

TÉCNICAS DE PROGRAMACIÓN

ALGORITMOS

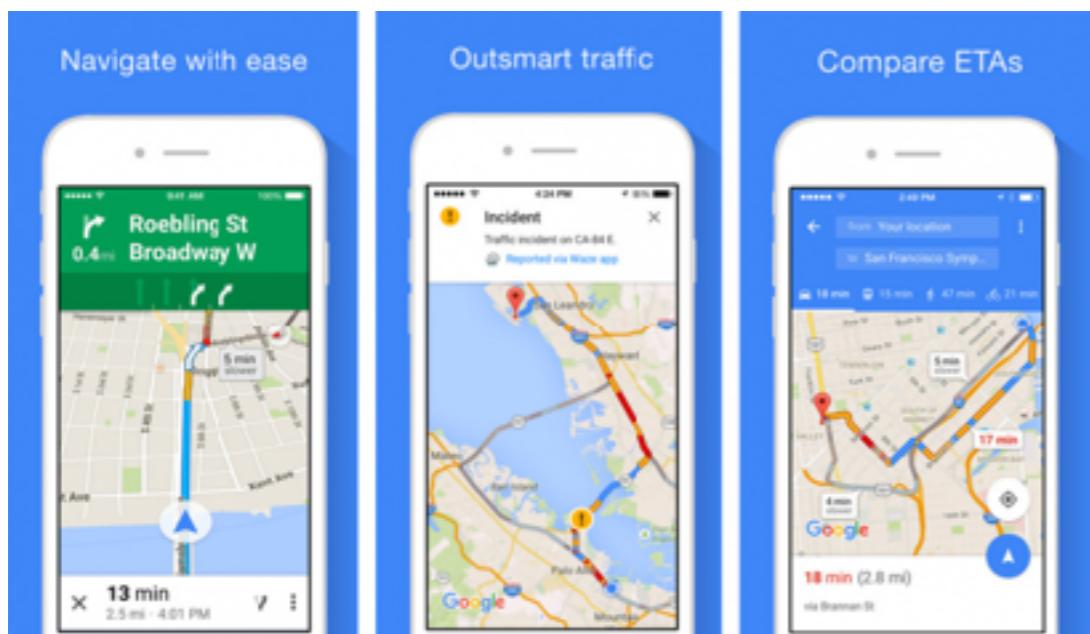
¿QUÉ ES UN ALGORITMO?

**CONJUNTO ORDENADO DE OPERACIONES
SISTEMÁTICAS QUE PERMITE HALLAR LA
SOLUCIÓN DE UN TIPO DE PROBLEMA**

ALGORITMOS



¿PARA QUÉ HAY ALGORITMOS?



VELOCIDAD

AHORRO DE ESPACIO

¿COMO SABEMOS QUE TAN
BUENO ES UN ALGORITMO?

CONSUMO DE RECURSOS

**¿QUE PUEDE AFECTAR EL LA
VELOCIDAD DE UN ALGORITMO?**

VELOCIDAD DE IN ALGORITMO

Insólito – Paloma mensajera mas rápida que internet

POR ROGER · SEPTIEMBRE 23, 2009 · 9.534

¡Imaginense tener una conexión de internet más lenta que una paloma mensajera!



GIZMODO

Bird Beats Broadband! Pigeon Flies 4GBs Faster than South African DSL



Joanna Stern

9/10/09 9:33am · Filed to: BROADBAND



22.6K 104



South Africa's broadband has got to be feeling pretty ill-equipped today considering a real, wing-flapping pigeon beat its transfer speeds. No really, a company found out that sending a bird with a 4GB USB drive was faster than uploading.

Pigeon flies past broadband in data speed race

© 16 September 2010 Technology



Broadband is the most modern of communication means, while carrier pigeons date back to Roman times.

— It on Thursday, a race between the two highlighted the low speed broadband, a pigeon won.



