



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

SEMESTRE 2023-1

PRACTICA 3

GRUPO: 3CV11

MATERIA: ANALISIS DE ALGORITMOS

ALUMNOS:

ISAAC SÁNCHEZ VERDIGUEL

ISANCHEZV1603@ALUMNO.IPN.MX

AXEL TREVIÑO PALACIOS

ATREVINOP1500@ALUMNO.IPN.MX

INSTITUTO POLITÉCNICO NACIONAL



ESCOM

MAESTRO:

BENJAMIN LUNA BENOSO

09 Noviembre 2022

Índice general

1	Introducción	3
1.1	Resumen	3
1.2	Introducción	3
2	Desarrollo	4
2.1	Conceptos Básicos	4
2.2	Algoritmos Iterativos	5
2.2.1	Algoritmo Iterativo 1: Calculo de cocientes I	5
2.2.2	Algoritmo Iterativo 2: Calculo de cocientes II	6
2.2.3	Algoritmo Iterativo 3: Búsqueda Terciaria	7
2.3	Algoritmos Recursivos	8
2.3.1	Algoritmo Recursivo 1: Calculo de cocientes III	8
2.3.2	Algoritmo Recursivo 2: Búsqueda Terciaria	8
3	Experimentación y Resultados	9
3.1	Algoritmos: Calculo de Cocientes (iterativos y recursivos)	9
3.1.1	Análisis a Priori 1	9
3.1.2	Análisis a Priori 2	10
3.1.3	Análisis a Priori 3	11
3.1.4	Análisis a Posteriori 1	12
3.2	Algoritmos Búsqueda Terciaria (iterativo y recursivo)	13
3.2.1	Análisis a Priori 4	13
3.2.2	Análisis a Priori 5	14
3.2.3	Análisis a Posteriori 2	15
3.3	Pantallas de Ejecución de Algoritmos	16
4	Conclusiones	17
4.1	Conclusiones Generales	17
4.2	Isaac Sánchez - Conclusiones	18
4.3	Axel Trevino - Conclusiones	19

Índice de figuras

3.1	Analisis a Priori: Calculo de Cocientes I	9
3.2	Analisis a Priori: Calculo de Cocientes II	10
3.3	Analisis a Priori: Calculo de Cocientes III	11
3.4	Análisis a Posteriori: Calculo de Cocientes	12
3.5	Analisis a Priori: Búsqueda Terciaria Iterativo	13
3.6	Analisis a Priori: Búsqueda Terciaria Recursivo	14
3.7	Análisis a Posteriori: Búsqueda Terciaria Iterativo	15
3.8	Análisis a Posteriori: Búsqueda Terciaria Recursivo	15
3.9	Ejecución de Calculo de Cocientes	16
3.10	Ejecución de Búsqueda Terciaria	16
4.1	Isaac Sánchez	18
4.2	Axel Treviño	19

1 | Introducción

1.1. Resumen

La practica consta del análisis a priori y posteriori de cinco algoritmos, 3 iterativos y 2 recursivos. Ambos algoritmos fueron desarrollados mediante el lenguaje de programación **Python** en un ambiente virtual de **Linux**.

Palabras Clave: Python, Algoritmo, Recursividad, Iteratividad, Complejidad.

1.2. Introducción

Los algoritmos son una parte fundamental de la ciencia de la computación, ya que estos al ser computables pueden dar solución o una idea más concreta acerca de la solución de un problema.

Un algoritmo no siempre dará una solución correcta, lo cual jamás será malo, porque esto nos ayudará a poder minimizar su radio de error. Una característica casi obligatoria para el buen funcionamiento de un algoritmo es su **rendimiento y eficacia**. El rendimiento adecuado se encuentra en la solución más rápida y menos costosa [Cormen, 2009].

Los algoritmos presentados en esta práctica fueron desarrollados siguiendo los principios de iteratividad y recursividad para poder obtener los resultados esperados planteados en cada problema. Los primeros algoritmos, iterativos y recursivos, presentados definen el cálculo de cocientes, mostrando el resultado de una división de enteros positivos sin sus residuos. El último algoritmo a comprende una búsqueda terciaria dentro de un arreglo, el arreglo es partido en 3 subarreglos para facilitar la búsqueda entre cada arreglo. El resultado es arrojar un *booleano* que indique si el elemento buscado está dentro de alguno de los arreglos.

