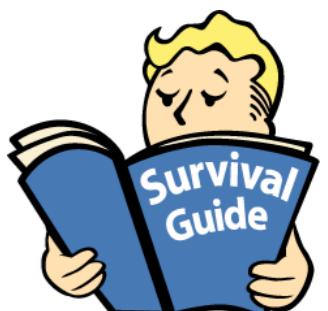


TÉCNICAS DE PROGRAMACIÓN

ORDENAMIENTO Y BÚSQUEDA

FUNCIONES ANÓNIMAS

```
add = lambda x, y: x + y  
print(add(3, 5)) # 8
```



MAP

MAP APLICA UNA FUNCIÓN A TODOS LOS ELEMENTOS DE UNA LISTA

```
items = [1, 2, 3, 4, 5, 6]
```

```
def multiply_elements_by_two(numbers):
    result = []
    for number in numbers:
        result.append(number * 2)
    return result
result = multiply_elements_by_two(items)
```

```
result = list(map(lambda x: x * 2, items))
```

FILTER

**FILTER CREA UNA LISTA DE ELEMENTOS PARA LOS QUE UNA
FUNCIÓN DEVUELVE TRUE**

```
items = [1, 2, 3, 4, 5, 6]
def get_low_than_three(numbers):
    result = []
    for number in numbers:
        if number < 3:
            result.append(number)
    return result
result = get_low_than_three(items)

result = list(filter(lambda x: x < 3, items))
```

REDUCE

REDUCE ES UNA FUNCIÓN APLICA UN CÁLCULO EN UNA LISTA Y DEVOLVIENDO UN ÚNICO RESULTADO.

```
items = [1, 2, 3, 4, 5, 6]

def get_average(numbers):
    result = 0
    for number in numbers:
        result += number
    return result / len(numbers)

average = get_average(items)

from functools import reduce
average = reduce((lambda x, y: x * y), items) / len(items)
```

REDUCE

REDUCE ES UNA FUNCIÓN APLICA UN CÁLCULO EN UNA LISTA Y DEVOLVIENDO UN ÚNICO RESULTADO.

```
items = [1, 2, 3, 4, 5, 6]

def get_average(numbers):
    result = 0
    for number in numbers:
        result += number
    return result / len(numbers)

average = get_average(items)

from functools import reduce
average = reduce((lambda x, y: x * y), items) / len(items)
```

USALOS CON CUIDADO

```
findShort = (s) => Math.min.apply(null, s.split(' ').map( (i) => i.length ))
```

```
function findShort (sentence) {
  words = sentence.split(' ')
  minWordSize = words[0].length

  words.forEach( word => {
    wordSize = word.length
    minWordSize = wordSize < minWordSize ? wordSize : minWordSize
  })

  return minWordSize
}
```



ALGORITMOS DE ORDENAMIENTO

SECTION SORT

1. DIVIDE LA INFORMACIÓN EN ORDENADO Y NO ORDENADO
2. ENCUENTRA EL ELEMENTO MÍNIMO DE LA SECCIÓN NO ORDENADA
3. AGRÉGALO A LA SECCIÓN ORDENADA



```
def section_sort(numbers):  
    result = numbers.copy()  
    for i in range(len(result)):  
        min_index = i  
        for j in range(i + 1, len(result)):  
            if result[min_index] > result[j]:  
                min_index = j  
        result[i], result[min_index] = result[min_index], result[i]  
    return result
```

$O(N^2)$

BUBBLE SORT

1. INICIA EL INDICE i EN 0
2. COMPARARA EL ELEMENTO $[i]$ E $[i + 1]$
3. SI EL ELEMENTO $[i + 1] > [i]$ INTERCAMBIAMOS DE LUGAR
4. REPITES CUANTAS VECES SEA NECESARIO

5 2 4 6 1 3

```
def bubble_sort(numbers):  
    result = numbers.copy()  
    size = len(numbers)  
  
    for i in range(size):  
        for j in range(0, size - i - 1):  
            if result[j] > result[j + 1]:  
                result[j], result[j + 1] = result[j + 1], result[j]  
    return result
```