In urban areas, broadband clearly

The Telegraph

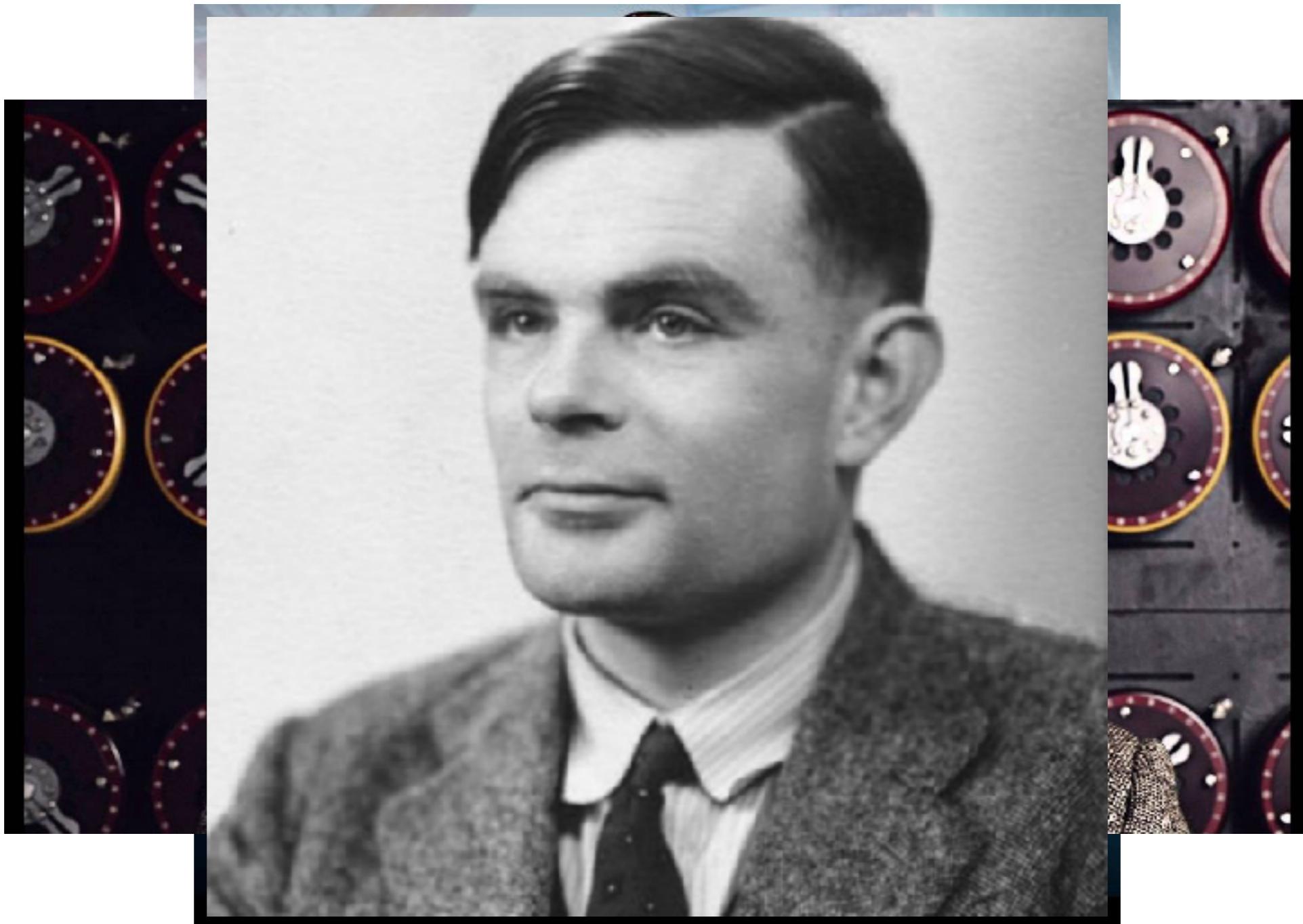
Home Video News World Sport Business Money Comment Culture Travel Life UK Politics Investigations Obituaries Education Science Family Weather Health Royal Obituaries

