Instituto Politécnico Nacional
Escuela Superior de Cómputo

Analisis de algoritmos, Sem: 2021-1, 3CV1,Práctica 3, 11/11/2020

Practica 3. Complejidades temporales polinomiales y no polinomiales.

Payán Téllez René
rpayant1500@alumno.ipn.mx

Resumen: En esta practica se compararan algoritmos cuya complejidad es polinomial (es decir esta acotada por un polinomio definido por "n") y algoritmos no polinomiales (es decir que no estan acotados por un polinomio). **Palabras clave:** Polinomial,No polinomial,Complejidad,C/C++

1 Introduccion

2 Conceptos Basicos

2.1 Algoritmo

La palabra algoritmo proviene del sobrenombre de un matemático árabe del siglo IX, Al-Khwarizmi, que fue reconocido por enunciar paso a paso las reglas para las operaciones matemáticas básicas con decimales (suma, resta, multiplicación y división). Vemos definición de algoritmo como un grupo de órdenes consecutivas que presentan una solución a un problema o tarea. Algunos ejemplos de algoritmos los podemos encontrar en las matemáticas (como el algoritmo para resolver una multiplicación) y en los manuales de usuario de un aparato (como una lavadora o una impresora). Sin embargo, hoy en día se relaciona la palabra algoritmo con el mundo de la informática, más concretamente en la programación; los conocidos como algoritmos informáticos.[1]

2.2 Complejidad algoritmica

Así que, por su naturaleza, un problema tiene la capacidad de ser solucionado por uno o varios métodos, pero si bien es importante llegar a la respuesta, más importante es evaluar su viabilidad. Siempre que se analiza y evalúa adecuadamente la efectividad de una solución, disminuye drásticamente el costo que representa su producción y mantenimiento, pues los recursos que se invierten posteriormente en codificación, pruebas y revisión es mucho menor siempre (como el tiempo, dinero y talento humano). Entrando en materia, la complejidad algorítmica es una métrica teórica que nos ayuda a describir el comportamiento de un algoritmo en términos de tiempo de ejecución (tiempo que tarda un algoritmo en resolver un problema) y memoria requerida (cantidad de memoria necesaria para procesar las instrucciones que solucionan dicho problema). Esto nos ayuda a comparar entre la efectividad de un algoritmo y otro, y decidir cuál es el que nos conviene implementar.[2]

2.3 Algoritmo Polinomial

2.4 Algoritmo No Polinomial

3 Experimentacion y Resultados

3.1 1. Implementar la sucecion de Fibonacci mediante un algoritmo recursivo y mediante unalgoritmo iterativo.

De la primer seccion de problemas se decidio iterar sobre todos los n y m posibles desde 0 hasta 1000.