$O(N^2)$

INSERTION SORT

1. INICIA EL INDICE **I** EN 0
2. COMPARA CON EL SIGUIENTE ELEMENTO
3. SI ES MAYOR INCREMENTA EL INDICE
4. SI ES MENOR TOMA EL NUMERO MENOR Y ENCUENTRA SU LUGAR EN UN INDICE ANTERIOR

6 5 3 1 8 7 2 4

```
def insertion_sort(numbers):  
    result = numbers.copy()  
    for i in range(1, len(result)):  
        key = result[i]  
        j = i - 1  
        while j >= 0 and key < result[j] :  
            result[j+1] = result[j]  
            j -= 1  
        result[j + 1] = key  
    return result
```

$O(N^2)$

MERGE SORT

1. DIVIDE EL ARREGLO EN DOS
2. SE PUEDE SEGUIR DIVIDIENDO EN 2, DIVIDE
3. JUSTA Y ORDENA

6 5 3 1 8 7 2 4

MERGE SORT

```
def merge(left, right):  
    left_size = len(left)  
    right_size = len(right)  
    if not left_size or not right_size:  
        return left or right  
  
    result = []  
    i, j = 0, 0  
    while (len(result) < left_size + right_size):  
        if left[i] < right[j]:  
            result.append(left[i])  
            i += 1  
        else:  
            result.append(right[j])  
            j += 1  
        if i == left_size or j == right_size:  
            result.extend(left[i:] or right[j:])  
            break  
    return result
```

```
def merge_sort(numbers):  
    if len(numbers) < 2:  
        return numbers  
    middle = len(numbers) // 2  
    left = merge_sort(numbers[:middle])  
    right = merge_sort(numbers[middle:])  
    return merge(left, right)
```

O(N LOG N)

QUICK SORT

1. ELEGIR UN ELEMENTO ALEATORIO DEL MEDIO DEL ARREGLO
2. RESITUA LOS DEMÁS ELEMENTOS DE LA LISTA A CADA LADO DEL PIVOTE MENORES A LA IZQUIERDA, MAYORES A LA DERECHA
3. REPITE CON LAS DOS NUEVAS LISTA IZQUIERDA Y DERECHA



QUICK SORT

```
def quick_sort(numbers):  
    size = len(numbers)  
    if size == 1:  
        return numbers  
  
    middle = size // 2  
    pivot = numbers[middle]  
    low_numbers = []  
    high_numbers = []  
  
    for index, number in enumerate(numbers):  
        if index == middle:  
            continue  
        if number > pivot:  
            high_numbers.append(number)  
        else:  
            low_numbers.append(number)  
    return quick_sort(low_numbers) + [pivot] + quick_sort(high_numbers)
```

$O(N \log N)$

ALGORITMOS DE BÚSQUEDA

BÚSQUEDA LINEAL

Linear Search



```
def search(arr, x):  
    for i in range(len(arr)):  
        if arr[i] == x:  
            return i  
    return -1
```

O(N)

BÚSQUEDA BINARIA

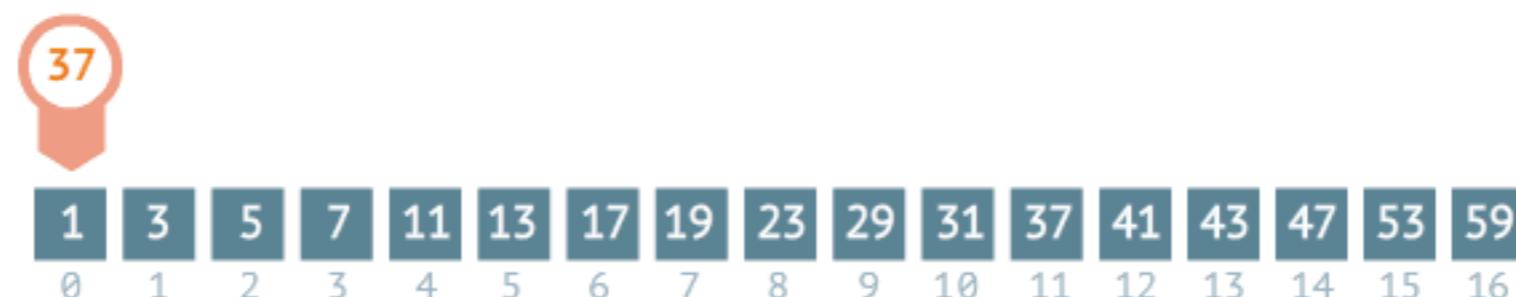
Binary search

steps: 0



Sequential search

steps: 0



ARCHIVOS

ARCHIVOS

SIMPLE TEXT FILE

.TXT

```
data_file = open('testfile.txt', 'w')
data_file.write('Hello World')
data_file.close()
```

