Instituto Politécnico Nacional

Escuela Superior de Cómputo

Ingeniería en Inteligencia Artificial, Análisis y Diseño de Algoritmos
Sem: 2024-1, 3BV1, Práctica 4, 22 de noviembre de 2023

Práctica 4: Divide y Vencerás.

Catonga Tecla Daniel Isaí 1, Olguin Castillo Victor Manuel 2.

*daniel*9513*importantes*@*gmail.com*$_1$, *manuelevansipn*@*gmail.com*$_2$

Resumen:

Se propuso un ejercicio, el cual consiste en un programa el cual pueda cargar una imagen que tenga tamaño cuadrado potencia de 2, y mediante una función de fuerza bruta y otra función utilizando divide y vencerás se podrá rotar 90 grados usando cualquiera de las dos funciones. Ocupamos el lenguaje de C++ y ocupamos imágenes con formato.BMP ya que C++ permite trabajar con flujo de archivos en binario y podemos acceder a el mapa de bits de una imagen.BMP , para poder pasar los datos de los píxeles a una matriz de píxeles para crear nuestras dos funciones. Se obtuvo el análisis a priori de la función "fuerza bruta"mediante análisis a bloques y el análisis a priori de la función "Divide y conquistarás"se realizó mediante método maestro, el análisis a posteriori de la función "fuerza frutaz la función "Divide y conquistaras"se hizo comparando el ancho de una imagen con el numero de pasos, con estos datos obtenidos se graficaron con la página web DESMOS para poder ver de manera gráfica el comportamiento de cada algoritmo.

Palabras Clave: Algoritmo, Big O, C++, Divide y Vencerás

1. Introducción

Un algoritmo es un conjunto ordenado y finito de operaciones que permite hallar la solución de un problema [1]. Los algoritmos son de suma importancia, estos ayudan a la resolución de problemas muy grandes o complejos como problemas matemáticos y científicos, por lo tanto, los algoritmos son fundamentales en varias áreas de las ciencias o ingeniería.

El análisis de algoritmos es otro parte fundamental por lo que los algoritmos tienen que ser los más eficientes posibles en complejidad temporal y complejidad espacial. En cuestión de complejidad temporal algunos algoritmos pueden requerir años en resolver un solo problema, por lo que, no es suficiente para la práctica. , y se opta por otros algoritmos que resuelvan el problema que presenten de una forma más eficaz. Por otra parte, una maquina no cuenta con recursos infinitos de memoria o espacio de almacenamiento, por lo que, existen algoritmos que ocupan mucho espacio de memoria y esto puede ocasionar problemas para equipos que no cuentan con el espacio suficiente, entonces se tiene que analizar también lo que es la complejidad espacial para tratar de ocupar la menor cantidad de memoria posible. Ambas complejidades son importantes, pero es más importante la complejidad temporal ya que las empresas pueden comprar más memorias de almacenamiento, pero no pueden comprar tiempo, por lo que algunas empresas optan sacrificar complejidad espacial por una complejidad temporal más eficiente.

"Divide y vencerás", es un paradigma de diseño de algoritmos que resuelve problemas dividiéndolos en subproblemas más pequeños y manejables [**div**]. Estos subproblemas se resuelven de manera independiente y se combinan para obtener la solución final. Ejemplos incluyen Merge Sort y QuickSort para ordenar, Binary Search para búsqueda eficiente, etc. Esta estrategia mejora la eficiencia y facilita la implementación modular de soluciones en algoritmos y problemas de programación.

La rotación de imágenes podría parecer una tarea sencilla. Este proceso, aparentemente simple, se convierte en una tarea compleja cuando se analiza desde la programación mediante el uso de algoritmos. Se busca que un algoritmo sea eficiente, ya que en la práctica, al desarrollar una aplicación, se requiere que la rotación de una imagen no tome minutos.

La optimización de algoritmos de rotación es esencial para garantizar una experiencia fluida y eficaz en el manejo de imágenes digitales cotidianas. En la práctica se muestran unos algoritmos funcionales para la rotación de imagenes cuadradas de $2^k \cdot 2^k$.

2. Conceptos Basicos

- **Algoritmo**. Un algoritmo es una secuencia de pasos lógicos que son precisos, ordenados y finitos que se ocupan para resolver un problema deseado [2].