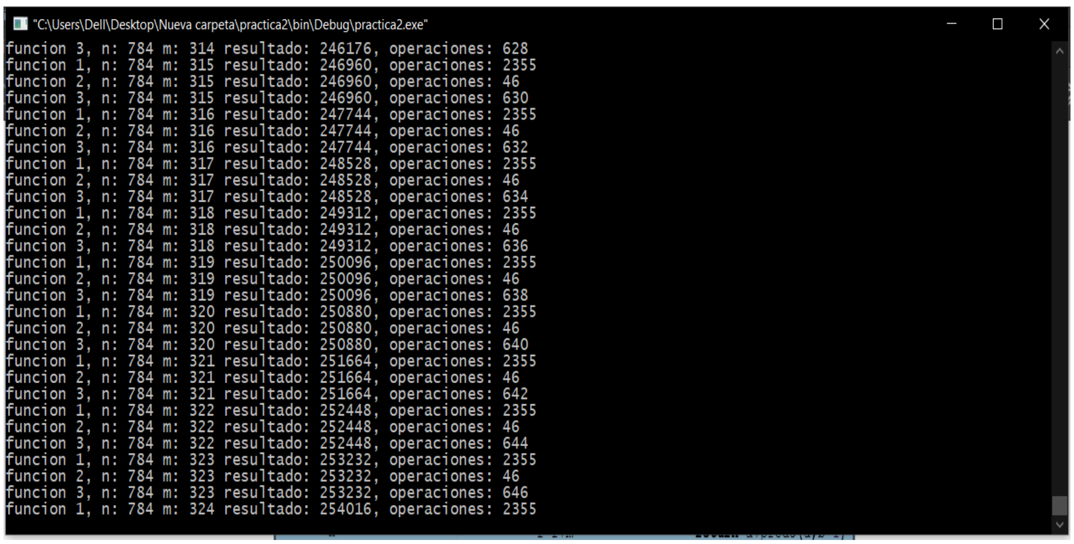

Figure 1: Ejecucion del programa que contiene las tres primeras funciones

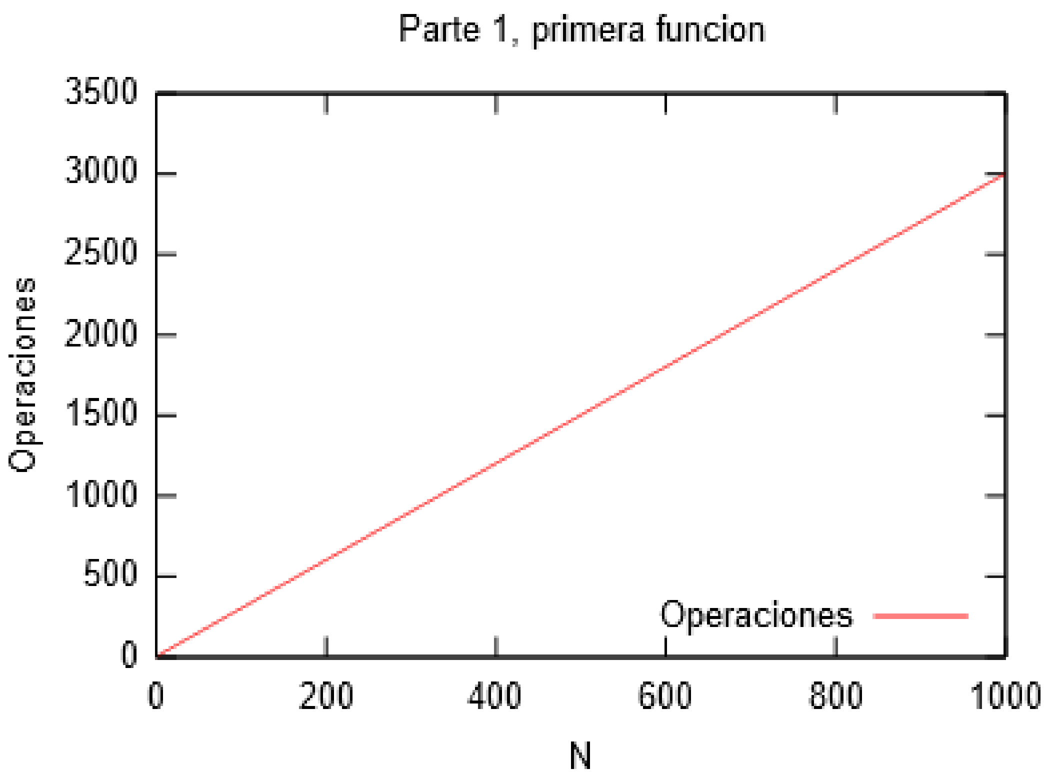

Figure 2: N contra Operaciones de la primer funcion

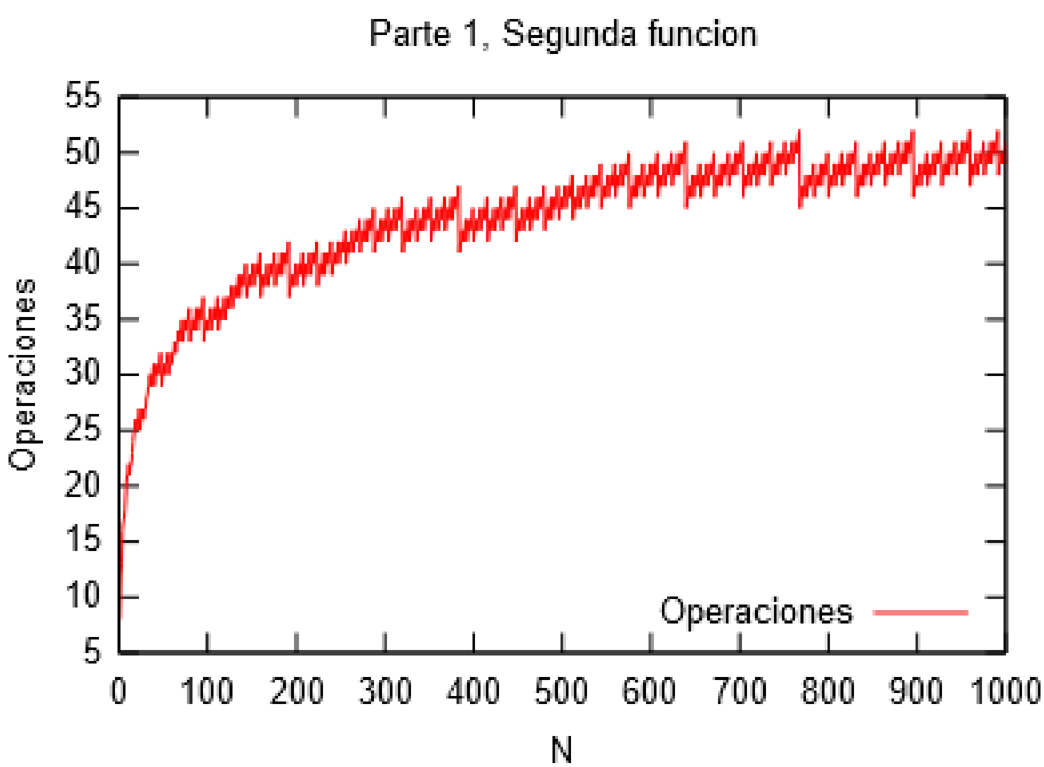

Figure 3: N contra Operaciones de la segunda funcion

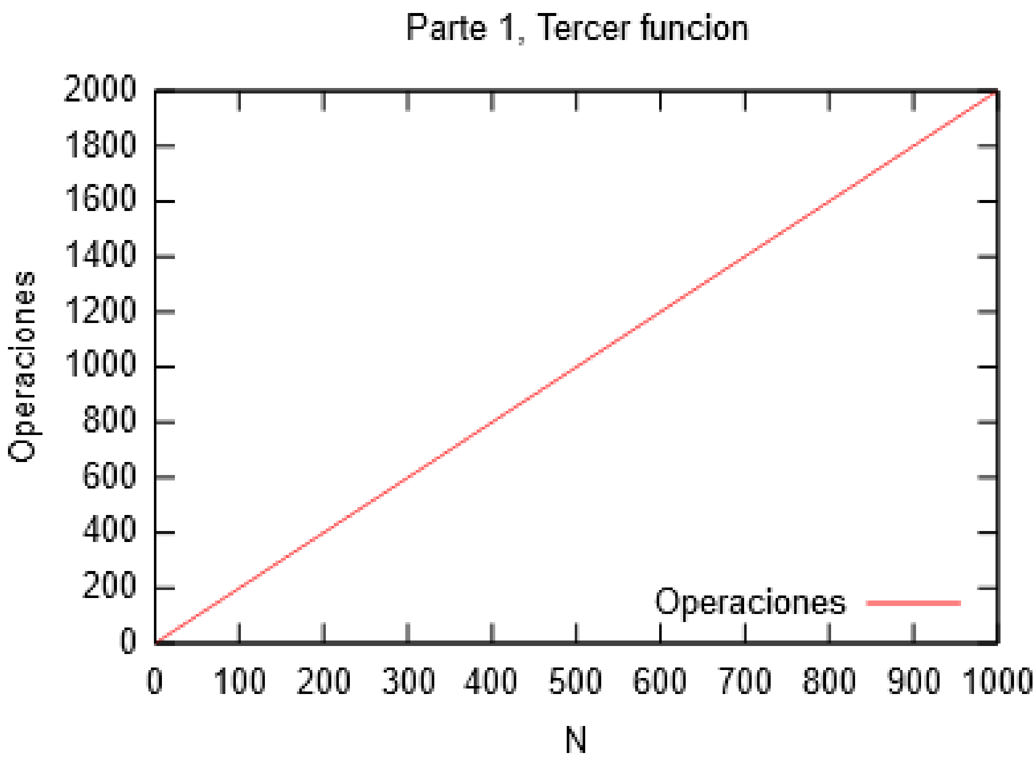

Figure 4: N contra Operaciones de la tercer funcion

