1][v-f];
            C[f-1][c-1] = A[c-1][v-f+1];
        }
    }else{//Divide
        RotarArreglo(A,C,f-(n/2),c-(n/2),n/2,v);
        RotarArreglo(A,C,f-(n/2),c,n/2,v);
        RotarArreglo(A,C,f,c-(n/2),n/2,v);
        RotarArreglo(A,C,f,c,n/2,v);
    }
}
```

Figura 1. Código para el **primer algoritmo** en lenguaje C.

Además, utilizaremos la función **Principal** la cual nos servirá para ejecutar el algoritmo y obtener los datos para el análisis, donde enviaremos los resultados a un archivo.

Para realizar el análisis la función **RotarArreglo (Figura 1)** debe recibir un parámetro por referencia **(ct)** que obtendrá el número ejecuciones de cada línea.

```
void RotarArreglo(char A[][TAM], char C[][TAM], int f, int c, int n, int *ct, int v){
    if(n == 2){
        (*ct)++;
        int aux = A[f-2][c-2];  (*ct)++;
        A[f-2][c-2] = A[f-2][c-1];(*ct)++;
        A[f-2][c-1] = A[f-1][c-1];(*ct)++;
        A[f-1][c-1] = A[f-1][c-2];(*ct)++;
        A[f-1][c-2] = aux;(*ct)++;
        if(f<=(v/2)){
            (*ct)++;
            C[f-2][c-2] = A[c-2][v-f+1];(*ct)++;
            C[f-2][c-1] = A[c-1][v-f+1];(*ct)++;
            C[f-1][c-2] = A[c-2][v-f];(*ct)++;
            C[f-1][c-1] = A[c-1][v-f];  (*ct)++;
        }else{
            C[f-2][c-2] = A[c-2][v-f];(*ct)++;