MODES FILE

Modo	Descripción
r	Read Only
rb	Read Only Binary
r+	Read and Write, el puntero aparece al inicio del archivo
rb+	Read and Write Binary, el puntero aparece al inicio del archivo
w	Write Only, sobre escribe todo el archivo, crea el archivo si no existe
wb	Write Only Binary, sobre escribe todo el archivo, crea el archivo si no existe
w+	Read and Write, sobre escribe todo el archivo, crea el archivo si no existe
wb+	Read and Write Binary, sobre escribe todo el archivo, crea el archivo si no existe
a	Append, el puntero esta al final de archivo, si el archivo no existe lo crea.
ab	Append Binary, el puntero esta al final de archivo, si el archivo no existe lo crea.
a+	Append and Read Binary, el puntero esta al final de archivo, si el archivo no existe lo crea.
ab+	Append and Read Binary, el puntero esta al final de archivo, si el archivo no existe lo crea.

COMMA-SEPARATED VALUES FILE

.CSV

JAVA SCRIPT OBJECT NOTATION

.JSON

NOTA: EL MANEJO DE JSON EN PYTHON ES A TRAVÉS DE UN PARSER

JSON

```
{  
  "name": "Miguel",  
  "age": 25,  
  "is_alive": True,  
  "friends": ["Hugo", "Paco", "Luis"],  
  "kids": None  
}
```

```
{  
  "name": "Miguel",  
  "age": 25,  
  "is_alive": true,  
  "friends": ["Hugo", "Paco", "Luis"],  
  "kids": null  
}
```

RETO 10: GAME RANKING

SUPONIENDO QUE TRABAJAS EN UNA EMPRESA DE VIDEO JUEGOS QUE TIENE UN VIDEO JUEGO MUY EXITOSO CON UNA GRAN CANTIDAD DE USUARIOS.

SE TE DA LA TAREA DE MEJORAR EL ALGORITMO DE ORDENAMIENTO DEL RANKING DE LOS JUGADORES, YA QUE SE ESTÁN QUEJANDO QUE EL RANKING SE ACTUALIZA MUY LENTO. ACTUALMENTE SE ESTÁ UTILIZANDO UN ALGORITMO DE $O(N \log N)$.

- ▶ GENERAL UNA FUNCIÓN LLAMADA **SORT_BY_SCORE**, LA CUAL DEBE DE RECIBIR UN ARREGLO DE DICCIONARIOS CON LA INFORMATION DE CADA USUARIO Y RETORNA UN ARREGLO DE DICCIONARIOS ORDENADOS
- ▶ DEBE OPERAR EN $O(N)$
- ▶ ACTUALMENTE LA BASE DE DATOS DE LA EMPRESA TE REGRESA UN ARCHIVO **.CSV**, LLAMADO **USERS . CSV**
- ▶ TE PIDEN ALMACENAR EL RESULTADO ORDENADO COMO UN ARCHIVO EN FORMATO **.JSON**, CON EL NOMBRE **SORTED_USERS . JSON**



AMOR Y DINERO

AMOR Y DINERO

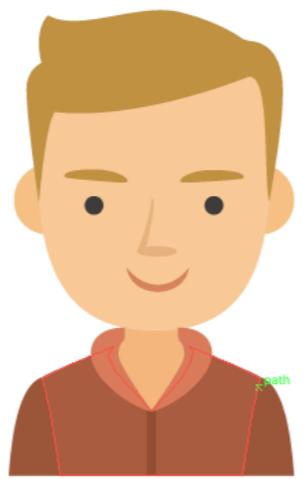
PROBLEMA DE LAS PAREJA ESTABLE



PROBLEMA DE LAS PAREJA ESTABLE



PAULA



ROMEO



JUELIETA



JORGE

- 1. MISMO NUMERO DE HOMBRES QUE DE MUJERES**
- 2. TODOS DEBEN TENER PAREJA**
- 3. NO CREAR PAREJAS INESTABLES**
- 4. SOLO MATCH ENTRE GÉNEROS**