- **Análisis de algoritmos** Es un proceso de evaluación donde conoceremos el rendimiento y la eficiencia de un algoritmo. Se evaluará el consumo de tiempo y de recursos computacionales que requiere el algoritmo para ser ejecutado con diversos datos de entrada, y esto determinará su complejidad [2].

- **Cota superior asintótica**. Es una función que delimita por la parte superior a otra función a medida que la entrada de la función delimitada crece.

- **Cota inferior asintótica**. Es una función que delimita por la parte inferior a otra función a medida que la entrada de la función delimitada crece.

- **Notacion O**. Esta notación se ocupa para describir la complejidad de un algoritmo definiendo una cota superior asintótica en el peor caso de ejecución de un algoritmo [2].

$$O(g(n)) = \{f : \mathbb{N} \to \mathbb{R}^+ \mid \exists\ c \text{ constante positiva y } n_0 \notin \mathbb{N} :$$
$$f(n) \leq cg(n),\ \forall\ n \geq n_0\}$$

- **Notacion Tetha "Θ"** Esta notación se ocupa para describir la complejidad de un algoritmo definiendo una cota superior y una cota inferior, esto nos da una idea más precisa del comportamiento y complejidad del mismo [2].

$$\Theta(g(n) = \{f : \mathbb{N} \to \mathbb{R}^+ \mid \exists\, c_1, c_2 \text{ constantes positivas}, n_0\ :$$
$$0 < c_1 g(n) \leq f(n) \leq c_2 g(n), \forall\, n \geq n_0\}$$

- **Notacion Omega "Ω"** Esta notación se ocupa para describir la complejidad de un algoritmo definiendo una cota inferior asintótica en el mejor de ejecución de un algoritmo [2].

$$\Omega(g(n)) = \{f : \mathbb{N} \to \mathbb{R}^+ \mid \exists\, c \text{ constante positiva y } n_0\ :$$
$$0 < cg(n) \leq f(n), \forall\, n \geq n_0\}$$

- **Analisis a posteriori**. Es una evaluación que se realiza de forma empírica, donde los resultados se obtienen con la ejecución del algoritmo y la medición del tiempo y recursos computacionales que requiere [2].
- **Analisis a priori**. Es una evaluacion que se realiza de forma teorica, donde los resultados se pueden obtener con el conteo de operaciones y/o analisis matematico que en base a sus formulas se obtiene su complegidad algoritmica [2].
- **Funciones recursivas**. En la programacion las funciones recursivas son aquellas que durante su proceso se invocan asi mismas.
- **Divide y vencerás**. Es un paradigma de diseño de algoritmos que resuelve problemas dividiéndolos en subproblemas más pequeños y manejables. Estos subproblemas se resuelven de manera independiente y se combinan para obtener la solución final [**div**].
- **Formato .BMP**. El formato WindowsBitMap es muy simple y sin compresion, consiste en una cabecera (54 bytes) seguida por por los valores de cada pixel, comenzando por la lina inferior de la imagen, pixel a pixel de izquierda a derecha.
- **Pixel** . Es la unidad mínima de cualquier imagen digital y esta compuesto por 3 colores (RGB) que determina su intensidad y color.
- **Pseudocódigo Rotar Imagen con Fuerza Bruta**. El algoritmo rota una matriz que contiene los valores equivalentes al RGB, la rotación se hace con el recorrido por partes de i,j a su equivalente rotado que es i',j' que será la nueva posición de i,j.

```
Rotar(Matriz[n][n]):
n = numero de filas de la matriz
for i = 0 to i = n/2 do:
    for j = i to j = n - i - 1 do:
        temp = matrix[i][j];
        matrix[i][j] = matrix[n - 1 - j][i]
        matrix[n - 1 - j][i] = matrix[n - 1 - i][n - 1 - j]
        matrix[n - 1 - i][n - 1 - j] = matrix[j][n - 1 - i]
        matrix[j][n - 1 - i] = temp
```