            C[f-2][c-1] = A[c-2][v-f+1];(*ct)++;
            C[f-1][c-2] = A[c-1][v-f];(*ct)++;
            C[f-1][c-1] = A[c-1][v-f+1];(*ct)++;
        }(*ct)++;
    }else{
        RotarArreglo(A,C,f-(n/2),c-(n/2),n/2, ct,v);(*ct)++;
        RotarArreglo(A,C,f-(n/2),c,n/2,ct,v);(*ct)++;
        RotarArreglo(A,C,f,c-(n/2),n/2,ct,v);(*ct)++;
        RotarArreglo(A,C,f,c,n/2,ct,v);(*ct)++;
    }
    (*ct)++;
}
```

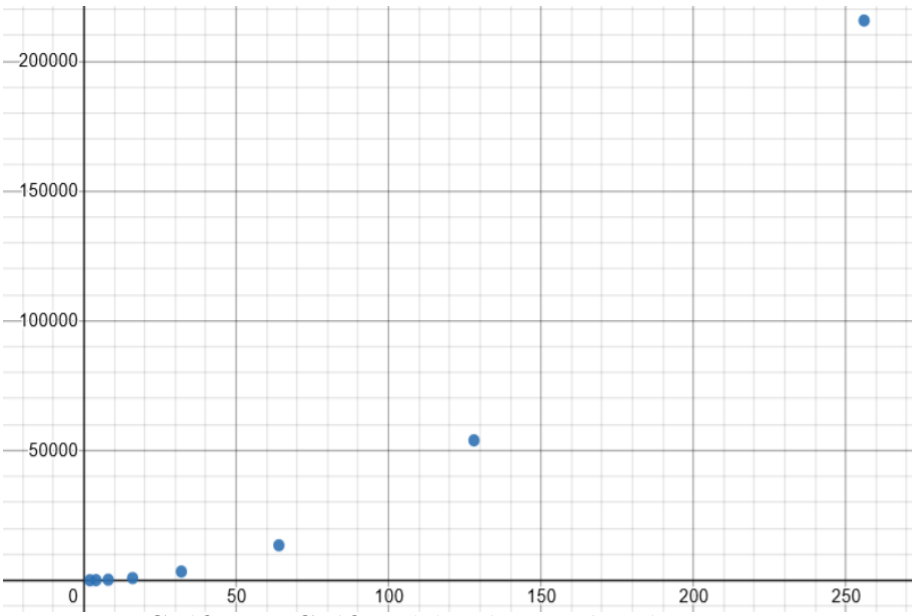
Figura 2. Código para el **primer algoritmo** con el parámetro **ct**.

Del análisis a priori sabemos qel peor caso y el mejor caso son iguales. Para obetener los puntos de muestra alimentamos el algoritmo con valores de n potencias de dos, es decir, con n = 2, 4, 8, 16, 32, etc. El resultado de esta ejecución arroja los siguientes valores.

n	Ct
2	11
4	51
8	209
16	841
32	3369
64	13481
128	53929
256	215721

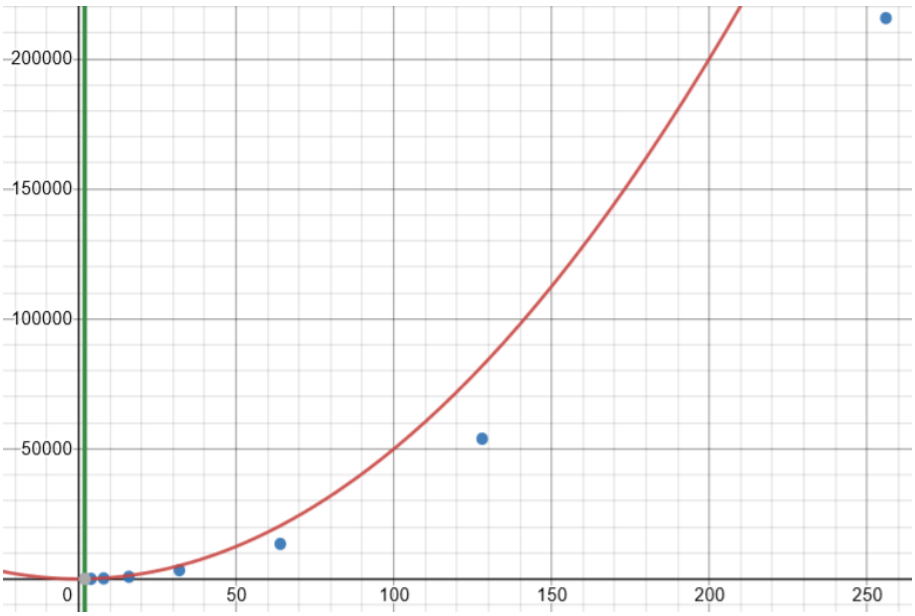
Tabla 1. Valores de la ejecución para el **primer algoritmo**.

Copiamos los valores que resultaron de la ejecución del programa y los graficamos mediante la calculadora Gráfica Desmos, observamos que es cuadratica $O(n^2)$.



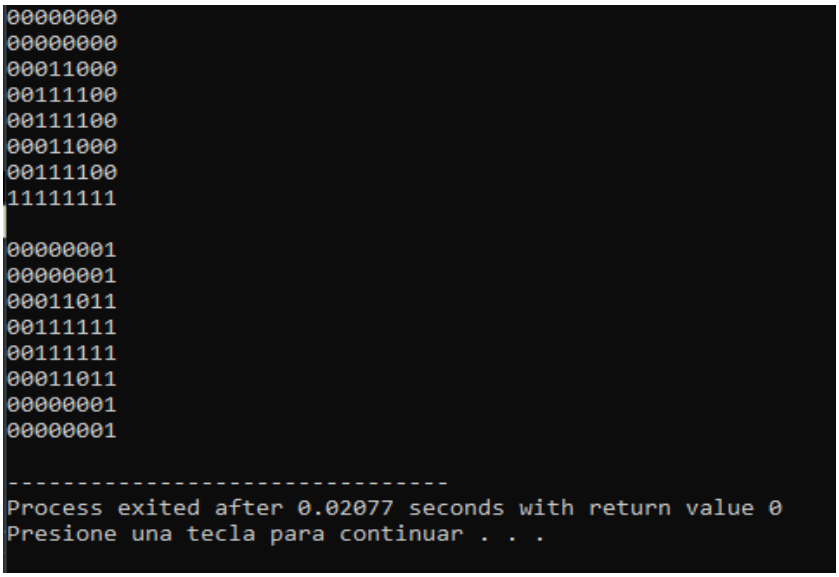
Gráfica 1. Gráfica del **primer algoritmo**.