2 | Desarrollo

2.1. Conceptos Básicos

La **complejidad temporal**, dentro del análisis de algoritmos, es el número de operaciones que ejecuta un algoritmo en cierto tiempo. Su denotación es $T(n)$ y puede ser analizada mediante dos tipos de análisis:

- Análisis de priori: entrega una función que muestra el tiempo de cálculo de un algoritmo.
- Análisis a posteriori: es la prueba en tiempo real del algoritmo, midiendo su costo mediante valores de entrada.

El análisis de complejidad temporal define que un algoritmo alcanza su máximo potencial cuando los valores de entrada son mayores al tiempo estimado de ejecución, siendo que es factible poder completar sus ejecuciones en menor tiempo posible.

Dentro del vasto mundo de los algoritmos, los algoritmos *recursivos* e *iterativos* serán analizados en esta práctica.

¿Qué es un algoritmo iterativo? Es un algoritmo que se caracteriza por ejecutarse mediante ciclos. Es muy útil al momento de realizar tareas repetitivas (como recorrer un arreglo de datos) [Wikipedia, 2020]. Casi todos los lenguajes de programación modernos tienen palabras reservadas para la realización de iteraciones.

¿Qué es un algoritmo recursivo? Es un algoritmo que expresa la solución un problema en términos de una llamada a sí mismo, de modo que resuelva una instancia más pequeña del mismo problema. La llamada a sí mismo se conoce como llamada recursiva o recursión [Wikipedia, 2021]. Este depende del algoritmo iterativo, ya que cumple la misma función con la diferencia de que al retornar se devuelve a sí mismo.

2.2. Algoritmos Iterativos

2.2.1. Algoritmo Iterativo 1: Calculo de cocientes I

Resumen del problema

Ejecutar y realizar los análisis a priori y posteriori de un algoritmo que sea capaz de calcular los cocientes de dos números enteros positivos de manera iterativa. Es el primer algoritmo iterativo.

Pseudocódigo Algoritmo Iterativo 1 Calculo de cocientes I

Los pseudocódigos consiguen los cocientes mediante ciclos que realizan operaciones básicas de suma y resta para retornar el resultado.

Algorithm 1: Calculo de cocientes Iterativo I

Result: (n/div)

int n , div , res ;

$q = 0$;

while $n \geq div$ **do**

$n = n - div$;
 $q = q + 1$;

$res = n$;

return q ;

2.2.2. Algoritmo Iterativo 2: Calculo de cocientes II

Pseudocódigo Algoritmo Iterativo 2 Calculo de cocientes II

Algorithm 2: Calculo de cocientes Iterativo II

Result: n/div

int n , div , res ;

int $dd = div$;

$q = 0$;

$r = n$;

while $dd \leq n$ **do**

$dd = 2 * dd$;

$q = q + 1$;

while $dd > div$ **do**

$dd = dd/2$;

$q = 2 * q$;

if $dd \leq r$ **then**

$r = r - dd$;

$q = q + 1$;

else

 —

return q ;

2.2.3. Algoritmo Iterativo 3: Búsqueda Terciaria

Resumen del problema

Realizar el análisis a priori y posteriori de un algoritmo que encuentre un elemento dentro de un arreglo de valores mediante la división del arreglo en 3 subarreglos del mismo tamaño, buscando en cada uno y devolviendo la posición del elemento en cualquiera de los 3 arreglos.

Pseudocódigo Algoritmo Iterativo 3 Búsqueda Terciaria

Algorithm 3: Búsqueda Terciaria

Result: *verificador* = *Booleano*

```
while length(arreglo) ≥ 3 do
    n = floor(length(arreglo)/3);
    if arreglo[0] == elemento or arreglo[n] == arreglo or arreglo[2*n] == elemento
        then
            | return True;
    else
        if elemento < arreglo[n] then
            | arr = arreglo[0:n];
        else
            if elemento < arreglo[n*2] then
                | arr = arreglo[n:n*2];
            else
                | arr = arreglo[n*2:n*3];
    [
[
[
return False;
```

2.3. Algoritmos Recursivos

2.3.1. Algoritmo Recursivo 1: Calculo de cocientes III

Pseudocódigo Algoritmo Recursivo 1 Calculo de cocientes III

Algorithm 4: Calculo de cocientes Iterativo III

Result: n/div

Data: n, div

if $div > n$ **then**

 | return 0;

else

 | return 1+Division3($n-div, div$);

