



TRABAJO PRÁCTICO INTEGRADOR



PROGRAMACIÓN I
UTN - TUP

AGUSTINA GRILLE - COMISION 15

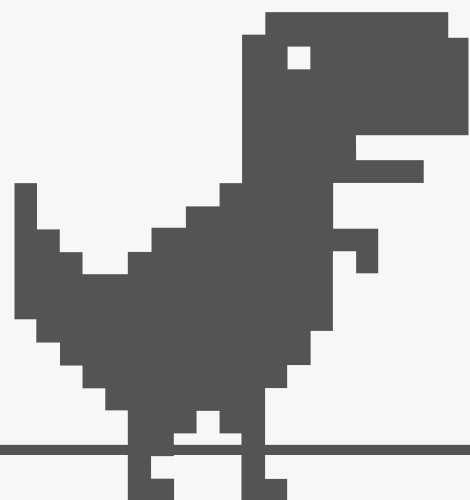
PROFESOR: ARIEL ENFERREL

TUTOR: MAXIMILIANO SAR FERNANDEZ

ALEXIS BORDA - COMISION 10

PROFESOR: CINTHIA RIGONI

TUTOR: BRIAN LARA



INTRODUCCIÓN

¿QUÉ ES UN ALGORITMO?

UN ALGORITMO ES UNA SECUENCIA FINITA Y ORDENADA DE PASOS QUE PERMITEN REALIZAR UNA TAREA ESPECÍFICA

¿PARA QUE ANALIZARLOS?

PORQUE NOS PERMITE APRENDER OBSERVANDO CÓMO DOS SOLUCIONES DIFERENTES PARA UN MISMO PROBLEMA PUEDEN TENER RENDIMIENTOS COMPLETAMENTE DIFERENTES ENTRE SÍ, ESTO NOS PERMITE ENTENDER Y APRENDER CUAL ES LA MEJOR MANERA DE RESOLVER ALGO.





NUESTRO OBJETIVO

SENTIMOS QUE ES MUY INTERESANTE PODER
OBSERVAR COMO, A PARTIR DE DOS SOLUCIONES QUE
RESUELVEN EL MISMO PROBLEMA PODEMOS APRENDER
DE LAS MISMAS, Y DE SUS DIFERENCIAS, QUÉ ES LO
MEJOR EN CUANTO A RENDIMIENTO.

A PARTIR DE ELLO, DECIDIMOS LLEVAR A CABO EL
ANALISIS COMPARATIVO ENTRE UNA SUMA ITERATIVA,
UNA SOLUCION BASADA EN UNA FORMULA MATEMÁTICA
Y DECIDIMOS SUMAR ADICIONALMENTE RECURSIVIDAD
PARA REALIZAR SUMATORIAS DENTRO DE UN MISMO
MÉTODO;

EVALUANDOLAS A PARTIR DE UN TIMER EN PYTHON.

MARCO TEORICO



NOTACIÓN BIG-O

LA NOTACIÓN BIG-O NOS PERMITE DESCRIBIR CÓMO
CRECE EL TIEMPO DE EJECUCIÓN DE UN ALGORITMO A
MEDIDA QUE AUMENTA LA CANTIDAD DE DATOS DE
ENTRADA (N) SE ENFOCA EN EL COMPORTAMIENTO A
GRAN ESCALA DEL ALGORITMO, IGNORANDO DETALLES
MENORES (COMO VARIABLES)

CADA TIPO DE ALGORITMO TIENE UNA COMPLEJIDAD
QUE DEFINIRÁ SU NIVEL DE EFICIENCIA.
CUANTO MÁS BAJA ES LA COMPLEJIDAD DEL MISMO,
MEJOR ESCALARÁ SU RENDIMIENTO.



The background is a pixel art landscape. At the top, there are three pixelated mountains. In the center, the title 'TIPOS DE COMPLEJIDAD' is written in a large, bold, pixelated font. Below the title, there is a light gray rectangular box containing a list of complexity types. At the bottom, there is a horizontal line representing the ground. On the left side of the ground, there are two pixelated cacti. On the right side, there is a pixelated dinosaur (T-Rex) facing right. The ground is decorated with small, scattered pixel patterns.