Ahora proponemos la función $\mathbf{g(n)} = 5(n^2) \in O(n^2)$ con $n_o = 2$ para la cota superior.

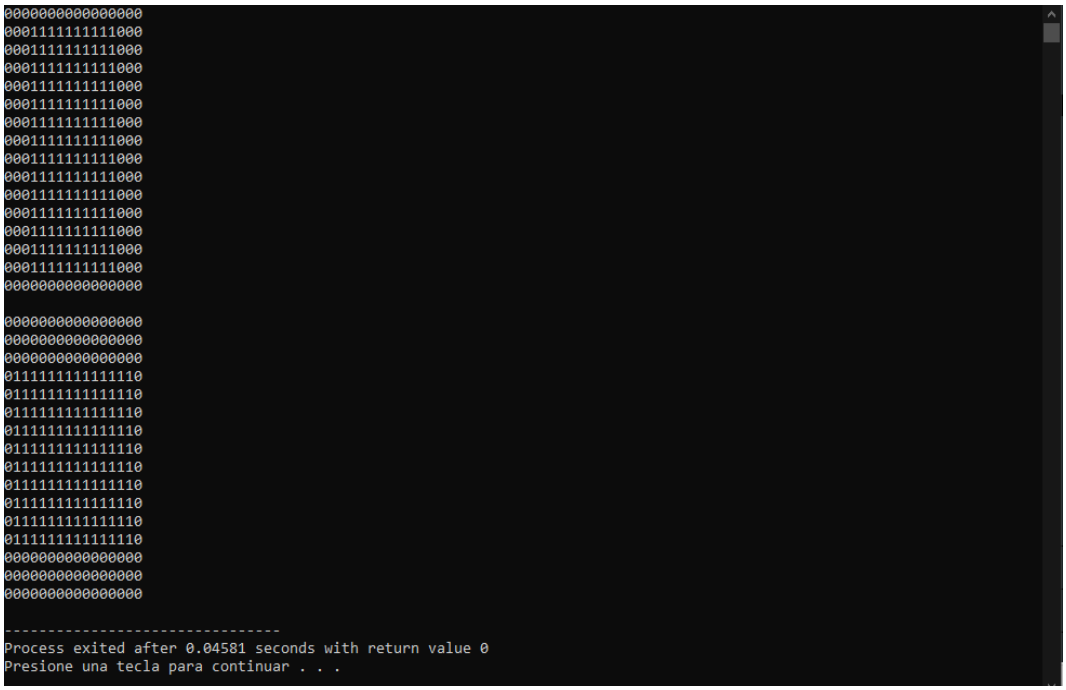


Gráfica 2. Gráfica del **primer algoritmo** con la función propuesta.

Ejemplos:


00000000
00000000
00011000
00111100
00111100
00011000
00111100
11111111
00000001
00000001
00011011
00111111
00111111
00011011
00000001
00000001

Process exited after 0.02077 seconds with return value 0
Presione una tecla para continuar . . .


0000000000000000
0001111111111000
0001111111111000
0001111111111000
0001111111111000
0001111111111000
0001111111111000
0001111111111000
0001111111111000
0001111111111000
0001111111111000
0001111111111000
0001111111111000
0001111111111000
0001111111111000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0111111111111110
0111111111111110
0111111111111110
0111111111111110
0111111111111110
0111111111111110
0111111111111110
0111111111111110
0111111111111110
0000000000000000
0000000000000000
0000000000000000

Process exited after 0.04581 seconds with return value 0
Presione una tecla para continuar . . .

Análisis a posteriori para el segundo algoritmo:

Comenzamos implementando el algoritmo como una función llamada **"RotarFxF"** con base en el pseudo-código del **Algorithm 2 RotarFxF**, mediante el Lenguaje de Programación en C.