2.3.2. Algoritmo Recursivo 2: Búsqueda Terciaria

Pseudocódigo Algoritmo Recursivo 1 Búsqueda Terciaria

Algorithm 5: Búsqueda Terciaria

Result: $verificador = \text{Booleano}$

if $length(arreglo) < 3$ **then**

 | return False;

else

 | continue;

$n = \text{floor}(length(arreglo)/3);$

if $arreglo[0] == elemento$ **or** $arreglo[n] == arreglo$ **or** $arreglo[2*n] == elemento$ **then**

 | return True;

else

 | **if** $elemento < arreglo[n]$ **then**

 | $arr = arreglo[0:n];$

 | **else**

 | **if** $elemento < arreglo[n*2]$ **then**

 | $arr = arreglo[n:n*2];$

 | **else**

 | $arr = arreglo[n*2:n*3];$

return binarioRecursivo($arr, elemento$);

3 | Experimentación y Resultados

Aquí se presentaran los resultados del **Análisis a Priori y Posteriori** de cada algoritmo solicitado respectivamente.

3.1. Algoritmos: Calculo de Cocientes (iterativos y recursivos)

Los algoritmos codificados en *Python* fue testeado en un entorno virtual de Linux, mientras que el análisis a priori fue realizado en los pseudocódigos.

3.1.1. Análisis a Priori 1

La figura 3.1 presenta el análisis a priori realizado sobre el pseudocódigo del primer algoritmo iterativo de calculo de cocientes. Concluyendo que el algoritmo presenta una complejidad $f(n) = O(n/div)$

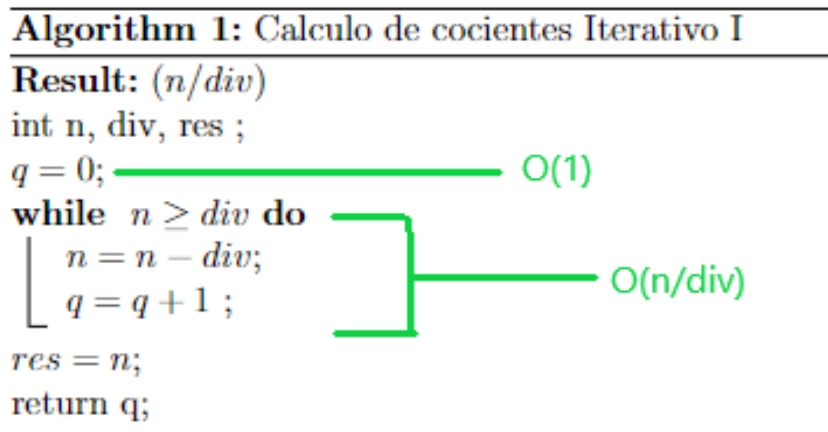


Figura 3.1: Analisis a Priori: Calculo de Cocientes I

3.1.2. Análisis a Priori 2

La figura 3.2 presenta el análisis a priori realizado sobre el pseudocódigo del segundo algoritmo iterativo de calculo de cocientes. Concluyendo que el algoritmo presenta una complejidad $f(n) = O(\log n)$