DONE > NEWS > UK NEWS

Tech City, where a pigeon is faster than the internet

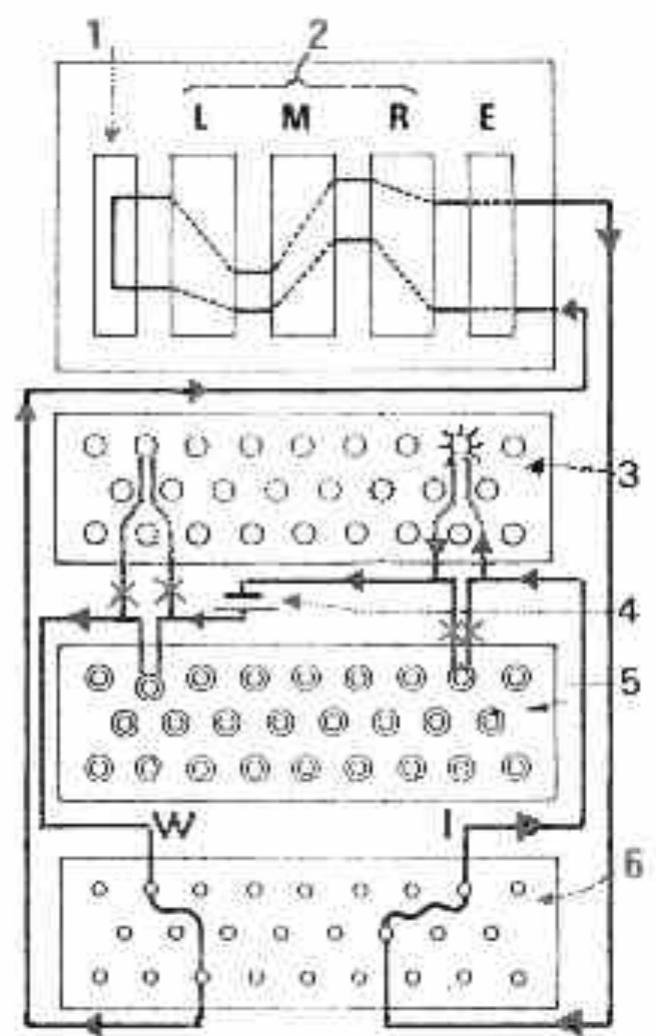
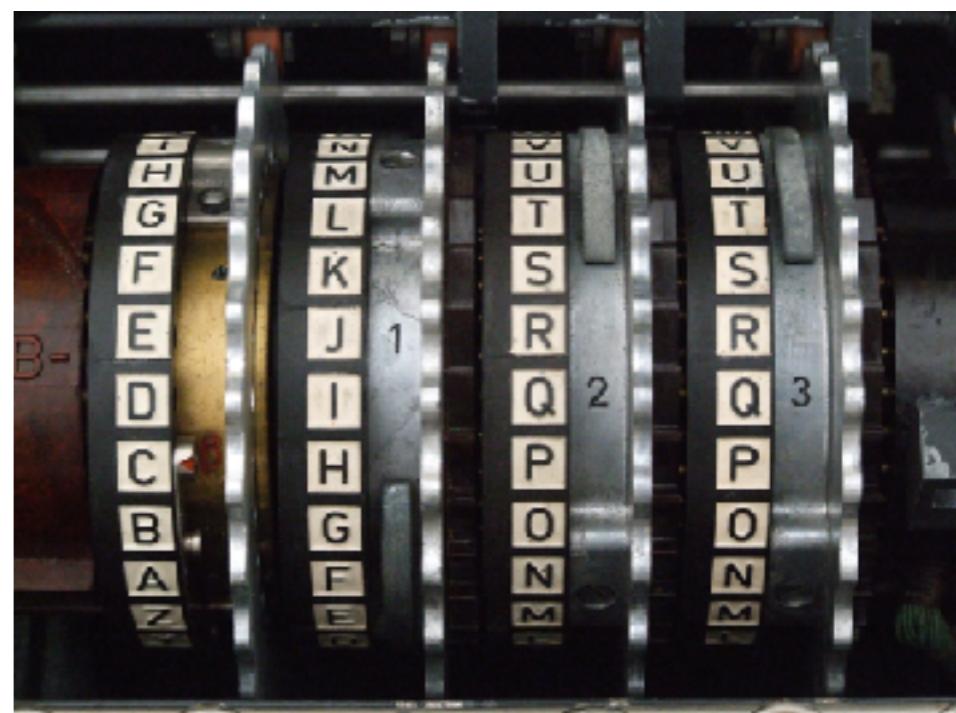
'Tech City should not be relying on Game of Thrones' ravens', Emily Thorberry MP says as she argues for the need for faster internet





ALAN TURING

ENIGMA



¿EL MISMO PROGRAMA CORRE A LA MISMA VELOCIDAD EN DIFERENTES COMPUTADORAS?

BIG O NOTATION ES EL LENGUAJE QUE USAMOS PARA ARTICULAR CUÁNTO TIEMPO Tarda UN ALGORITMO EN EJECUTARSE.

BIG O NOTATION DESCRIBIR LA FORMA DE UNA CURVA SEGÚN LA CANTIDAD DE INFORMACIÓN A PROCESAR

BIG O NOTATION SIEMPRE HARÁ REFERENCIA A EL PEOR POSIBLE ESCENARIO

BIG O NOTATION SIEMPRE SE APROXIMARA A LA CURVA DISCRETA MÁS CERCANA

BIG O NOTATION

```
def get_coordinates_from(angle, radius):  
    ... x = r * cos(angle)  
    ... y = r * sin(angle)  
    ... return (x, y)
```

O(1)

```
def item_in_list(to_check, the_list):  
    ... for item in the_list:  
    ...     ... if to_check == item:  
    ...         ... return True  
    ... return False
```

O(N)

BIG O NOTATION

```
def all_combinations(the_list):  
    results = []  
    for item in the_list:  
        for inner_item in the_list:  
            results.append((item, inner_item))  
    return results
```

$O(N^2)$

```
def print_to_100():  
    for i in range(0,100,10):  
        for j in range(i,i + 10):  
            print(j)
```

$O(N)$

BIG O NOTATION

```
def diamond(n):  
    if (n <= 0) or (n % 2 == 0):  
        return None  
    diamond = get_flor(n, n)  
    items_in_flor = n - 2  
    while items_in_flor > 0 :  
        new_flor = get_flor(n, items_in_flor)  
        diamond = new_flor + diamond + new_flor  
        items_in_flor -= 2  
    return diamond  
  
def get_flor(max_elements, num_in_flor):  
    spaces = ((max_elements - num_in_flor) // 2) * ' '  
    return spaces + (num_in_flor * '*') + '\n'
```