TIPOS DE COMPLEJIDAD

$O(1)$ CONSTANTE
TIEMPO FIJO

$O(n)$ LINEAL
TIEMPO PROPORCIONAL A N

$O(n^2)$ CUADRÁTICA
TIEMPO PROPORCIONAL A N^2

$O(2^n)$ EXPONENCIAL
TIEMPO SE DUPLICA CON CADA
AUMENTO DE UNA UNIDAD EN
EL TAMAÑO DE ENTRADA N



EFICIENCIA ALGORITMICA

AL PROGRAMAR LOS ALGORITMOS SON ALGO ASI COMO UNA RECETA, UN PASO A PASO QUE LA COMPUTADORA SEGUIRÁ PARA RESOLVER EL PROBLEMA QUE LE PLANTEAMOS.

PERO, COMO TODA RECETA, EXISTEN MAS DE UNA FORMA DE REALIZARLAS, Y PARA ENCONTRAR LA MEJOR, LA EFICIENCIA ALGORITMICA NOS PERMITE ENTENDER CUANTO TIEMPO Y CUANTA MEMORIA NECESITA UN ALGORITMO PARA FUNCIONAR. SOBRE TODO SI TRABAJAMOS CON MUCHO VOLUMEN DE DATOS.

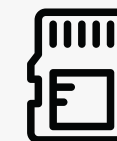
ESTO PODEMOS DIFERENCIARLO CON:

EFICIENCIA TEMPORAL



EL TIEMPO QUE
DEMORA EN
EJECUTARSE

EFICIENCIA ESPACIAL



LA CANTIDAD DE
MEMORIA QUE
NECESITARÁ
UTILIZAR



RECURSOS UTILIZADOS



PYTHON



MODULO TIME

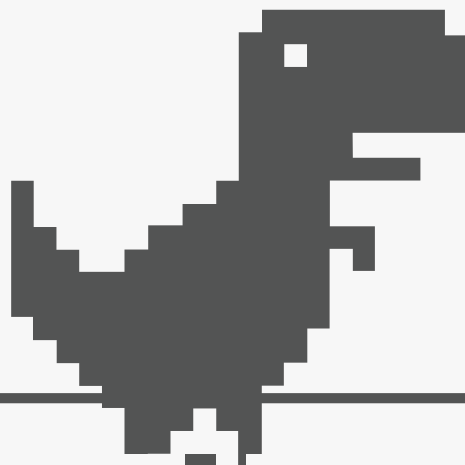
BIG O CHEAT SHEET



VSCODE



GITHUB



The image is a pixel art illustration of a desert landscape. In the center, a white rectangular box with a black border contains the text "EJEMPLO" and "CASO PRACTICO" in a pixelated font. The background features a light blue sky with three white, pixelated clouds. The ground is a solid black horizontal line. Below this line, there are several small, black, pixelated cacti and a larger, black, pixelated dinosaur standing on the right side. The overall style is reminiscent of early computer graphics or video game sprites.

EJEMPLO CASO PRACTICO

ALGORITMOS UTILIZADOS PARA EL ANALISIS

Suma Iterativa

```
def suma_iterativa(n):  
    total = 0  
    for i in range(1, n + 1):  
        total += i  
    return total
```

Complejidad temporal $O(n)$
Complejidad espacial $O(1)$

Suma con fórmula

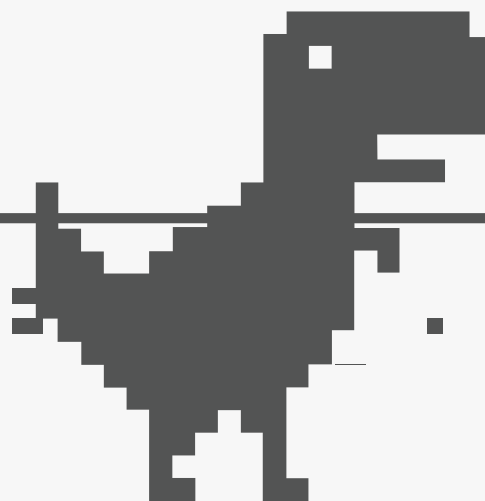
```
def suma_formula(n):  
    return n * (n + 1) // 2
```