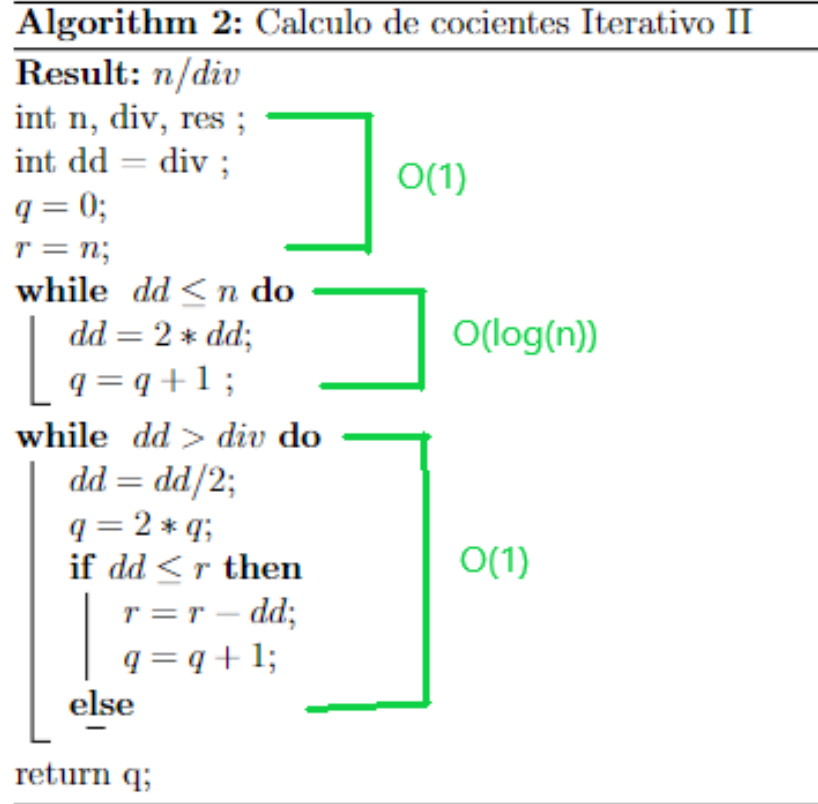


Figura 3.2: Analisis a Priori: Calculo de Cocientes II

3.1.3. Análisis a Priori 3

La figura 3.3 presenta el análisis a priori realizado sobre el pseudocódigo del tercer algoritmo iterativo de calculo de cocientes. Resolviendo

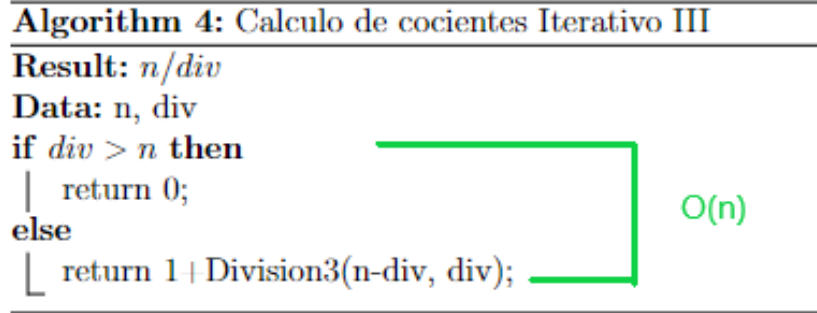


Figura 3.3: Analisis a Priori: Calculo de Cocientes III

$$T(n) = T(n - div) + O(1)$$

$$T(n) = T(n - div) + c$$

Aplicando sustitución hacia atrás:

$$T(n) = [T(n - 2div) + c] + c$$

$$T(n) = T(n - 2div) + 2c$$

Donde $div = i, n-i = 0, i=n$

$$T(n) = T(n - i) + ic$$

$$T(n) = T(0) + nc$$

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

3.1.4. Análisis a Posteriori 1

En el análisis posteriori se verifica el análisis a priori, efectivamente pudimos concluir que el cálculo de cocientes presenta tanto una complejidad lineal como logarítmica para sus diferentes casos. Para la gráfica mostrada a continuación (figura 3.4) se muestran 1000 elementos graficados.