```c
void RotarFxF(char A[][TAM], int n){
    int i,j,z = n-1;
    char C[n][n];

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

Figura 3. Código para el **segundo algoritmo** en lenguaje C.

Además, utilizaremos la función **Principal** para el **segundo algoritmo** la cual nos servirá para ejecutar y obtener los datos para el análisis, donde enviaremos los resultados a un archivo.

Para realizar el análisis la función **RotarFxF (Figura 3)** debe recibir un parámetro por referencia **(ct)** que obtendrá el número ejecuciones de cada línea.

```c
void RotarFxF(char A[][TAM], int n, int *ct){
    int i,j,z = n-1;(*ct)++;(*ct)++;(*ct)++;
    char C[n][n];(*ct)++;

    (*ct)++;
    for(i = 0 ; i<n ; i++){
        (*ct)++;(*ct)++;
        for(j = 0 ; j<n ; j++){
            (*ct)++;
            C[i][j] = A[j][z];(*ct)++;
        }
        (*ct)++;
        z--;(*ct)++;
    }
    (*ct)++;
}
```

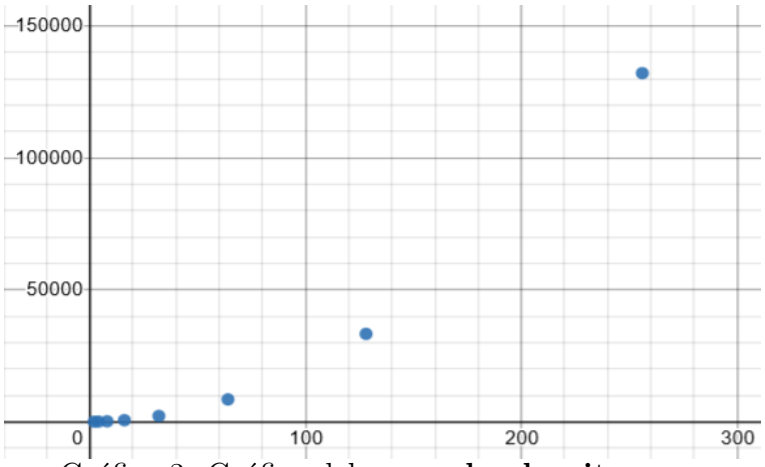
Figura 4. Código para el **segundo algoritmo** con el parámetro **ct**.

De manera similar para el análisis a priori sabemos qel peor caso y el mejor caso son iguales. De manera similar para obetener los puntos de muestra alimentamos el algoritmo con valores de n potencias de dos, es decir, con n = 2, 4, 8, 16, 32, etc. El resultado de esta ejecución arroja los siguientes valores.

n	Ct
2	22
4	54
8	166
16	582
32	2182
64	8454
128	33286
256	132102

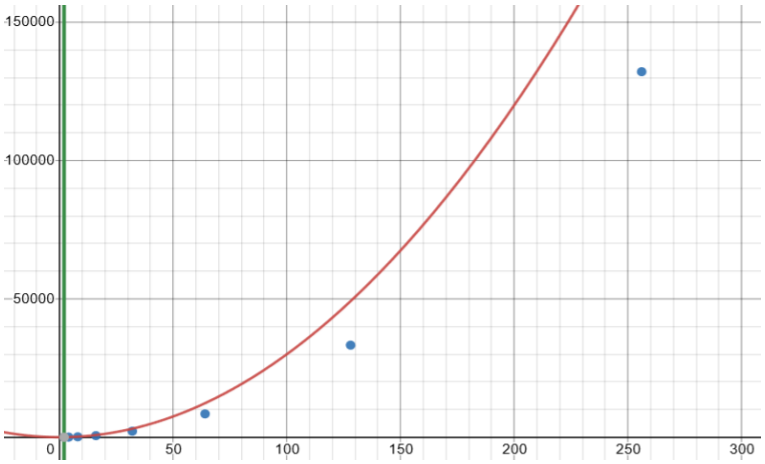
Tabla 2. Valores de la ejecución para el **segundo algoritmo**.

Copiamos los valores que resultaron de la ejecución del programa y los graficamos mediante la calculadora Gráfica Desmos, observamos que es cuadrática $O(n^2)$.



Gráfica 3. Gráfica del **segundo algoritmo**.

Ahora proponemos la función $\mathbf{g(n)} = 3(n^2) + 20 \in O(n^2)$ con $n_o = 2$ para la cota superior.