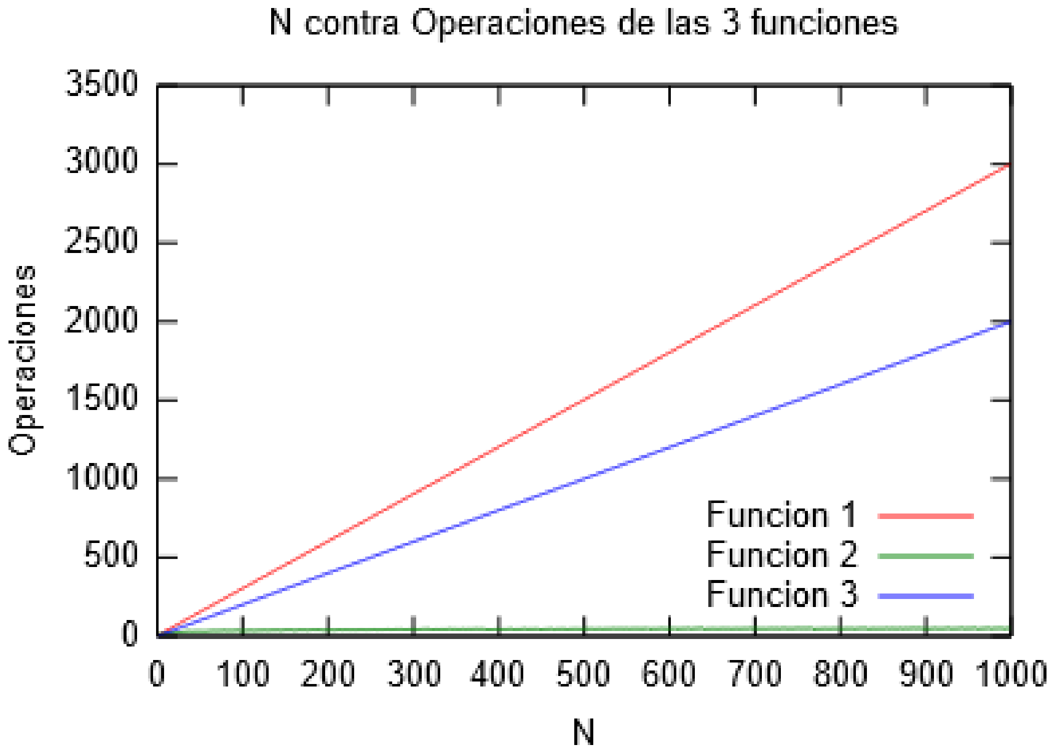

Figure 5: N contra Operaciones de las 3 funciones combinadas

Al terminar se determino que el número de operaciones solo se ve afectado por ”n”, no por ”m”, por lo que se decidio graficar solo dichos valores.
Tambien podemos observar que de las 3 funciones, la función número 2 es la mas optima y la numero 1 la menos optima.
De forma Experimental podemos concluir lo siguiente:

Funcion	**Peor escenario**	**Mejor escenario**	**Orden de complejidad**
1	n>0	$n = 0$	$O(3 * n)$
2	n>0 y n%2!=0	$n = 0$	$O(5log_2(n + 1))$
3	n=0 (Nunca termina), n>1 para todo lo demas	$n = 1$	$O(2 * n)$

Table 1: Resultados obtenidos a partir del analisis a posteriori

3.2 Cociente

De la segunda seccion de problemas se decidio iterar sobre todos los n y div posibles desde 1 hasta 1000.