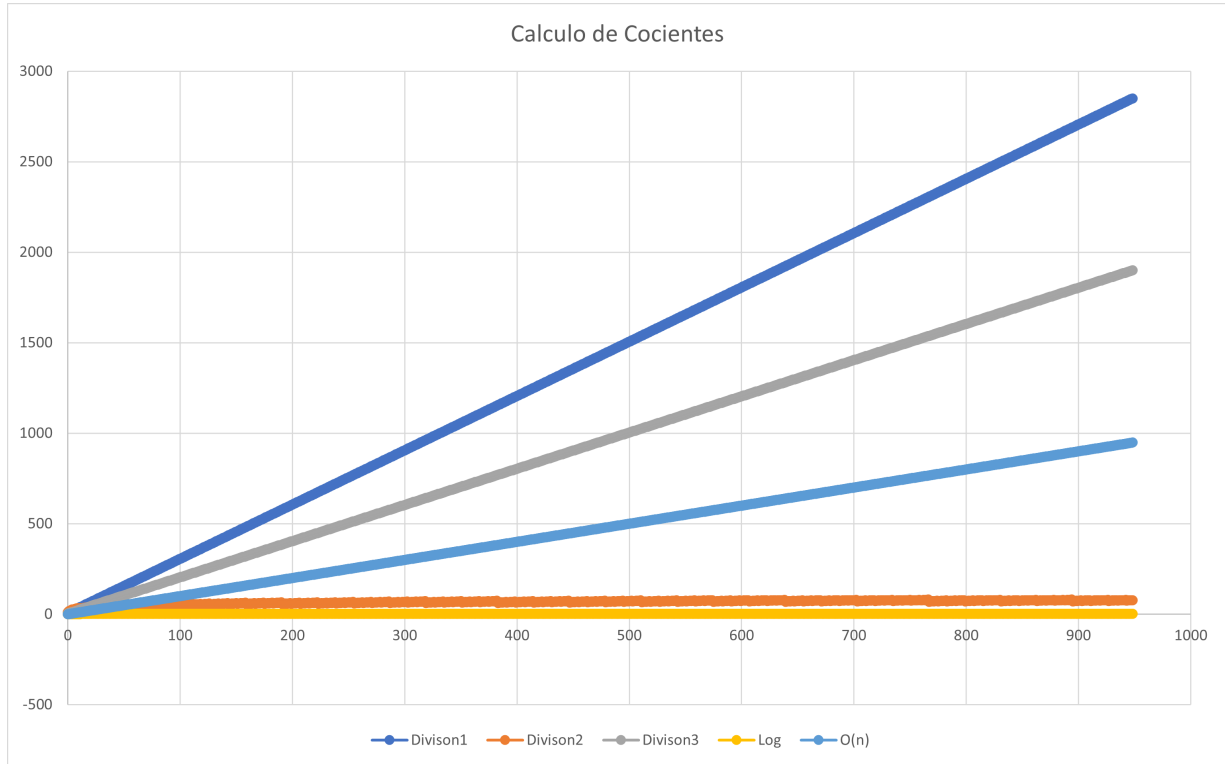


Figura 3.4: Análisis a Posteriori: Calculo de Cocientes

3.2. Algoritmos Búsqueda Terciaria (iterativo y recursivo)

El algoritmo codificado en *Python* fue testeado en un entorno virtual de Linux, mientras que el análisis a priori se realizó sobre los pseudocódigos.

3.2.1. Análisis a Priori 4

La figura 3.5 presenta el análisis a priori realizado sobre el pseudocódigo del primer algoritmo de búsqueda terciaria. Concluyendo que el algoritmo presenta una complejidad $f(n) = O(\log n)$

Algorithm 3: Búsqueda Terciaria

Result: *verificador* = *Booleano*

```
while length(arreglo) ≥ 3 do
  n = floor(length(arreglo)/3);
  if arreglo[0] == elemento or arreglo[n] == arreglo or arreglo[2*n] == elemento
    then
      return True;
    else
      if elemento < arreglo[n] then
        arr = arreglo[0:n];
      else
        if elemento < arreglo[n*2] then
          arr = arreglo[n:n*2];
        else
          arr = arreglo[n*2:n*3];
  return False;
```

O(log(n))

O(1)

Figura 3.5: Analisis a Priori: Búsqueda Terciaria Iterativo

3.2.2. Análisis a Priori 5

La figura 3.6 presenta el análisis a priori realizado sobre el pseudocódigo del segundo algoritmo de búsqueda terciaria. Resolviendo

Algorithm 5: Búsqueda Terciaria

Result: *verificador* = *Booleano*

```

if length(arreglo) < 3 then
    return False;
else
    continue;
n = floor(length(arreglo)/3); O(1) T(n/3)+O(1)
if arreglo[0] == elemento or arreglo[n] == arreglo or arreglo[2*n] == elemento then
    return True;
else
    if elemento < arreglo[n] then T(n/3)+O(1)
        arr = arreglo[0:n];
    else
        if elemento < arreglo[n*2] then T(n/3)+O(1)
            arr = arreglo[n:n*2];
        else
            arr = arreglo[n*2:n*3];
return binarioRecursivo(arr, elemento);

```

Figura 3.6: Analisis a Priori: Búsqueda Terciaria Recursivo

$$T(n) = T(n/3) + O(1)$$

$$T(n) = T(n/3) + c$$

Aplicando sustitución hacia atrás donde $n = 3^k$ y $k = \log_3 n$:

$$T(n) = T(3^{k-1}) + c$$

$$T(n) = [T(3^{k-1}) + c] + c$$

$$T(n) = T(3^{k-1}) + 2c$$

Donde $k-1 = 0$, $i=k$

$$T(n) = T(3^{k-i}) + ic$$

$$T(n) = T(3^0) + kc$$

$$T(n) = c + c \log_3 n$$

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

3.2.3. Análisis a Posteriori 2

En el análisis posteriori del algoritmo en su forma iterativa y recursiva, en las figuras 3.7 y 3.8 se verifica que el grado de complejidad de ambos algoritmos es el mismo, $f(n) = O(\log n)$, siendo que su complejidad aumenta en gran medida mediante más elementos aleatorios se introducen.