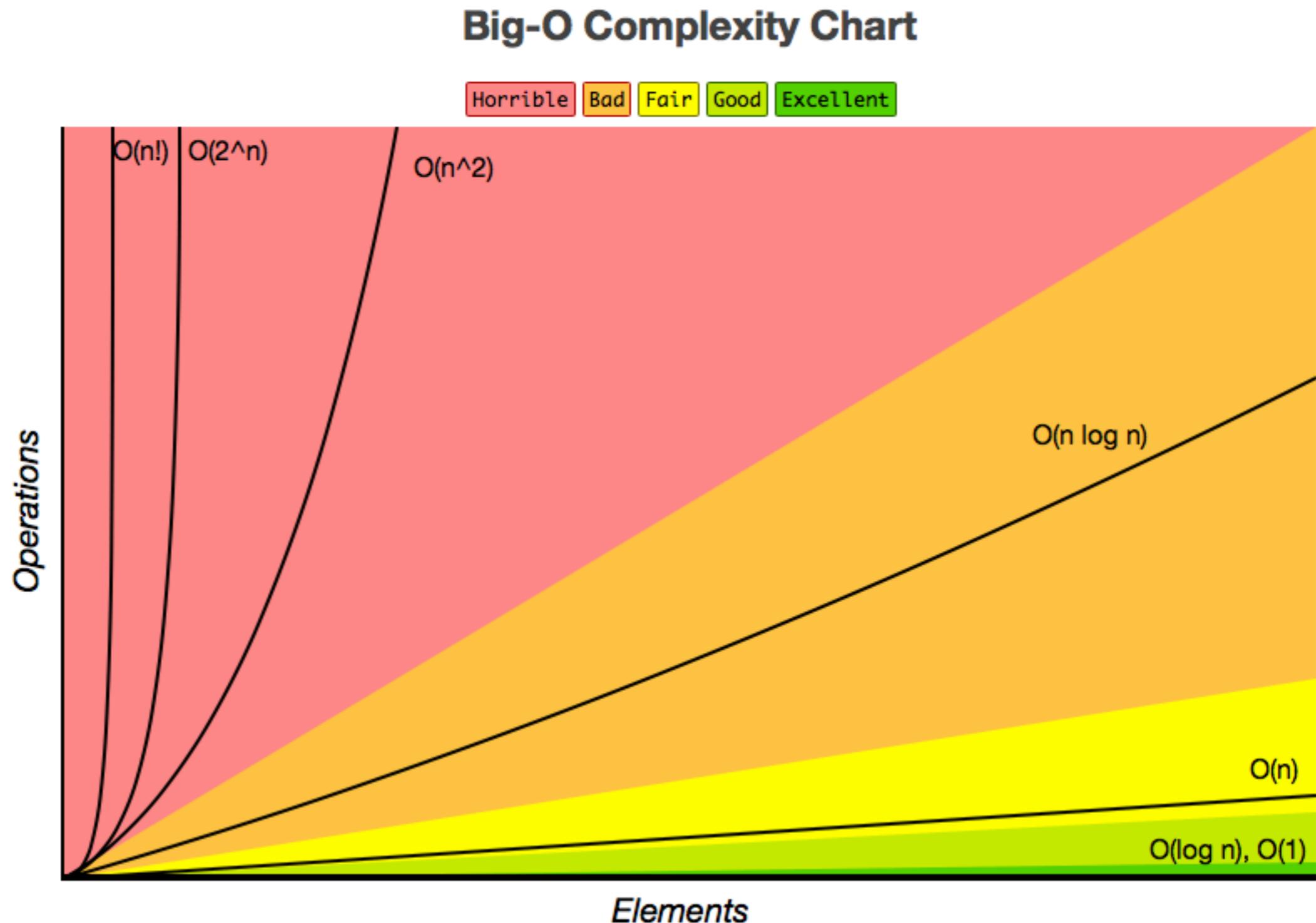
O(N)

BIG O NOTATION

```
def front_and_back (number):  
    for i range(0, number):  
        print(i)  
    for i range(number, 0, -1):  
        print(i)
```

O(N)

BIG O NOTATION



LISTAS

Operation	Average Case	Amortized Worst Case
Copy	$O(n)$	$O(n)$
Append[1]	$O(1)$	$O(1)$
Pop last	$O(1)$	$O(1)$
Pop intermediate	$O(k)$	$O(k)$
Insert	$O(n)$	$O(n)$
Get Item	$O(1)$	$O(1)$
Set Item	$O(1)$	$O(1)$
Delete Item	$O(n)$	$O(n)$
Iteration	$O(n)$	$O(n)$
Get Slice	$O(k)$	$O(k)$
Del Slice	$O(n)$	$O(n)$
Set Slice	$O(k+n)$	$O(k+n)$
Extend[1]	$O(k)$	$O(k)$
Sort	$O(n \log n)$	$O(n \log n)$
Multiply	$O(nk)$	$O(nk)$
x in s	$O(n)$	
min(s), max(s)	$O(n)$	
Get Length	$O(1)$	$O(1)$