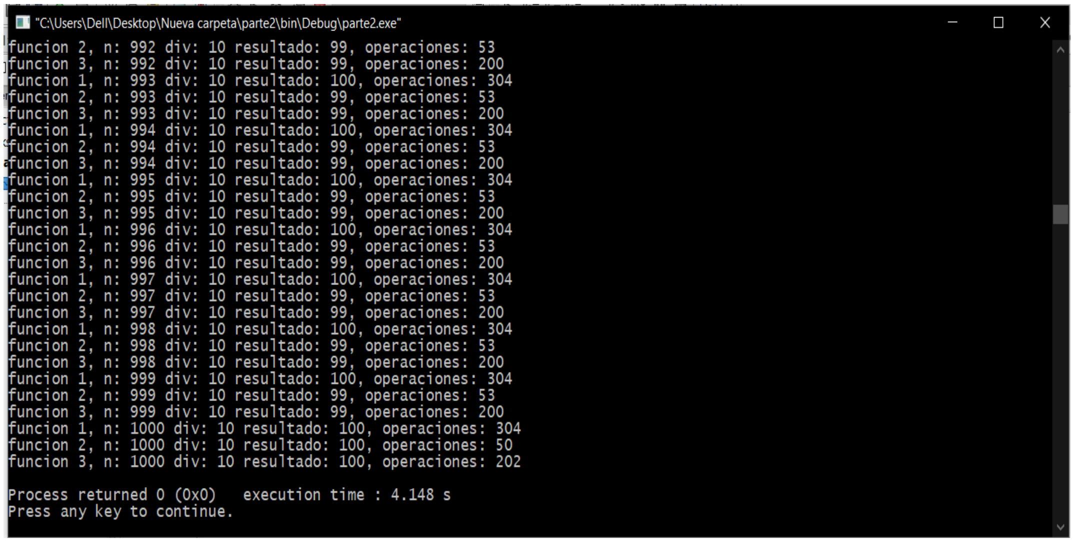

Figure 6: Ejecucion del programa que contiene las tres segundas funciones

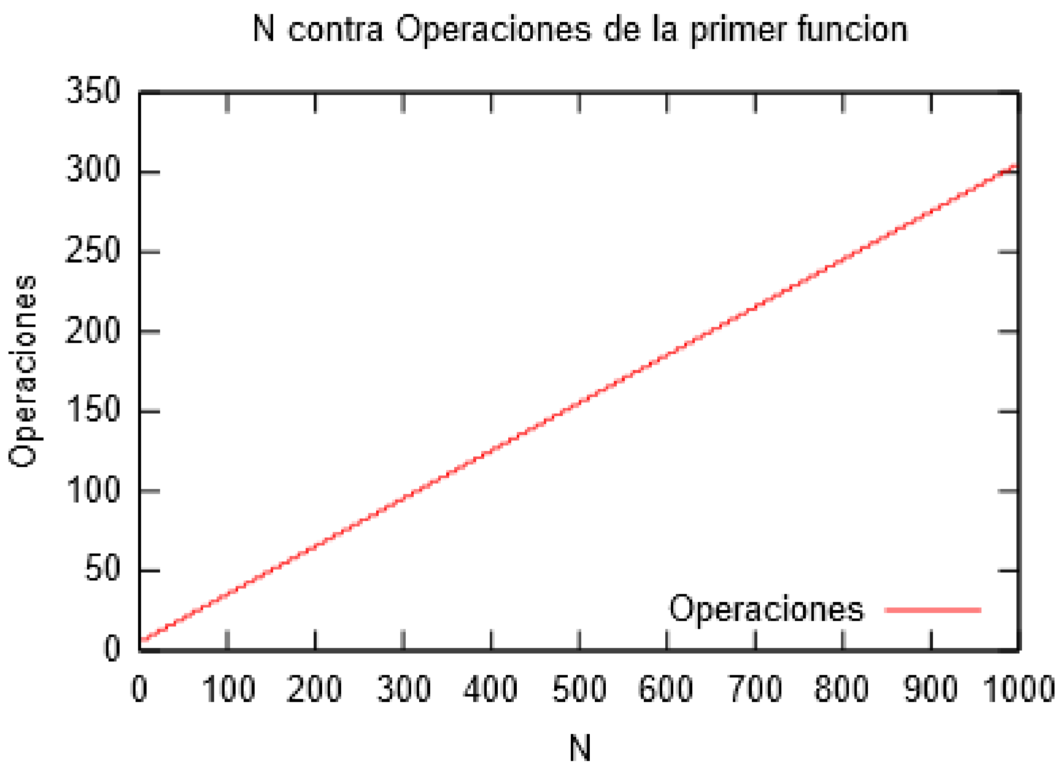

Figure 7: N contra Operaciones de la primer funcion

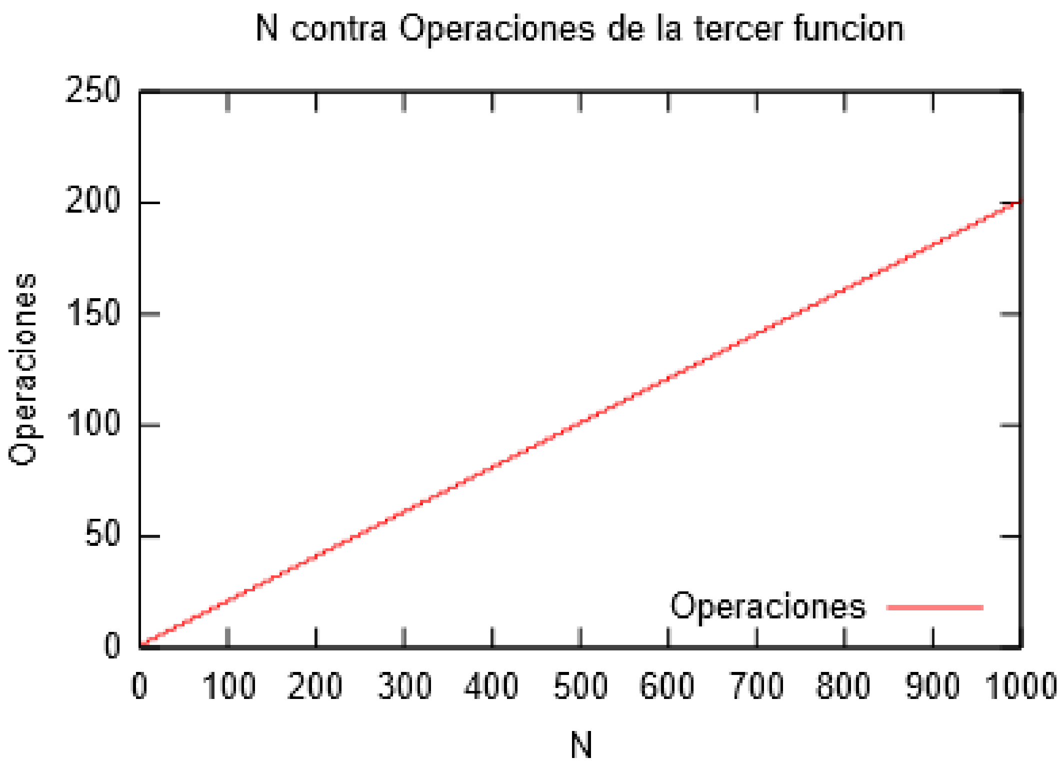

Figure 9: N contra Operaciones de la tercer funcion

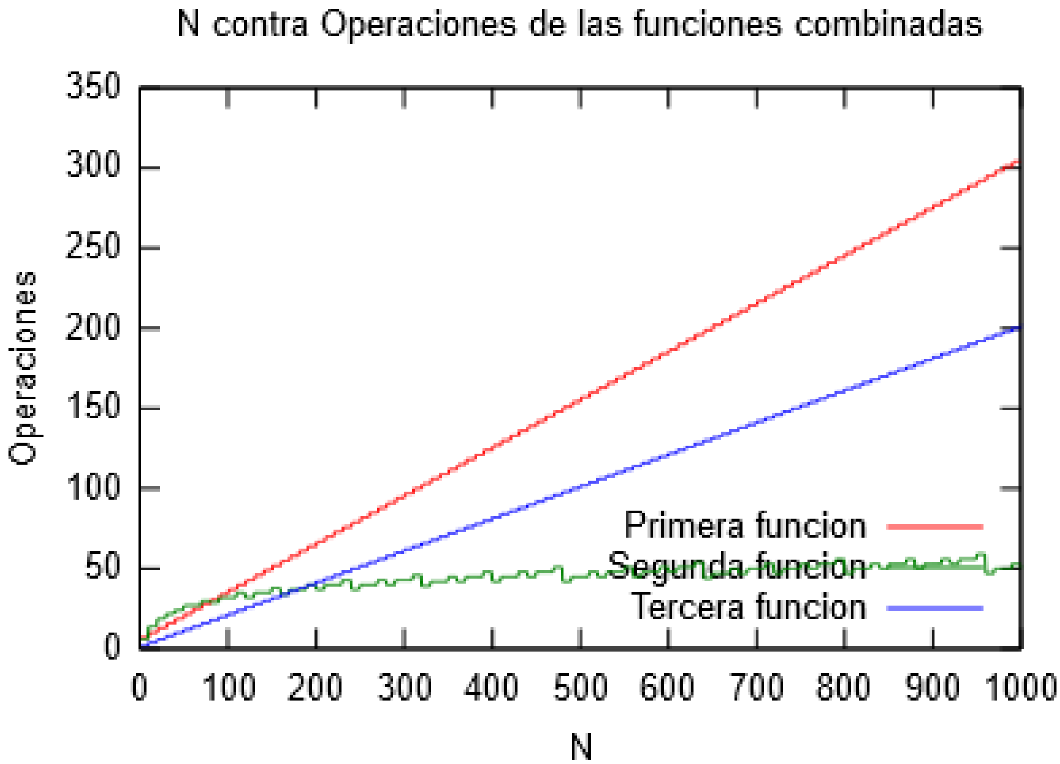

Figure 10: N contra Operaciones de las 3 funciones combinadas

Al terminar se determino que el número de operaciones solo se ve afectado por "n", no por "div", por lo que se decidio graficar solo dichos valores.
Tambien podemos observar que de las 3 funciones, la función número 2 es la mas optima y la numero 1 la menos optima.
De forma Experimental podemos concluir lo siguiente:

Funcion	**Peor escenario**	**Mejor escenario**	**Orden de complejidad**
1	n>0	$n = 0$	$O(3 * n/100)$
2	n>0 y n%2!=0	$n = 0$	$O(log_2(n + 1)/100)$
3	n¿0	$n = 1$	$O(2 * n/100)$

Table 2: Resultados obtenidos a partir del analisis a posteriori

4 Conclusiones

4.1 Payán Téllez René

Puedo concluir que despues de haber realizado los analisis practicos y teoricos de los seis algoritmos, el hecho de que un algoritmo sea recursivo, puede no ser necesariamente un indicador de su complejidad de forma directo, o ser una señal directa de que ese algoritmo es el mas optimo. Por ejemplo en ambos escenarios el algoritmo mas optimos termino siendo un algoritmo iterativo, que aunque aclaro los algoritmos recursivos fueron mejores que su contraparte iterativa, pero aun asi, lo que determino cual fue el mejor algoritmo, es aquel que pudiera optimizar mejor los recursos dados y solventar de forma mas ingeniosa el problema planteado.

5 Anexo

5.1 Producto

Función 1

De esta función podemos analizar que su peor escenario es cuando: $n > 0$, ya que a partir de este punto se comportara de forma lineal. Mientras que su mejor escenario es cuando $n = 0$.
Posteriormente se realizo el analisis a priori:

Codigo	Costo	Veces ejecutado
r = 0	C_1	1
while(n>0)	C_2	n+1
r = r+m	C_3	n
n–	C_4	n
return r	C_5	1

$$T(n) = C_1 + (n+1)C_2 + n(C_3 + C_4) + C_5$$
$$T(n) = n(C_2 + C_3 + C_4) + C_1 + C_2 + C_5$$
$$T(n) \in \Omega(1)$$
$$T(n) \in \O(n)$$

Función 2

De esta función podemos analizar que su peor escenario es cuando: $n > 0$ y n no sea par (ejecuta el ciclo una vez mas), ya que a partir de este punto se comportara de forma logaritmica. Mientras que su mejor escenario es cuando $n = 0$.
Posteriormente se realizo el analisis a priori:

Codigo	Costo	Veces ejecutado
r =0	C_1	1
while(n>0)	C_2	$log_2(n) + 1$
if(n & 1)	C_3	$log_2(n)$
r = r+m	C_4	$log_2(n)/2$
*m=2*m*	C_5	$log_2(n)$
n=n/2	C_6	$log_2(n)$
return r	C_7	1

$$T(n) = C_1 + C_2(log_2(n) + 1) + C_4(\frac{log_2(n)}{2}) + log_2(n)(C_5 + C_6 + C_3) + C_7$$
$$T(n) = log_2(C_2 + C_5 + C_6 + C_7) + C_4\frac{log_2(n)}{2} + C_1 + C_7$$
$$T(n) = log_2(C_2 + \frac{C_4}{2} + C_5 + C_6 + C_7) + C_1 + C_7$$

$$T(n) \in \Omega(1)$$
$$T(n) \in \O(log_2(n))$$

Función 3

De esta función podemos analizar que su peor escenario es cuando: $n > 1$, ya que a partir de este punto se comportara de forma lineal. Mientras que su mejor escenario es cuando $n = 1$.
Nota: debido a la forma de la implementacion cuando n $<$ 1 el algoritmo no termina
Posteriormente se realizo el analisis a priori:

Codigo	Costo	Veces ejecutado
if b == 1	C_1	n
return 1	C_2	1
else	0	
return a+prod3(a,b-1)	C_4	n-1

$$T(n) = C_1(n) + C_2 + C_3(n-1)$$
$$T(n) = n(C_1 + C_3) + C_2 - C_3$$
$$T(n) \in \Omega(1)$$
$$T(n) \in \O(n)$$

Comparacion

Despues de obtener la complejidad a priori y a posteriori de las 3 funciones, se concluye que la funcion 2 es la mas optima y el algoritmo número 1 es el menos optimo.