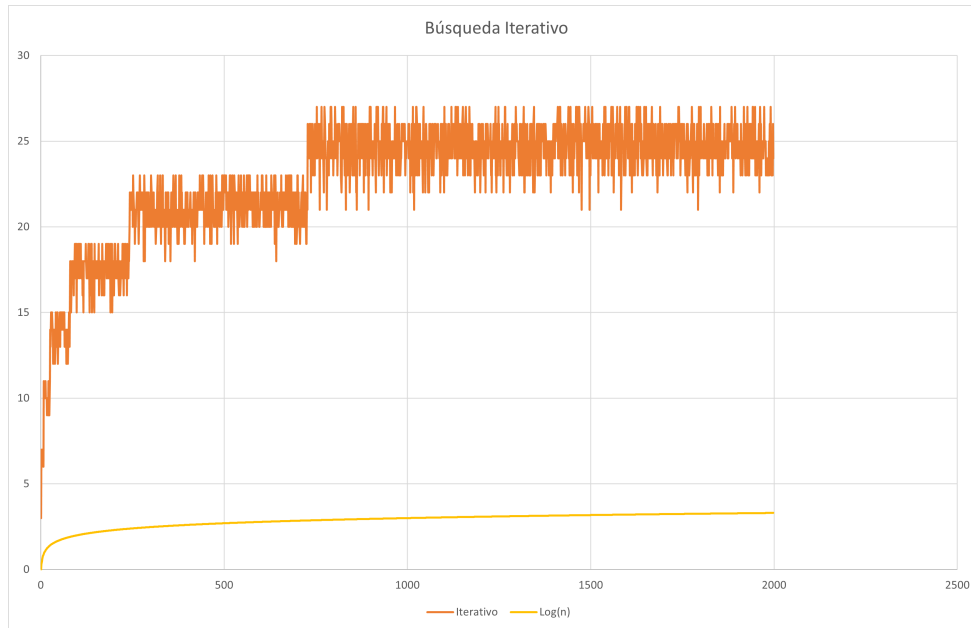


Figura 3.7: Análisis a Posteriori: Búsqueda Terciaria Iterativo

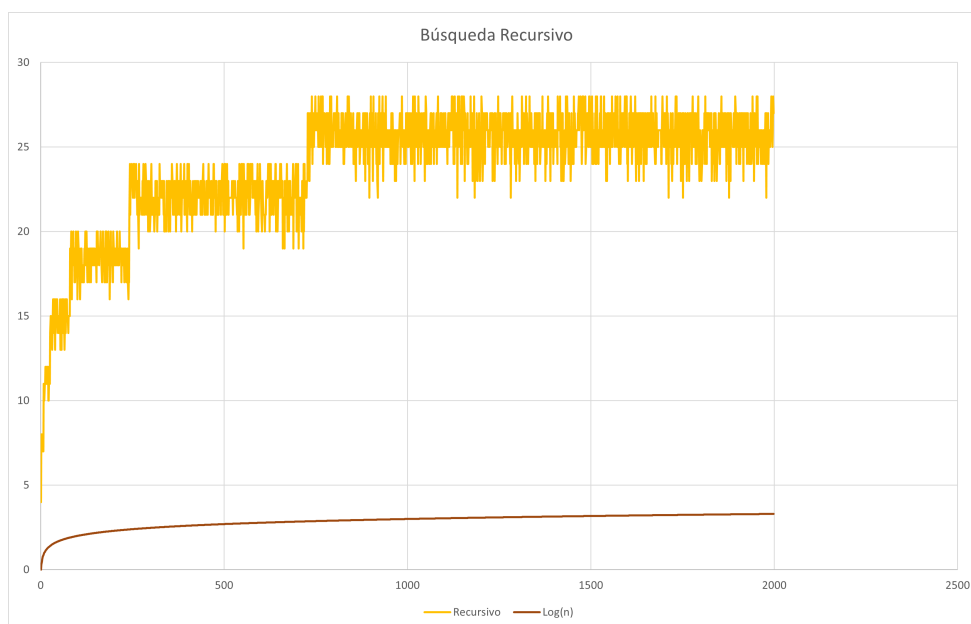
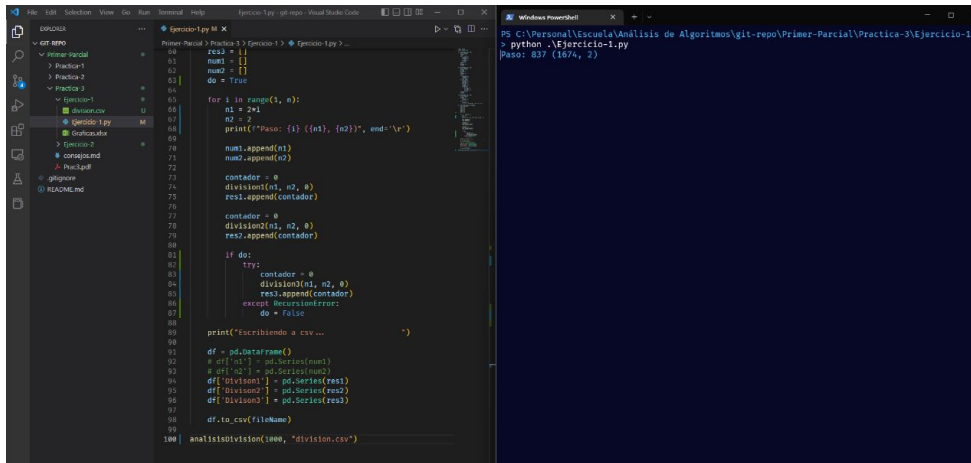