DICCIONARIOS

Operation	Average Case	Amortized Worst Case
Copy[2]	$O(n)$	$O(n)$
Get Item	$O(1)$	$O(n)$
Set Item[1]	$O(1)$	$O(n)$
Delete Item	$O(1)$	$O(n)$
Iteration[2]	$O(n)$	$O(n)$

¿COMO ATACAR UN PROBLEMA DE OPTIMIZACIÓN?

- ▶ ENTENDER EL PROBLEMA (LEER, ESCUCHAR)
- ▶ ESTABLECER CASOS DE PRUEBA (GRANDES, PARA CASOS ESPACIALES, DIFERENTES)
- ▶ FUERZA BRUTA (DE PREFERENCIA NO LO PROGRAMES)
- ▶ OPTIMIZAR
- ▶ REVISAR EL ALGORITMO PASO A PASO
- ▶ PROGRAMA (MODULARIZAR, BUENAS PRACTICAS)
- ▶ PRUEBA (PEQUEÑOS, CRÍTICOS Y GRANDES)

Completa la función 'products_from()', la cual recibe 1 arreglo de números enteros llamado numbers y retorna un nuevo arreglo de números enteros de la misma dimensión donde cada elemento del arreglo sea el producto de todos los números del arreglo excepto el elemento que se encuentra en ese índice.

- ▶ **Ten cuidado con las posibles errores**
- ▶ **Tu solución debe de ser en $O(n)$**

Entrada: [1, 2, 6, 5, 9]

Salida: [540, 270, 90, 108, 60]



Completa la función 'find_safe_place', la cual recibe el número de soldados alineados en un círculo y retorna el lugar del último soldado sobreviviente.Ten cuidado con las posibles errores

- › **Cada soldado matara al soldado inmediato vivo un turno a la vez**
- › **Para soluciones en $O(n^2)$ menos 50 puntos del total**
- › **Para solución en $O(n)$ nada**
- › **Para solución en $O(\log(n))$ una participación**
- › **Para solución en $O(1)$ dos participaciones**