PROBLEMA DE LAS PAREJA ESTABLE



MARCO

MARIA
SOFIA
ANA
JULIETA



JUAN

MARIA
JULIETA
SOFIA
ANA



HUGO

MARIA
SOFIA
ANA
JULIETA



PEDRO

MARIA
JULIETA
SOFIA
ANA



MARIA

MARCO
JUAN
HUGO
PEDRO



SOFIA

MARCO
JUAN
HUGO
PEDRO



ANA

JUAN
HUGO
PEDRO
MARCO



JULIETA

PEDRO
JUAN
HUGO
JUAN

DIA 1

DIA 2

DIA 3

DIA 4

DIA 5

MARIA

SOFIA

ANA

JULIETA

RETO 11: PAREJA FELIZ

**IMPLEMENTA EL ALGORITMO VISTO EN CLASE PARA HACER MATCH Y RESPONDE EL COMENTARIO
SEPARA LAS FUNCIONES DE MATCH DE LAS FUNCIONES DE MANEJO DE ARCHIVOSCOMPLETA EL
COMENTARIO FINAL**

SI LAS MUJERES PROPONEN

'ALLISON ESTRADA CARR' ->

'NIXON LEONARD ROSE' ->

'JEANNIE MORTON NOBLE' ->

SI LOS HOMBRES PROPONEN

'WILLIAM FRANCIS BENJAMIN' ->

'BUTLER MATTHEWS MUÑOZ' ->

'HOLLAND JENNINGS ENGLISH' ->



Pareja Feliz

ALGORITMOS DE CIFRADO Y LOS DINEROS

SSH



HTTPS

SCL



SHA-1 SECURE HASH ALGORITHM

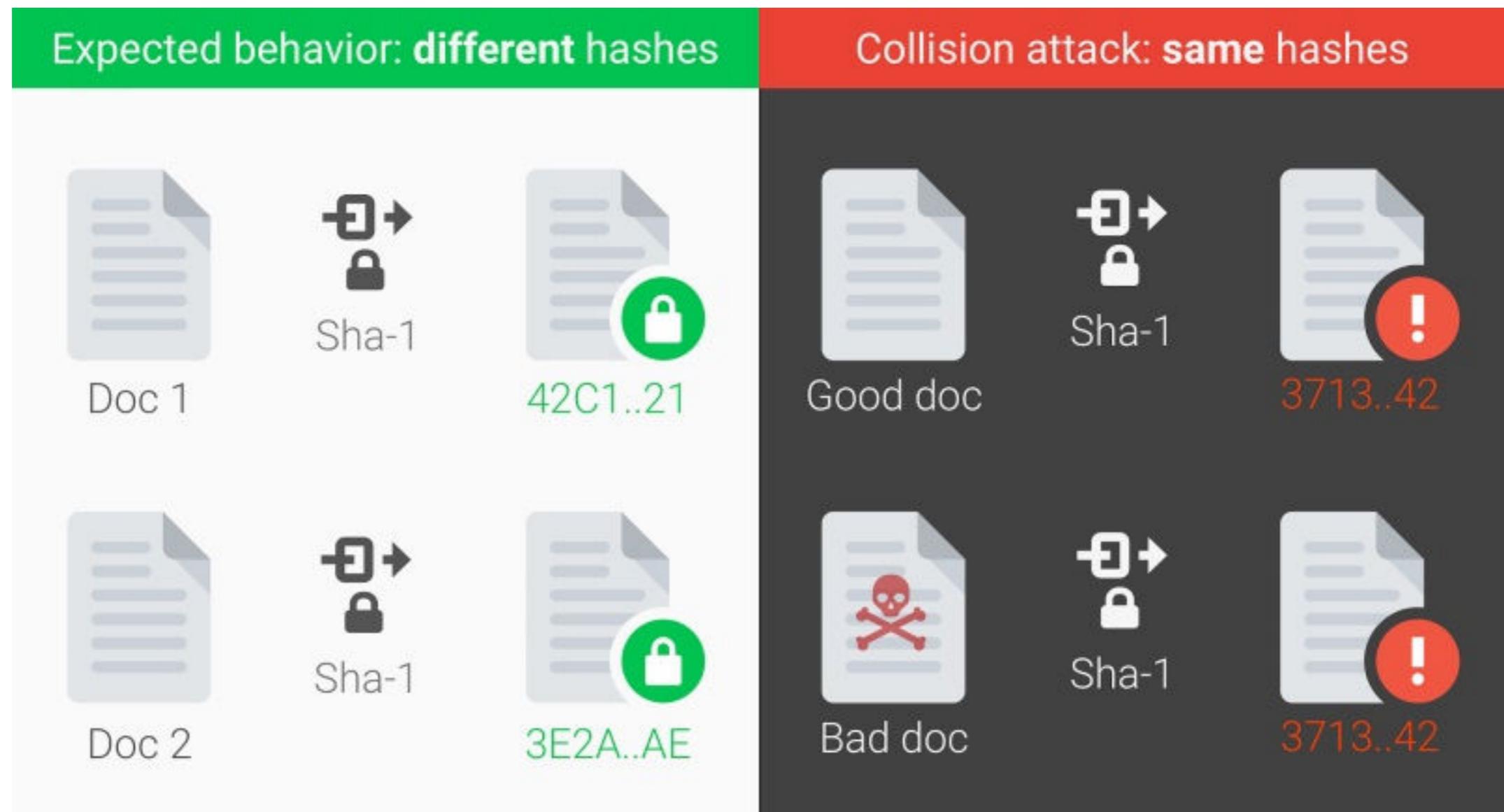
HOLA MAMA → AE23JV232

ESTE ES OTRO UN MENSAJE → WF53D62D

HOLA MAMA. → GSGJ29D3