Figura 3.8: Análisis a Posteriori: Búsqueda Terciaria Recursivo

3.3. Pantallas de Ejecución de Algoritmos

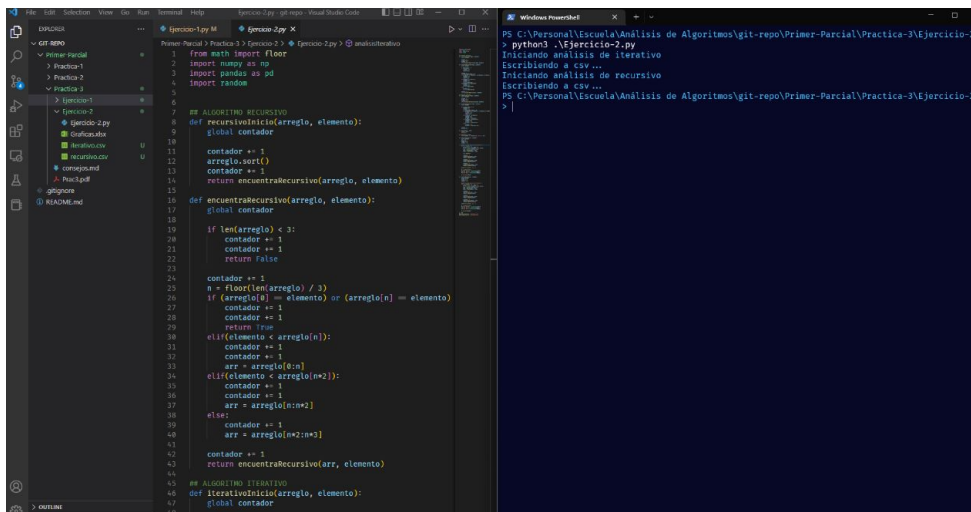
Se muestra en las figura 3.9 y 3.10 la ejecución de los algoritmos. En la figura 3.9 se puede observar la ejecución de los 3 algoritmos del cálculo de cocientes en un solo script de *Python*. Mientras que en la figura 3.10 se observa la ejecución de ambos algoritmos de búsqueda terciaria. En la terminal los resultados son compuestos por la longitud de n para el arreglo y ambos tiempos de ejecución.



```
64 res2 = []
65 num1 = []
66 num2 = []
67 do = True
68 for i in range(1, n):
69     n1 = 2*i
70     n2 = 2
71     print("Paso: {} ({n1}, {n2})", end="\\n")
72     num1.append(n1)
73     num2.append(n2)
74     contador = 0
75     division1(n1, n2, 0)
76     res1.append(contador)
77     contador = 0
78     division2(n1, n2, 0)
79     res2.append(contador)
80     if do:
81         try:
82             contador = 0
83             division3(n1, n2, 0)
84             res3.append(contador)
85         except RecursionError:
86             do = False
87     print("Escribiendo a csv ...")
88     df = pd.DataFrame()
89     a["n1"] = pd.Series(num1)
90     a["n2"] = pd.Series(num2)
91     df["Division1"] = pd.Series(res1)
92     df["Division2"] = pd.Series(res2)
93     df["Division3"] = pd.Series(res3)
94     df.to_csv(filename)
95     analisisDivision(1000, "division.csv")
```

```
PS C:\Personal\Escuela\Análisis de Algoritmos\git-repo\Primer-Parcial\Practica-3\Ejercicio-1> python .\Ejercicio-1.py
Paso: 837 (1674, 2)
```