- **Pseudocódigo Rotar Imagen con Divide y Vencerás**. El algoritmo rota una matriz que contiene los valores equivalentes al RGB, la rotación se hace a tráves de la estrategia de divide y vencerás en la cual la matriz original se va dividiendo en Sub-Matrices.

```
RotarDivideVencer(Matriz[n][n],n):
if n <= 0 retorna
matrizA [n/2][n/2]
matrizB [n/2][n/2]
matrizC [n/2][n/2]
matrizD [n/2][n/2]
for i = 0 to i = n/2 - 1 do:
    for j = 0 to j = n/2 - 1 do:
        matrizA[i][j] = matrix[i][j];
        matrizB[i][j] = matrix[i][j + size/2];
        matrizC[i][j] = matrix[i + size/2][j];
        matrizD[i][j] = matrix[i + size/2][j + size/2];
RotarDivideVencer(matrizA,n/2)
RotarDivideVencer(matrizB,n/2)
RotarDivideVencer(matrizC,n/2)
RotarDivideVencer(matrizD,n/2)
for i = 0 to i = n/2 - 1 do:
    for j = 0 to j = n/2 - 1 do:
        matrix[i][j] = c[i][j];
        matrix[i][j + size/2] = a[i][j];
        matrix[i + size/2][j] = d[i][j];
        matrix[i + size/2][j + size/2] = b[i][j];
```

3. Experimentación y Resultados

Análisis a priori de rotación de imágenes con divide y vencerás. El análisis a priori del algoritmo se basa con el pseudocódigo del algoritmo que se muestra a continuación, titulado RotarDivideVencer".

```
RotarDivideVencer(Matriz[n][n],n):
01.  if n <= 1 return
02.  matrizA[n/2][n/2]
03.  matrizB[n/2][n/2]
04.  matrizC[n/2][n/2]
05.  matrizD[n/2][n/2]
06.  for i = 0 to i = n/2 - 1 do:
07.    for j = 0 to j = n/2 - 1 do:
08.         matrizA[i][j] = matriz[i][j]
09.         matrizB[i][j] = matriz[i][j + n/2]
10.         matrizC[i][j] = matriz[i + n/2][j]
11.         matrizD[i][j] = matriz[i + n/2][j + n/2]
12.  RotarDivideVencer(matrizA,n/2)
13.  RotarDivideVencer(matrizB,n/2)
14.  RotarDivideVencer(matrizC,n/2)
15.  RotarDivideVencer(matrizD,n/2)
16.  for i = 0 to i = n/2 - 1 do:
17.    for j = 0 to j = n/2 - 1 do:
18.         matriz[i][j] = matrizC[i][j];
19.         matriz[i][j + n/2] = matrizA[i][j]
20.         matriz[i + n/2][j] = matrizD[i][j];
21.         matriz[i + n/2][j + n/2] = matrizB[i][j]
```

El algoritmo posee una ecuación de recurrencia, la cual se expresa como $T(n) = 4T(\frac{n}{2}) + n^2$. Esta ecuación se deriva a partir del cálculo de la complejidad por bloques de la Imagen 1.

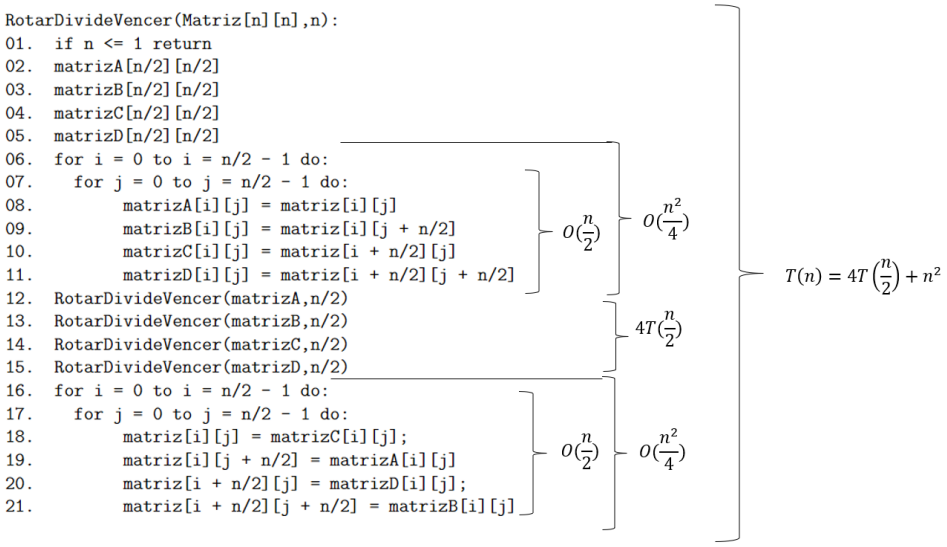

Imagen 1. Ecuación de Recurrencia.