Entrada: 12

Salida: 9

Entrada: 47

Salida: 31

Entrada: 1000

Salida: 977

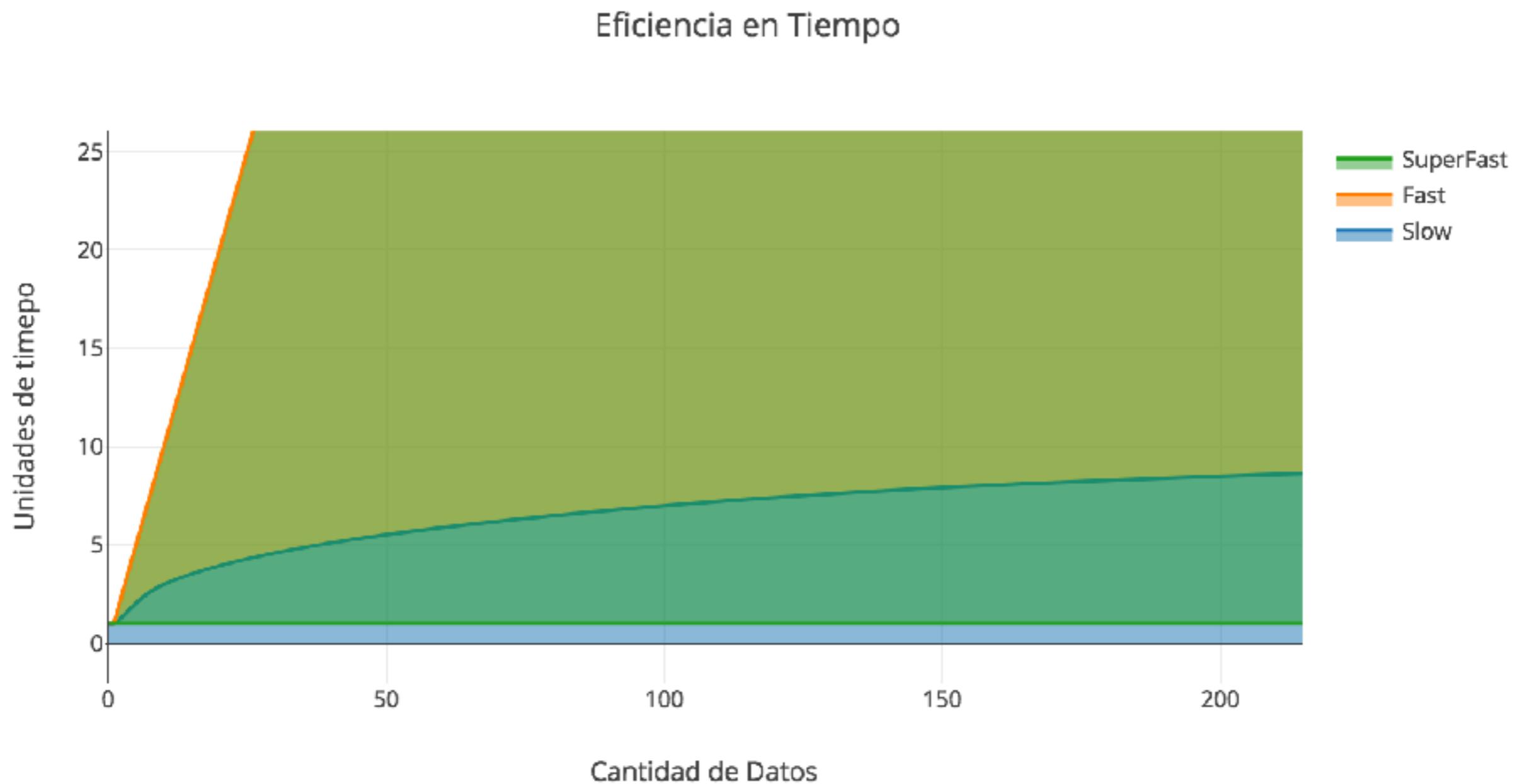


DESERTOR

Soldados Sobreviviente

1	1
2	1
3	3
4	1
5	3
6	5
7	7
8	1
9	3
10	5
11	7
12	9
13	11
14	13
15	15
16	1





RECURSION

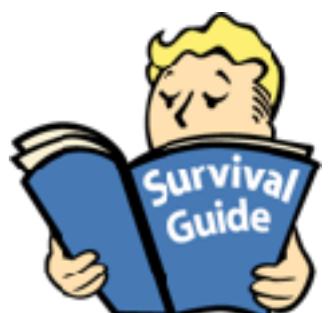
RECUSIVIDAD ES LA FORMA DE EXPRESAR LA SOLUCIÓN A UN PROBLEMA BASADO EN SU PROPIA DEFINICIÓN, EL PROBLEMA DEBE DE TENER UNA CONDICIÓN DE LA CUAL SE CONOZCA EL RESULTADO O CONDICIÓN DE SALIDA



RECUSIÓN - FACTORIAL

```
def factorial(num):  
    result = 1  
    for i in range(1, num + 1):  
        result *= i  
    return result
```

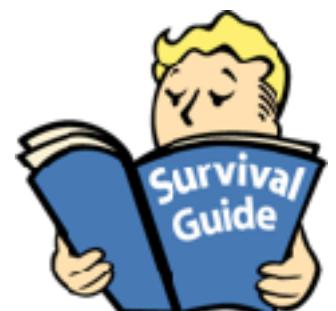
```
def factorial(num):  
    if num <= 1:  
        return 1  
    return num * factorial(num - 1)
```



RECUSIÓN - FIBONACCI



```
def fibonacci(n):
    actual_value = 1
    last_value = 1
    for i in range(n - 1):
        before = last_value
        last_value = actual_value + last_value
        actual_value = before
    #actual_value, last_value = last_value, actual_value + last_value
    return actual_value
```



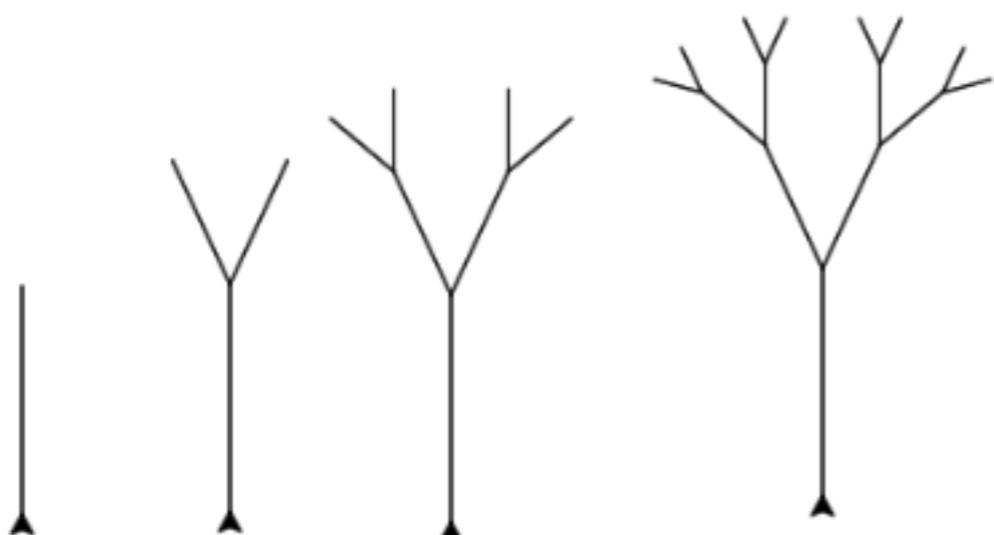
RECUSIÓN - FIBONACCI

```
def fibonacci_r(n):
    if n == 1 or n == 2:
        return 1
    return fibonacci_r(n-1) + fibonacci_r(n-2)
```