Figura 3.9: Ejecución de Calculo de Cocientes



```
1 from math import floor
2 import numpy as np
3 import pandas as pd
4 import random
5
6 # ALGORITMO RECURSIVO
7 def recursivoInicio(arreglo, elemento):
8     global contador
9     contador += 1
10    arreglo.sort()
11    contador += 1
12    return encuentraRecursivo(arreglo, elemento)
13
14 def encuentraRecursivo(arreglo, elemento):
15     global contador
16     if len(arreglo) < 3:
17         contador += 1
18         return False
19     contador += 1
20     n = floor(len(arreglo) / 3)
21     if (arreglo[0] == elemento) or (arreglo[n] == elemento):
22         contador += 1
23         return True
24     elif (elemento < arreglo[n]):
25         contador += 1
26         arr = arreglo[0:n]
27     elif (elemento > arreglo[n+2]):
28         contador += 1
29         arr = arreglo[n+2:]
30     else:
31         contador += 1
32         arr = arreglo[n+2:n+3]
33     contador += 1
34     return encuentraRecursivo(arr, elemento)
35
36 # ALGORITMO ITERATIVO
37 def iterativoInicio(arreglo, elemento):
38     global contador
```

```
PS C:\Personal\Escuela\Análisis de Algoritmos\git-repo\Primer-Parcial\Practica-3\Ejercicio-2> python .\Ejercicio-2.py
Iniciando análisis de iterativo
Escribiendo a csv...
Iniciando análisis de recursivo
Escribiendo a csv...
PS C:\Personal\Escuela\Análisis de Algoritmos\git-repo\Primer-Parcial\Practica-3\Ejercicio-2>
```

Figura 3.10: Ejecución de Búsqueda Terciaria

4 | Conclusiones

4.1. Conclusiones Generales

Esta práctica ha sido la más difícil que hemos realizado, no por el hecho de la dificultad de los algoritmos, más bien por el análisis a posteriori. Encontramos dificultades a la hora de introducir los números necesarios para poder expresar bien el potencial del algoritmo. Fuera de esos problemas, encontramos fortalezas a la hora de poder crear algoritmos recursivos y las maneras en que podemos abordar la complejidad de un algoritmo.

4.2. Isaac Sánchez - Conclusiones

Agradezco poder incrementar mis habilidades dentro de los algoritmos recursivos gracias a la practica anterior. Aprendí de mejor manera el análisis a priori de un algoritmo recursivo, acerca de su complejidad y el análisis a priori. Tuvimos complicaciones a la hora de realizar el análisis a posteriori del algoritmo recursivo de la búsqueda de un elemento en un arreglo, esto debido a que desbordaba la memoria dificultando conocer la cantidad adecuada de elementos que debía contener el arreglo.



Figura 4.1: Isaac Sánchez

4.3. Axel Trevino - Conclusiones

Durante esta práctica se pudo ver la importancia de la recursividad, ya que comparando los algoritmos de ambos ejercicios pudimos visualizar la diferencia de iterativo y recursivo, lo único malo siendo el límite del stack de recursividad



Figura 4.2: Axel Treviño

Bibliografía

- [Cormen, 2009] Cormen, T. H. (2009). *Introduction to Algorithms*. The MIT Press, London.
- [Wikipedia, 2020] Wikipedia (2020). Algoritmo iterativo. https://es.wikipedia.org/wiki/Algoritmo_iterativo#:~:text=Los%20algoritmos%20iterativos%20son%20algoritmos,para%20la%20realizaci%C3%B3n%20de%20iteraciones. [Online; accessed 07-November-2022].
- [Wikipedia, 2021] Wikipedia (2021). Recursión (ciencias de computación). [https://es.wikipedia.org/wiki/Recursi%C3%B3n_\(ciencias_de_computaci%C3%B3n\)](https://es.wikipedia.org/wiki/Recursi%C3%B3n_(ciencias_de_computaci%C3%B3n)). [Online; accessed 07-November-2022].