Por lo tanto, la solución de la ecuación de recurrencia se obtiene por método maestro.

Solución:

$T(n) = 4T(\frac{n}{3}) + n^2$

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

Se tiene: $O(n^{log_b a}) = O(n^{log_2 4}) = O(n^2)$

por el caso II:

(II) si $f(n) = \theta(n^{log_b a})$, entonces $T(n) = \theta(n^{log_b a} \cdot log_b n)$

El algoritmo tiene complejidad:

$$\therefore T(n) \in \theta(n^2 \cdot log_2 n)$$

Análisis a posteriori del algoritmo de rotación de imágenes con divide y vencerás. El algoritmo en ejecución se muestra en la imagen 2, y el número de pasos se muestra en la tabla 1. Los datos muestran un aumento en la complejidad a medida que n crece (n es la altura o anchura de la imagen). Para este algoritmo no hay un peor ni mejor caso.

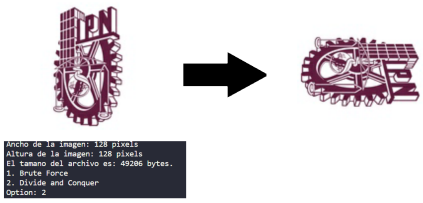

Imagen 2. Ejecución algoritmo de rotación de la imagen con divide y vencerás.

Valor de n	**# de pasos**
4	213
8	1049
16	4897
32	22257
64	99473
128	439249
256	1921617
⋮	⋮

Tabla 1. Datos de algoritmo de rotación de imagen con divide y vencerás.

Los datos de ejecución del algoritmo se muestra en la imagen 3. El algoritmo muestra crecimiento muy rápido con respecto a la anchura o altura de la imagen dado que es cuadrada.

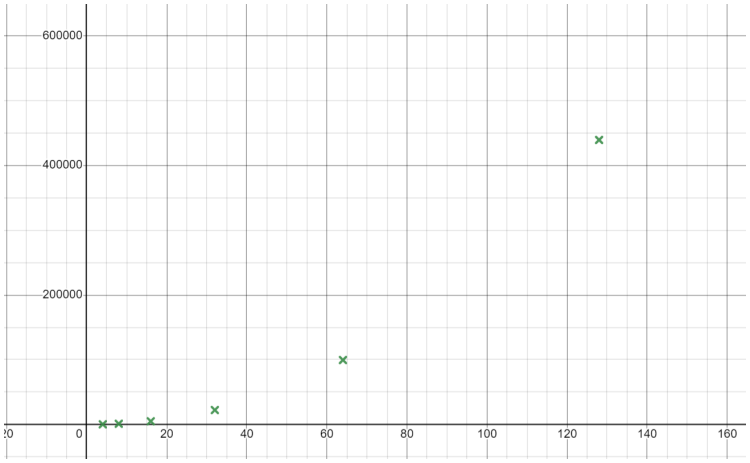

Imagen 3. Gráfica del comportamiento del algoritmo de rotación de imagen. con divide y vencerás.

En la Imagen 4 se muestra la función tal que $g(n) = 8{,}5n^2log_2(n)$ acota por arriba al algoritmo. El ajuste asíntotico es para cuando $n_0 \geq 4$. En la imagen 5 se muestra que $f(n)$ esta delimitado por $g(n)$ cuando $n \geq n_0$.

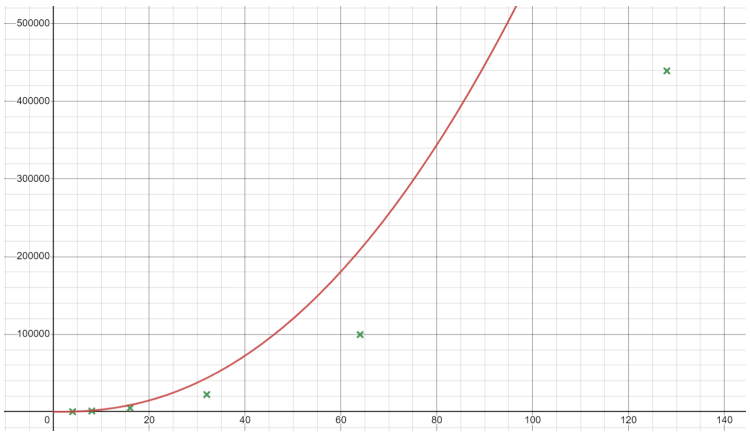

Imagen 4. Gráfica con acotación para $f(n)$.