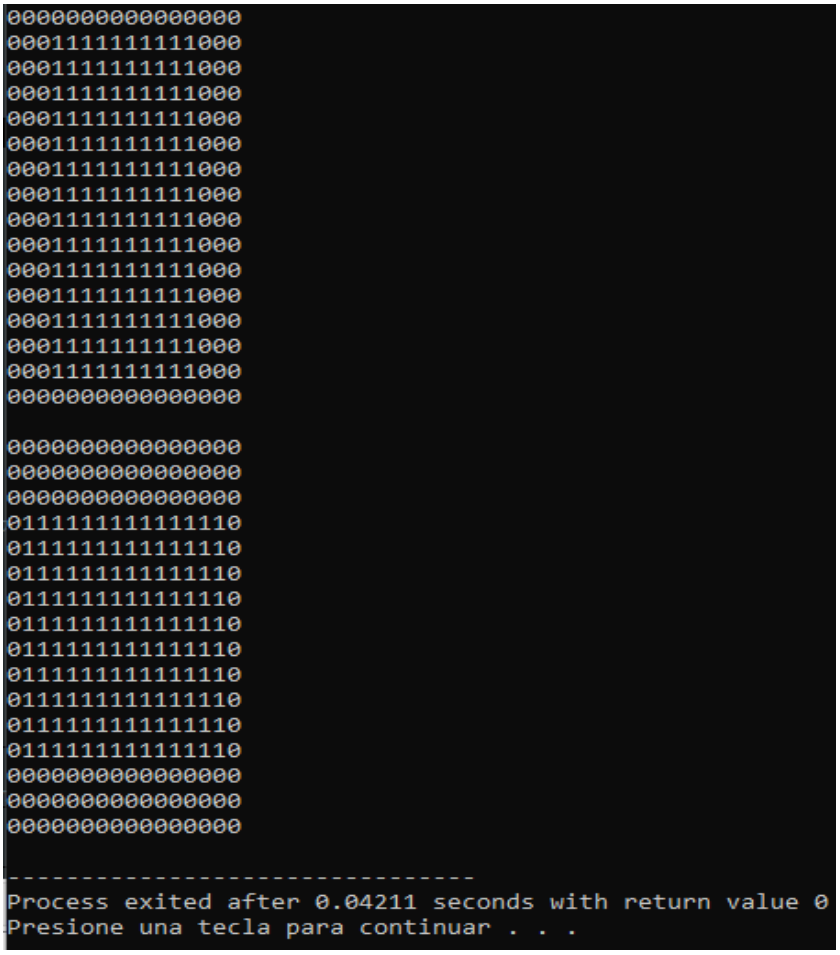
Gráfica 4. Gráfica del **segundo algoritmo** con la función propuesta.

Ejemplos:

```
00000000
00000000
00011000
00111100
00111100
00011000
00111100
11111111

00000001
00000001
00011011
00111111
00111111
00011011
00000001
00000001

--------------------------------
Process exited after 0.03389 seconds with return value 0
Presione una tecla para continuar . . .
```



Características del equipo de cómputo:

- Procesador: Ryzen 5 1400
- Tarjeta gráfica: GTX 1050 ti de la marca PNY
- Tarjeta Madre: Asus A320M-K
- Memoria RAM: 8 Gb
- Disco duro: Disco Duro Interno 1tb Seagate

4 Conclusiones

Conclusiones generales: Durante el desarrollo de esta práctica ocurrieron bastantes problemas para la implementación del algoritmo puesto que no lograba encontrar una solución para el problema planteado haciendo uso de divide y vencerás, además encontrar la ecuación de recurrencia de igual manera fue bastante complicado determinarla. Por otro lado, para encontrar la otra solución al problema me parecio bastante sencilla.

Conclusiones individuales (Brayan Ramirez Benitez): Divide y venceás es una estrategia bastante utilizada para resolver problemas, es posible generar el algoritmo para el problema planteado en la práctica haciendo uso de está técnica de manera eficiente, conocer como aplicar esto nos permite tener otra perspectiva para resolver problemas, además nos ayuda a ahorrar bastantes líneas de código y tiempo para desarrollar algún algoritmo, en cuanto a la complejidad de este tipo de algoritmos va a depender de las descomposiciones que realizemos y las sucesivas llamadas recursivas a los subproblemas, por ultimo es importante mencionar que no todos los algoritmos pueden resolverse mediante está técnica, lo cual es una desventaja.

Brayan Ramirez Benitez.