Funcion	Calculo a priori	Calculo a posteriori
1	$O(n)$	$O(3n)$
2	$O(log_2(n))$	$O(5log_2(n+1))$
3	$O(n)$	$O(2n)$

Cociente

Función 1

De esta función podemos analizar que su peor escenario es cuando: $n > 0$, ya que a partir de este punto se comportara de forma lineal. Mientras que su mejor escenario es cuando $n = 0$.
Posteriormente se realizo el analisis a priori:

Codigo	Costo	Veces ejecutado
q = 0	C_1	1
while(n>0)	C_2	n+1
n = n-div	C_3	n
q++	C_4	n
**r = n*	C_5	1
return q	C_6	1

$$T(n) = C_1 + (n+1)C_2 + n(C_3 + C_4) + C_5 + C_6$$
$$T(n) = n(C_2 + C_3 + C_4) + C_1 + C_2 + C_5 + C_6$$
$$T(n) \in \Omega(1)$$
$$T(n) \in \O(n)$$

Función 2

De esta función podemos analizar que su peor escenario es cuando: $n > 0$ y n no sea par (ejecuta el ciclo una vez mas), ya que a partir de este punto se comportara de forma logaritmica. Mientras que su mejor escenario es cuando $n = 0$.
Posteriormente se realizo el analisis a priori:

Codigo	Costo	Veces ejecutado
dd = div	C_1	1
q = 0	C_2	1
**r = n*	C_3	1
while(d≤n)	C_4	$log_2(n) + 1$
*dd=2*dd*	C_5	$log_2(n)$
while(dd>div)	C_6	$log_2(n) + 1$
dd=dd/2	C_7	$log_2(n)$
*q=2*q*	C_8	$log_2(n)$
*if(dd≤*r)*	C_9	$log_2(n)/2 + 1$
**r=*r-dd*	C_{10}	$log_2(n)/2$
q++	C_{11}	$log_2(n)/2$
return q	C_{12}	1

$$T(n) = C_1 + C_2 + C_3 + C_4(log_2(n) + 1) + C_5(log_2(n)) + C_6(log_2(n) + 1) + log_2(n)(C_7 + C_8) + \\ (\tfrac{log_2(n)}{2} + 1)(C_9) + (\tfrac{log_2(n)}{2})(C_{10} + C_{11}) + C_{12}$$
$$T(n) = log_2(C_4 + C_5 + C_6 + C_7 + C_8) + C_4\tfrac{log_2(n)}{2} + C_1 + C_7$$
$$T(n) = log_2(C_2 + \tfrac{C_4}{2} + C_5 + C_6 + C_7) + C_1 + C_7$$
$$T(n) \in \Omega(1)$$
$$T(n) \in \O(log_2(n))$$

Función 3

De esta función podemos analizar que su peor escenario es cuando: $n > 1$, ya que a partir de este punto se comportara de forma lineal. Mientras que su mejor escenario es cuando $n = 1$.
Nota: debido a la forma de la implementacion cuando n < 1 el algoritmo no termina
Posteriormente se realizo el analisis a priori:

Codigo	Costo	Veces ejecutado
if div¿n	C_1	n
return 0	C_2	1
else	0	
return 1+div3(n-div,div)	C_4	n-1

$$T(n) = C_1(n) + C_2 + C_3(n - 1)$$
$$T(n) = n(C_1 + C_3) + C_2 - C_3$$
$$T(n) \in \Omega(1)$$
$$T(n) \in \O(n)$$

Comparacion

Despues de obtener la complejidad a priori y a posteriori de las 3 funciones, se concluye que la funcion 2 es la mas optima y el algoritmo número 1 es el menos optimo.

Funcion	Calculo a priori	Calculo a posteriori
1	$O(n)$	$O(\frac{3n}{100})$
2	$O(log_2(n))$	$O(log_2(n+1))$
3	$O(n)$	$O(\frac{2n}{100})$

6 Bibliografia

[1]https://openwebinars.net/blog/que-es-un-algoritmo-informatico/

[2]https://medium.com/@joseguillermo_/qu%C3%A9-es-la-complejidad-algor%C3%ADtmica-y-con-qu%C3%A9-se-come-2638e7fd9e8c

[3]https://rsanchezs.gitbooks.io/ciencia-de-datos-con-r/content/estructuras_control/iterativas/estructuras_iterativas.html

[4]https://thatcsharpguy.com/tv/recursion-iteracion/

[5]http://formacion.desarrollando.net/cursosfiles/formacion/curso_454/deda-03.pdf