LA HUELLA DE UN ARCHIVO

ALGORITMOS DE CIFRADO



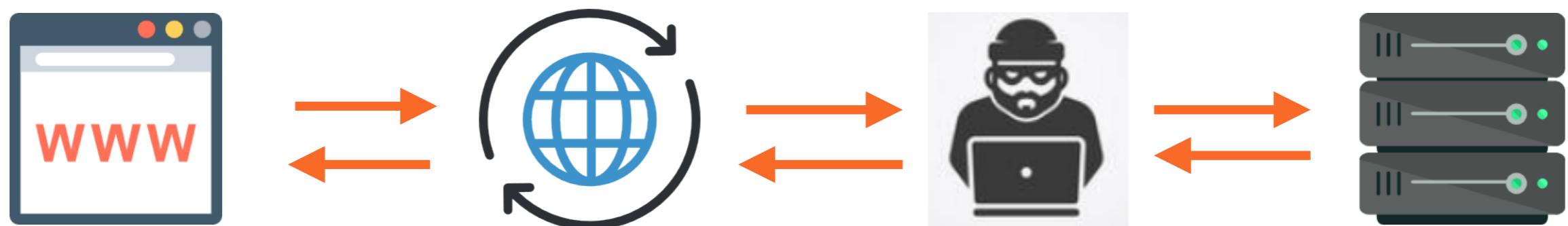
GUARDAR CONTRASEÑAS

VERIFICAR ARCHIVOS

ALGORITMOS DE CIFRADO



ALGORITMOS DE CIFRADO

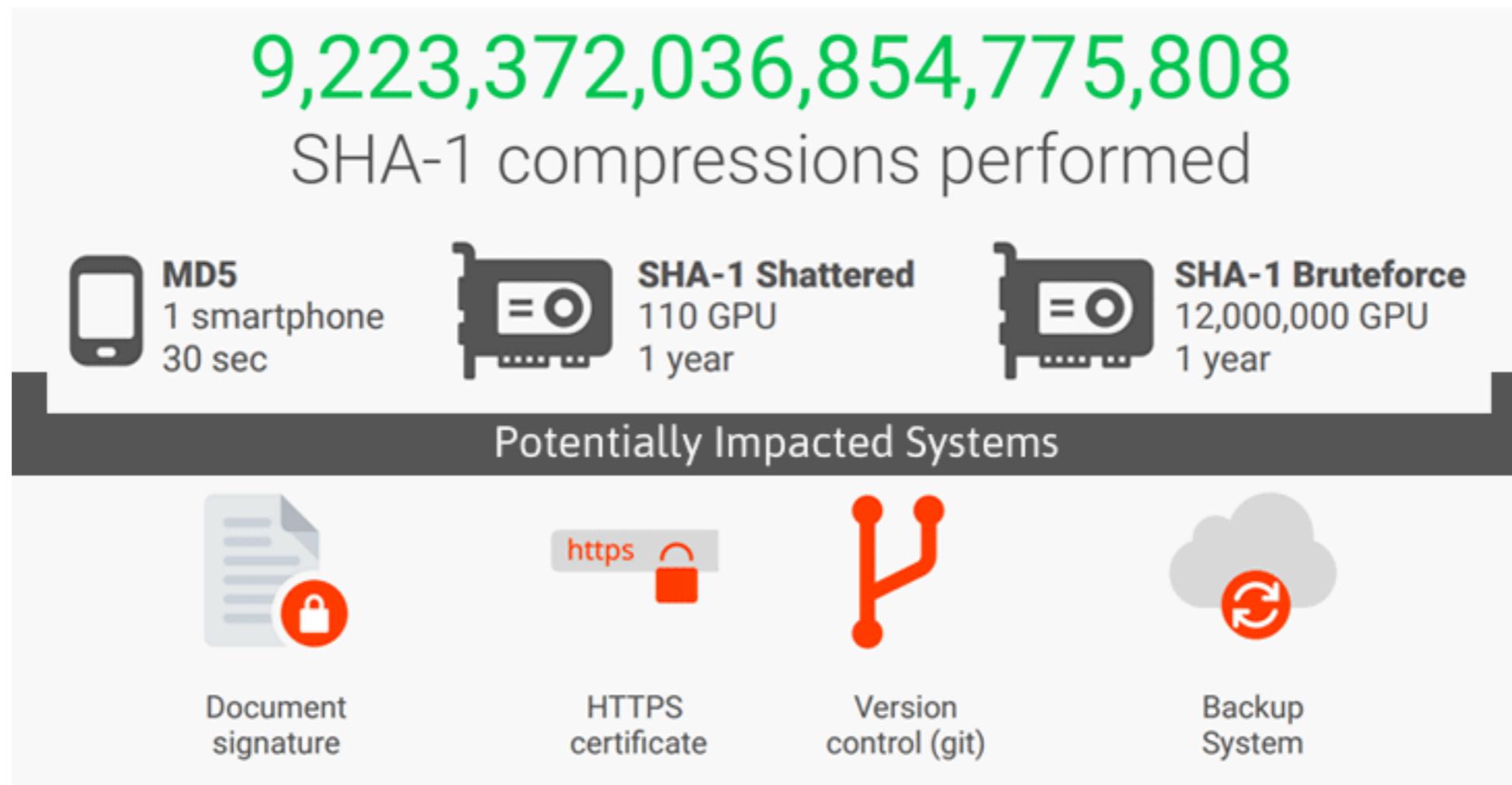


ALGORITMOS DE CIFRADO



**¿ENTONCES QUE ENCONTRO GOOGLE?
¿COMO LE HIZO?**

ALGORITMOS DE CIFRADO



ALGORITMOS DE CIFRADO

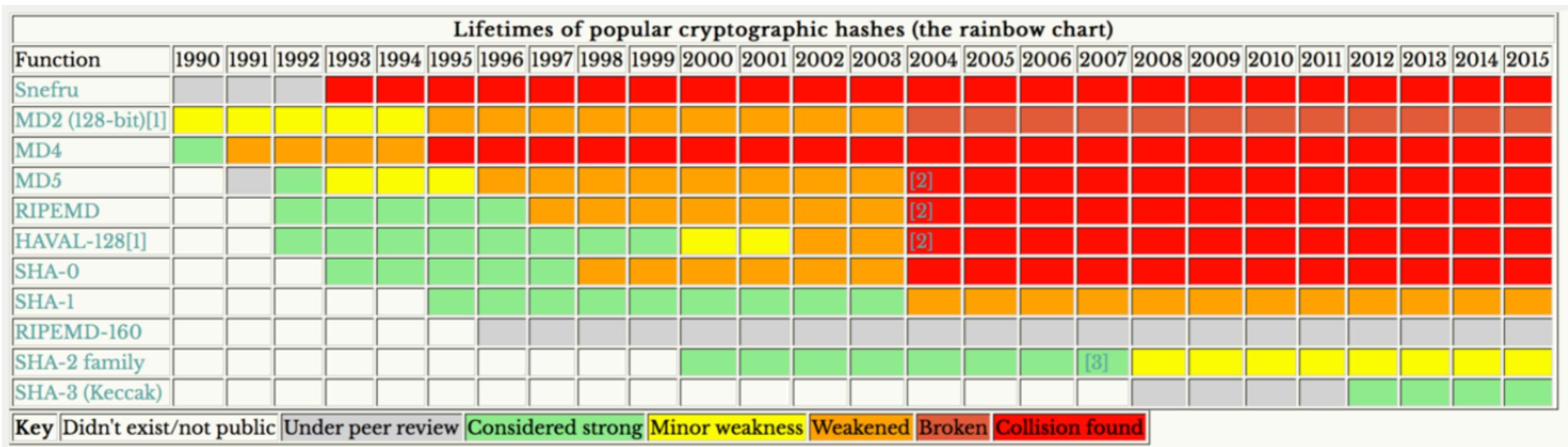
HTTPS://ALF.NU/SHA1

The diagram shows the hex dump of two PDF files, shattered-1.pdf and shattered-2.pdf, with annotations explaining the structure and differences between them.

Annotations:

- PDF Header:** Located at the top of both files.
- JPEG Start:** Indicated by green boxes and arrows pointing to the start of JPEG data blocks.
- JPEG Comment:** Indicated by blue boxes and arrows pointing to large comments within the JPEG data blocks.
- Comment length = 0x173:** A red box highlights the length of a specific comment in shattered-1.pdf.
- Comment length = 0x17F:** A red box highlights the length of a specific comment in shattered-2.pdf.