RETO GRAFICO 3: ARBOL

COMPLETA LA FUNCIÓN **TREE** GENERA LA UNA IMAGEN DE UN ÁRBOL DE N NUMERO DE NIVELES,
TREE RECIBE 2 PARÁMETROS OBLIGATORIOS Y UNO OPCIONAL:

- ▶ SIZE: DETERMINA LA LONGITUD INICIAL DE LA RAMA DEL ARBÓL, PARA CADA NIVEL LA LONGITUD DE LA RAMA DISMINUYE A 0.6 DE LA LONGITUD ANTERIOR
- ▶ LEVELS: LA CANTIDAD DE NIVELES
- ▶ ANGEL: EL ÁNGULO EN QUE SE ABREN LAS RAMAS PARA CADA NIVEL, ES UN ARGUMENTO OPCIONAL POR DEFAULT ES 25°
- ▶ SE DEBE DE RESOLVER SI O SI CON RECURSIÓN
- ▶ EL TURTLE INICIA Y TERMINA EN LA MISMA POSICIÓN



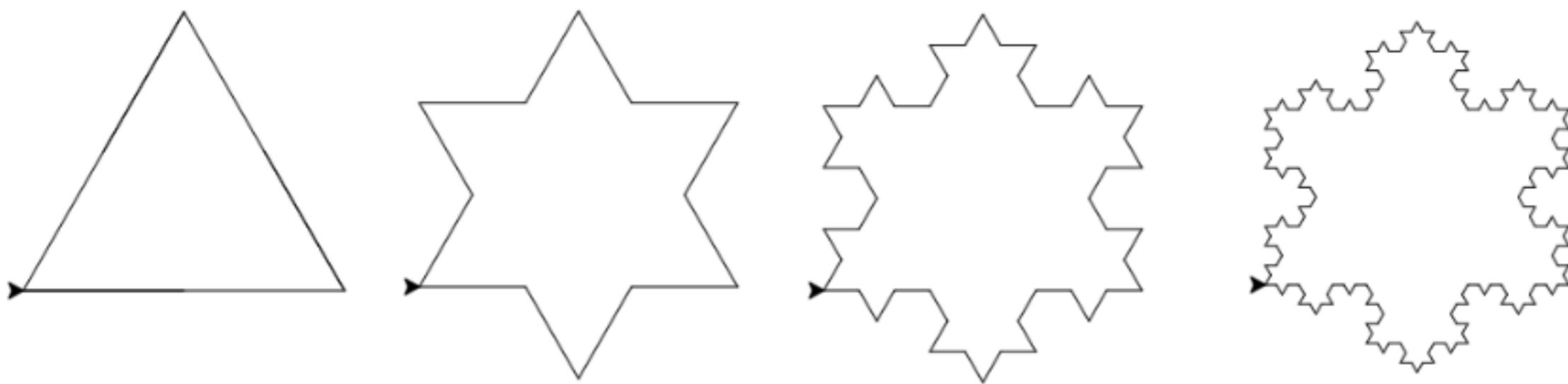
RETO GRAFICO 3: ARBOL



RETO GRAFICO 4: COPO DE NIEVE

COMPLETA LA FUNCIÓN SNOWFLAKE GENERA LA IMAGEN DE UN COPO DE NIEVE DE NÚMERO DE NIVELES, SNOWFLAKE RECIBE 2 PARÁMETROS:

- ▶ SIDE_SIZE: DETERMINA LA LONGITUD INICIAL CADA LADO DEL COPO
- ▶ LEVELS: LA CANTIDAD DE NIVELES, PARA CADA NIVEL SE REDUCE EN 1/3 EL TAMAÑO DEL SIDE_SIZE
- ▶ SE DEBE DE RESOLVER SI O SI CON RECURSIÓN
- ▶ EL TURTLE INICIA Y TERMINA EN LA MISMA POSICIÓN