5 Anexo

Problema 1: Probar mediante sustitución hacia atrás que $T(n) \in \Theta(nlogn)$
Se tiene que T(n) es:

$$T(n) = \begin{cases} c, & n = 1 \\ 2T(n/2) + cn, & n > 1 \end{cases}$$

Resolviendo T(n) mediante sustitución hacia atrás, se tiene:

$T(n) = 2T(n/2) + cn$

Sea $n = 2^k$ $(k = logn)$, se tiene:

$$T(2^k) = \begin{cases} c, & k = 0 \\ 2T(2^{k-1}) + c2^k, & k > 0 \end{cases}$$

$T(2^k) = 2T(2^{k-1}) + c2^k$

$= 2[2T(2^{k-2}) + c2^{k-1}] + c2^k$

$= 4T(2^{k-2}) + 2c2^k$

$= 4[2T(2^{k-3}) + c2^{k-2}] + 2c2^k$

$= 8T(2^{k-3}) + 3c2^k$

$= 8[2T(2^{k-4}) + c2^{k-3}] + 3c2^k$

$= 16T(2^{k-4}) + 4c2^k$

$\vdots$

$(i) = 2^iT(2^{k-i}) + i \cdot c2^k$

$\vdots$

$k - i = 0 \Rightarrow k = i$

$= 2^kT(2^0) + kc2^k$

$= 2^kT(2^0) + kc2^k$

$= 2^kc + kc2^k$

$= c2^k(1 + k)$

sustituyendo el valor de k

$= cn(1 + logn)$

$\therefore T(n) \in \Theta(nlogn)$

Problema 2: Utilizando decremento por uno, pruebe que $T(n) \in O(n^2)$

Donde:

$T(n) = T(1) + T(n-1) + cn$

$T(n) = T(n-1) + cn + c$

Aplicando decremento por 1 con $f(n) = cn + c$, se tiene:

$$\sum_{j=1}^{n} f(j) = \sum_{j=1}^{n} c(j) + c = c\sum_{j=1}^{n} j + 1$$

$$c[\sum_{j=1}^{n} j + \sum_{j=1}^{n} 1] = c[\frac{n(n+1)}{2} + n]$$

$\therefore T(n) \in O(n^2)$

Problema 3: Utilizando el teorema maestro, pruebe que $T(n) \in \Omega(nlogn)$
Donde:

$T(n) = 2T(n/2) + \Theta(n)$

Sea

$a = 2$, $b = 2$ y $f(n) = cn$

Se tiene

$n^{log_b a} = n^{log_2 2} = n$

Puesto que

$f(n) = cn \in \Omega(n^{log_b a}) = \Omega \in n^{log_2 2} = \Omega \in (n)$

Aplicando el teorema maestro, caso II
Se tiene

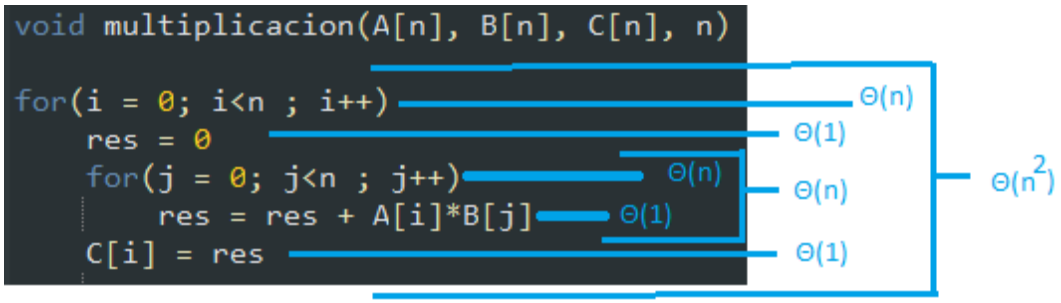
$T(n) = \Omega(n^{log_b a} \cdot log_b n) = \Omega(n^{log_2 2} \cdot log_2 n)$

$T(n) = \Omega(n \cdot logn)$

$\therefore T(n) \in \Omega(nlogn)$

Problema 4: Multiplicacíon usual de números tiene orden de complejidad $\Theta(n^2)$

La multiplicacíon usual de manera general tiene la siguiente forma:



$\therefore multiplicacion \in \Theta(n^2)$

Problema 5: Mostrar que $T(n) \in \Theta(n^2)$
Donde

$T(n) = 4T(n/2) + \Theta(n)$

Sea

$a = 4$, $b = 2$ y $f(n) = cn$

Se tiene

$n^{log_b a} = n^{log_2 4} = n^2$

$f(n) = cn \in \Theta(n^{log_b a - \epsilon}) = \Theta \in n^{log_2 4 - \epsilon} = \Theta \in (n^{2-\epsilon})$

Aplicando el teorema maestro, caso I
Se tiene

$T(n) = \Theta(n^{log_b a}) = \Theta(n^{log_2 4})$

$T(n) = \Theta(n^2)$

$\therefore T(n) \in \Theta(n^2)$

Problema 6: Mostrar que $T(n) \in \Theta(n^{log3})$
Donde

$T(n) = 3T(n/2) + \Theta(n)$

Sea

$a = 3$, $b = 2$ y $f(n) = cn$

Se tiene

$n^{log_b a} = n^{log_2 3} = n^{1.58}$

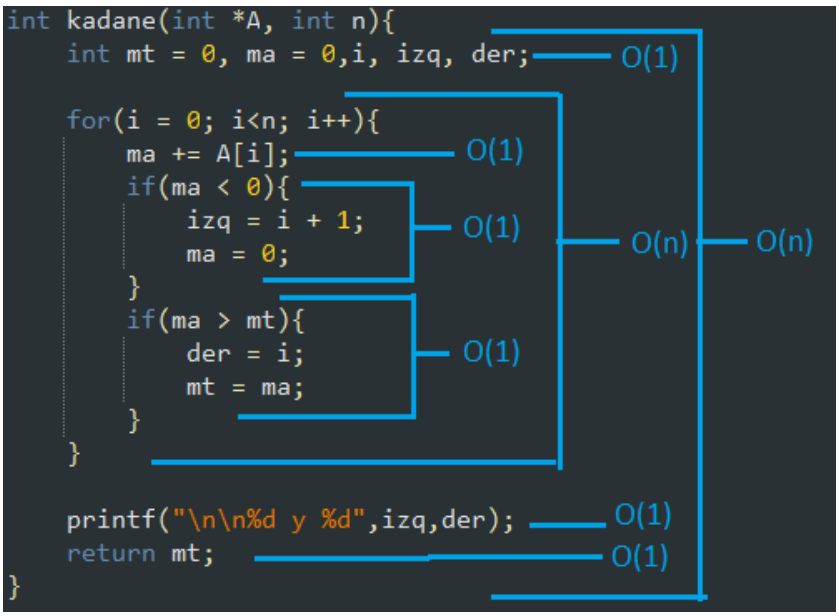
$f(n) = cn \in \Theta(n^{log_b a - \epsilon}) = \Theta \in n^{log_2 3 - \epsilon} = \Theta \in (n^{1.58-\epsilon})$

Aplicando el teorema maestro, caso I
Se tiene

$T(n) = \Theta(n^{log_b a}) = \Theta(n^{log_2 3})$

$\therefore T(n) \in \Theta(n^{log_2 3})$

Problema 7: Implementar la idea de este último algoritmo (Kadane) para encontrar el máximo subarreglo y realizar sus análisis a priori. El algoritmo además, debe de regresar los índices en donde se encuentra el máximo subarreglo.



$\therefore Kadane \in O(n)$

6 Bibliografía

References

[1] Aguilar, I. (2019) *Introducción al análisis de algoritmos.* PDF. Recuperado de http://ri.uaemex.mx/bitstream /handle/20.500.11799/105198/Libro%20Complejidad.pdf? sequence=1&isAllowed=y

[2] Cormen, E. A. (2009) *Introduction to Algorithms.* 3Rd Ed. (3.a ed., Vol. 3). Phi.

[3] Rodriguez, E.A. (2018, 7 de marzo) *Divide y vencerás* PDF. Recuperado de https://www.tamps.cinvestav.mx/ / ertello/algorithms/sesion12.pdf

[4] UMA. (2014, 7 de septiembre) *Técnicas de diseño de algoritmos.* PDF. Recuperado de http://www.lcc.uma.es/ av/Libro/CAP3.pdf