- Collision blocks:** A yellow box highlights the difference between the two files in the middle section.
- Desync:** A blue box highlights a section where the parsing gets out of sync.
- Interleaving:** A blue box highlights interleaved comments on the right side.
- Real JPEG data starts much later...**: A yellow box highlights the delay before actual JPEG data begins.
- Variable:** A green box highlights variable sections in the middle of the files.
- JFIF Header:** Indicated by a green box and arrow pointing to the JFIF header in shattered-2.pdf.
- Quantization table:** Indicated by a green box and arrow pointing to the quantization table in shattered-2.pdf.
- SOF2 header:** Indicated by a green box and arrow pointing to the SOF2 header in shattered-2.pdf.
- Huffman tables:** Indicated by a green box and arrow pointing to the Huffman tables in shattered-2.pdf.
- JPEG Comment:** Indicated by a blue box and arrow pointing to a comment in shattered-2.pdf.
- Image data:** Indicated by a blue box and arrow pointing to the image data in shattered-2.pdf.

ALGORITMOS DE CIFRADO



¿POR QUE ES PREOCUPANTE?

ALGORITMOS DE CIFRADO



ALGORITMOS DE CIFRADO



RSA-2048
US\$200,000

```
251959084756578934940271832400483985714292821262040320277713783604366202070  
7595556264018525880784406918290641249515082189298559149176184502808489120072  
8449926873928072877767359714183472702618963750149718246911650776133798590957  
0009733045974880842840179742910064245869181719511874612151517265463228221686  
9987549182422433637259085141865462043576798423387184774447920739934236584823  
8242811981638150106748104516603773060562016196762561338441436038339044149526  
3443219011465754445417842402092461651572335077870774981712577246796292638635  
6373289912154831438167899885040445364023527381951378636564391212010397122822  
120720357
```

- 1. ALGORITMOS DE ORDENAMIENTO**
- 2. ALGORITMOS DE BÚSQUEDA**
- 3. MANEJO Y TIPOS DE ARCHIVOS**
- 4. OTROS ALGORITMOS**