Complejidad temporal $O(1)$
Complejidad espacial $O(1)$

Suma Recursiva

```
def suma_recursiva(n):  
    if n == 0:  
        return 0  
    return n + suma - 1
```

Complejidad temporal $O(n)$
Complejidad espacial $O(n)$



COMPARACION Y ANALISIS

TIEMPOS DE EJECUCIÓN
CON N = 900

ITERATIVA: RECORRE CADA NUMERO DEL 1
AL N Y LOS SUMA UNO A UNO.

COMPLEJIDAD: $O(N)$

ES FACIL DE ENTENDER, PERO ES LENTA
CON UN VOLUMEN GRANDE DE NUMEROS
PARA SUMAR

0.000036 SEGUNDOS

FORMULA: APLICA UNA FORMULA PARA OBTENER
EL RESULTADO DE A UNA SOLA VEZ

COMPLEJIDAD: $O(1)$

ES LA MAS RAPIDA Y EFICIENTE

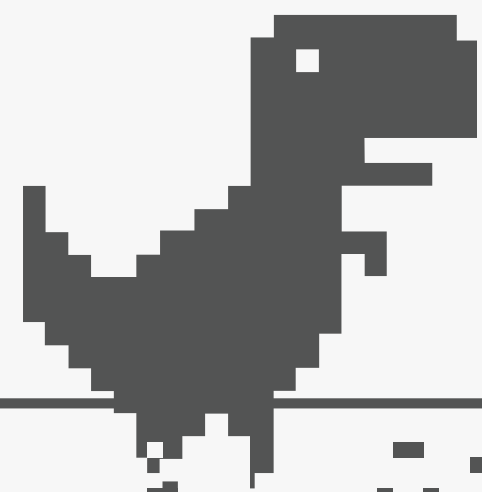
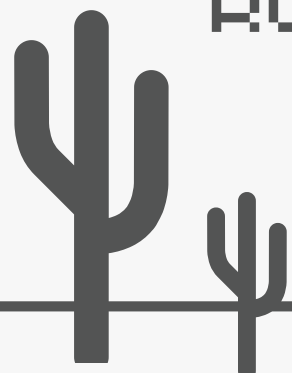
0.000002 SEGUNDOS

RECURSIVIDAD: LA FUNCION SE LLAMA A SI
MISMA HASTA VALER 0

COMPLEJIDAD: $O(N)$

ES LA MENOS EFICIENTE, A PARTIR DE GRANDES
VOLUMENES (MAYORES A 1000) LA EJECUCION SE
ROMPE CON UN ERROR DE LIMITE DE RECURSIVIDAD

0.000134 SEGUNDOS



CONCLUSIÓN

NUESTRA ELECCIÓN DEL ALGORITMO IDEAL
FUE LA FUNCION CON FORMULA.

ES LA MAS OPTIMA, QUE OCUPA MENOR
VOLUMEN DE ESPACIO Y TIEMPO.

CONSIDERAMOS QUE EN GRANDES CANTIDADES DE
VOLUMEN, O EN PROYECTOS CON NECESIDAD DE
TIEMPOS DE RESPUESTA MUY BAJOS, LA ELECCION
DEL ALGORITMO IMPACTARA DIRECTAMENTE EN
COMO FUNCIONARA NUESTRO PROYECTO.
ES POR ELLO QUE ANALIZAR EL RENDIMIENTO Y LA
EFICIENCIA DE CADA UNO, ES CRUCIAL PARA LA
VIDA UTIL DE NUESTRO CODIGO. ESTO NOS PUEDE
AHORRAR A FUTURO NUEVOS FIXES NECESARIOS
PARA MEJORAS EN LOS TIEMPOS DE EJECUCIÓN.





FIN!