Imagen 5. Gráfica para valor n_0.

Por lo tanto, en el análisis a posteriori se muestra que el algoritmo es:

$$\therefore T(n) \in \theta(n^2log_2n)$$

Análisis a priori de rotacionde imágenes con fuerza bruta. El análisis a priori del algoritmos se basa con el pseudo código del algoritmo que se muestra a continuación.

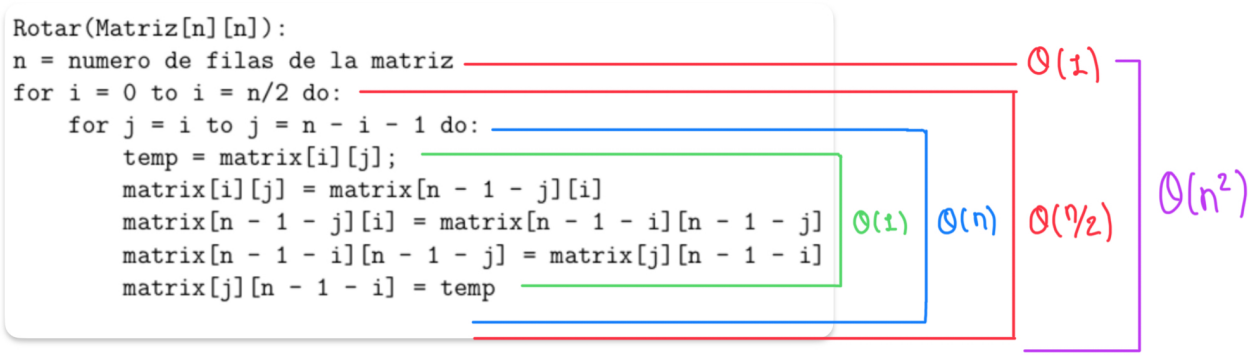

Imagen 6. Análisis a priori del algoritmo con fuerza bruta n_0.

Con el análisis a bloques se deduce que el algoritmo tiene una complejidad $\emptyset(n^2)$

Análisis a posteriori del algoritmo de rotación de imágenes con fuerza bruta. El algoritmo en ejecución se muestra en la imagen 7, y el número de pasos se muestra en la tabla 2. Los datos muestran un aumento de pasos a medida que crece n (alto o ancho de una imagen)

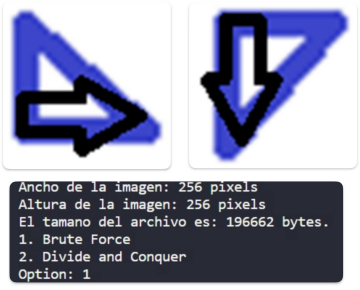

Imagen 7. Ejecución algoritmo de rotación de la imagen con fuerza bruta.

Valor de n	**# de pasos**
4	33
8	111
16	411
32	1587
64	6243
128	24771
256	98691
⋮	⋮

Tabla 2. Datos de algoritmo de rotación de imagen con fuerza bruta.

Los datos de ejecución del algoritmo se muestran en la imagen 8. El algoritmo muestra un crecimiento muy rapido con respecto al ancho de las imágenes, este crecimiento esta acotado por la función $g(x) = 3x^2$ a partir de $n_0 \geq 4$