copo de nieve

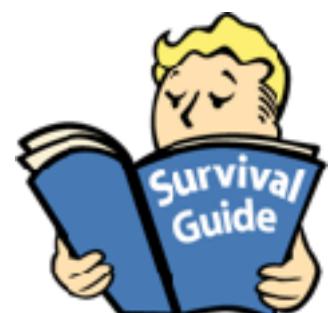
RETO GRAFICO 4: COPO DE NIEVE



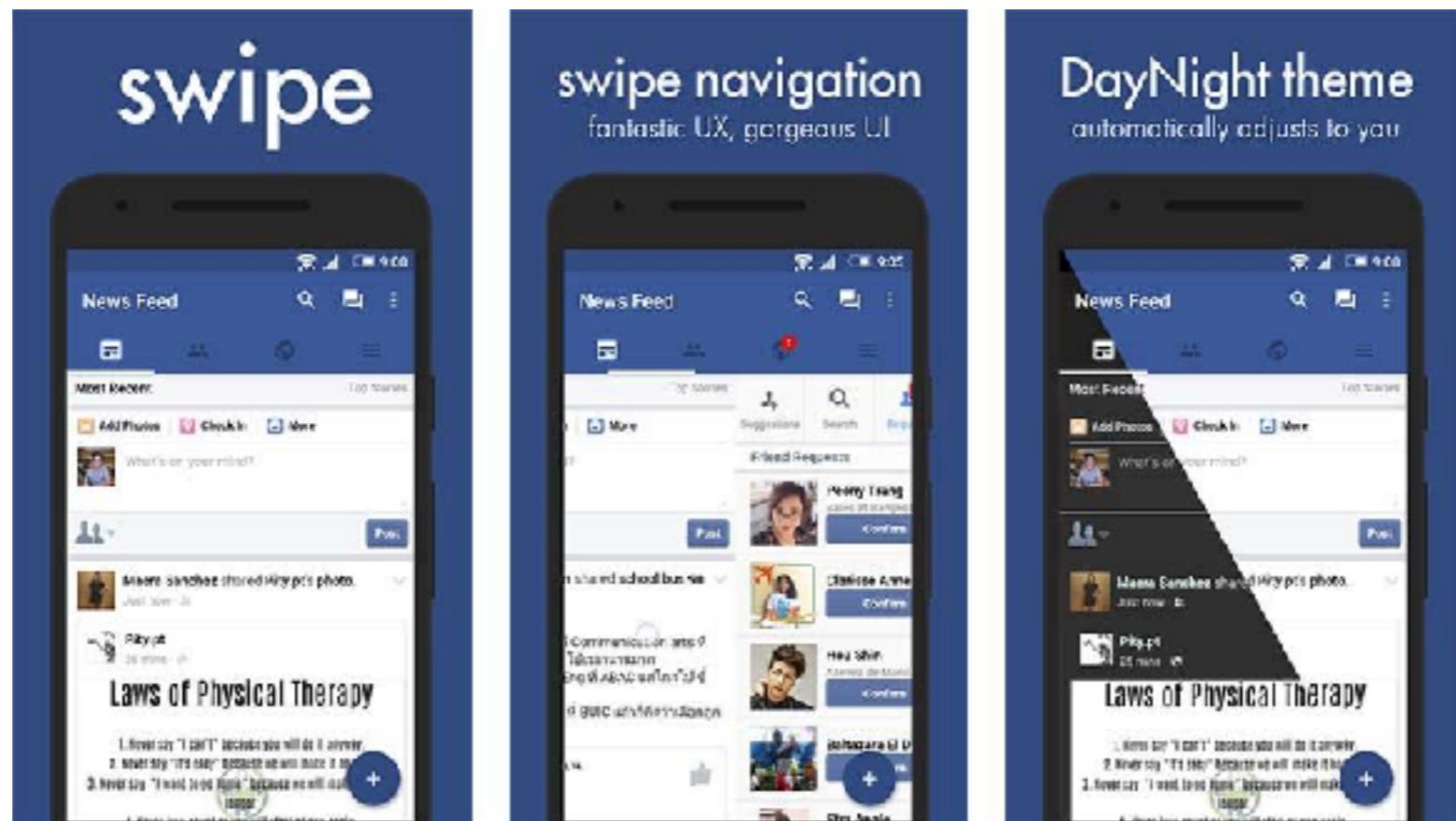
copo de nieve

MEMORIZACIÓN

MEMORIZACIÓN ES UNA TÉCNICA DE OPTIMIZACIÓN UTILIZADA PRINCIPALMENTE PARA ACELERAR LOS PROGRAMAS INFORMÁTICOS ALMACENANDO LOS RESULTADOS DE COSTOSAS LLAMADAS DE FUNCIÓN Y DEVOLVIENDO EL RESULTADO ALMACENADO EN **CACHÉ** CUANDO VUELVEN A PRODUCIRSE LAS MISMAS ENTRADAS.



CACHE APPS



CACHE WEB



- 1. ALGORITMOS**
- 2. BIG O NOTATION**
- 3. GRÁFICAS DE BIG O**
- 4. COMPLEJIDAD DE OPERACIONES EN PYTHON**
- 5. METODOLOGIAS PARA RESOLVER ALGORITMOS**
- 6. RECURSIVIDAD**
- 7. MEMORIZACIÓN**