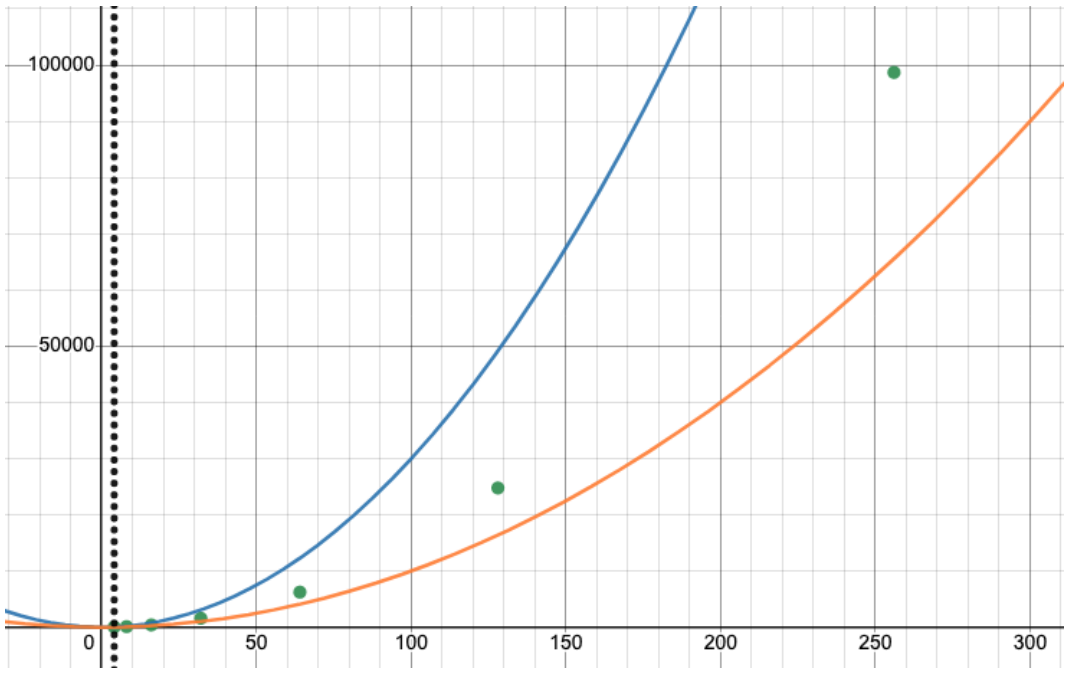

Imagen 8. Gráfica análisis a posteriori Fuerza Bruta

Por lo tanto, en el análisis a posteriori se muestra que el algoritmo es:

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

4. Conclusiones

Las estrategias de programación como divide y vencerás son óptimas pero no en todos los casos, es decir que dependiendo al problema que se implenta divide y vencerás será óptimo o no, por lo que, antes de implementar un algoritmo para un problema se tiene que hacer una análisis de complejidad para saber si es conveniente en lo práctico o no. Se muestra que la complejidad es mayor con divide y vencerás con respecto a la fuerza bruta.

Conclusiones Catonga Tecla Daniel Isaí 1

En esta práctica se pudo hacer rotar una imagen cuadrada usando divide y vencerás pero nos dimos cuenta que no es óptima, por lo que el análisis es esencial en cuanto a implementación de algoritmos. Por otra parte la práctica fue un buen ejemplo para calcular la complejidad ya que el rotar una imagen, ya que, rotar una imagen es algo que la mayoría de personas que cuentan con dispositivos móviles han hecho, pero no conocen lo que hay detrás, que son algoritmos óptimos para que se haga rápido.

Conclusiones Olguin Castillo Victor Manuel 2

Los lenguajes c y c++ me parecían lenguajes bastante básicos y con muy poco poder para resolver problemas interesantes en el área de inteligencia artificial, pero esta práctica me hizo darme cuenta como puedo analizar una imagen desde lo más profundo que son los píxeles y manipularlos, sin necesidad de alguna librería externa, esta práctica me hizo darme cuenta del poder que tiene c y c++ si se sabe implementar correctamente y con los conocimientos necesarios.

5. Anexo

Merge Sort

$$T(n) = 2T(n/2) + n$$
$$sean = 2^k$$
$$k = log(n)$$
Tenemos $T(2^k) = 2T(2^{k-2}) + 2^k$
Aplicando Sustitucion Hacia Atras
$$T(2^k) = 2T(2^{k-2}) + 2^k$$
$$T(2^k) = 2(2T(2^{k-2}) + 2^k) + 2^k$$
$$T(2^k) = 2^2T(2^{k-2}) + 2(2^k)$$
$$T(2^k) = 2^2(2T(2^{k-3}) + (2^{k-2})) + 2(2^k)$$
$$T(2^k) = 2^3T(2^{k-3}) + 3(2^k)$$
$$i \vdots$$
$$T(2^k) = 2^iT(2^{k-i}) + i(2^k)$$
$$i = k \vdots$$
$$T(2^k) = 2^kT(1) + k(2^k)$$
$$T(2^k) = 2^k + k(2^k)$$
Regresando el cambio de variable
$$T(n) = n + n(logn)$$
$$\therefore \mathrm{T(n)} \in n(logn)$$

Algoritmo Multiplicación no Clásica
Aplicando Metodo Maestro
$T(n) = 4T(\frac{n}{2}) + n$
con $a = 4, b = 2, f(n) = n$
luego $f(n) = n = O(n^log_ba - \xi) = O(n^(2 - 1)) = O(n)$
Caso I
$T(n) = \theta n^log_ba = \theta(n^2)$
$\therefore T(n) \in O(n^2)$

Multiplicacion Clasica

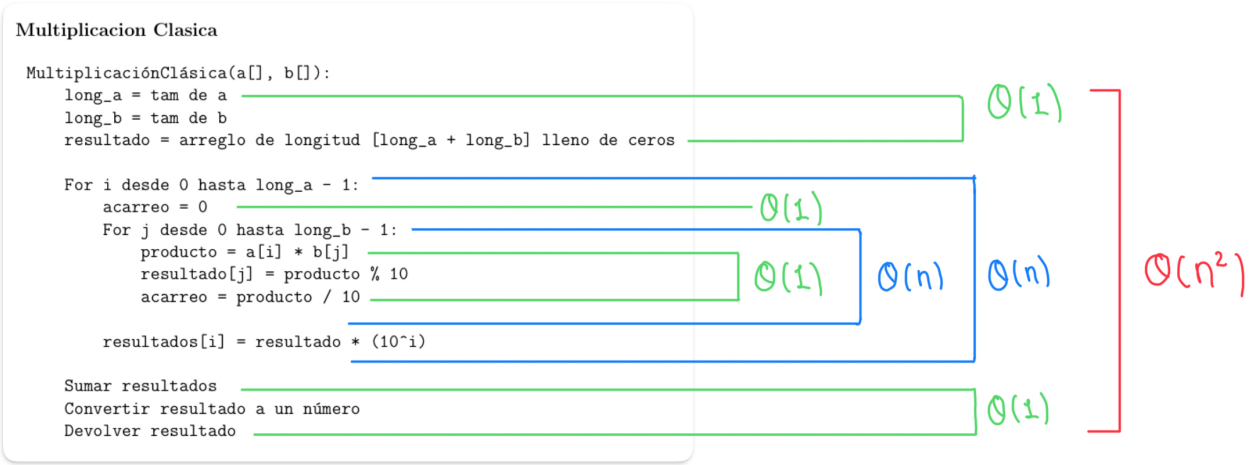

Imagen 6. Complejidad multiplicacion clasica.

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

Algoritmo Karatsuba
Aplicando Metodo Maestro
$T(n) = 3T(\frac{n}{2}) + n$
con $a = 3$, $b = 2$ y $f(n) = n$
Se tiene: $O(n^{log_b a}) = O(n^{log_2 3 - \xi}) = O(n)$
por el caso I:
(I) si $f(n) = O(n^{log_b a - \xi})$, donde $\xi = log_2 3 - 1$, entonces
$\therefore T(n) \in O(n^{log_2 3})$

Max Subarray

```
MaxSubarray(arr[0,...,1]):
suma = 0;
maximasuma = 0
i = 0
j = 0
tempi = 0
for k = 0 to k = n - 1 do:
    suma += arr[k]
    if suma > maximasuma then
        maximasuma = suma
        i = tempi
        j = k
    if suma < 0 then
        suma = 0
        tempi = k + 1
```

```
MaxSubarray(arr[0,...,1]):
suma = 0;
maximasuma = 0
i = 0
j = 0
tempi = 0
for k = 0 to k = n - 1 do:
    suma += arr[k]
    if suma > maximasuma then
        maximasuma = suma
        i = tempi
        j = k
    if suma < 0 then
        suma = 0
        tempi = k + 1
```

$O(C)$

$O(n)$

$O(n)$

$$\therefore T(n) \in O(n).$$

6. Bibliografía

Referencias

[1] REAL ACADEMIA ESPAÑOLA. *Diccionario de la lengua española, 23.ª ed.* 2023. URL: `https://dle.rae.es/algoritmo`.

[2] R. C. t. Lee. *Introducción al diseño y análisis de algoritmos: un enfoque estratégico*. Ed. por Mc Graw Hill. 2014.