Ejercicios

0.8 Estructuras de datos y algoritmos

Introducción

La utilización de algoritmos es común a todos los trabajos relacionados con la programación y la gestión de datos. A veces los implementamos nosotros, y otras veces nos apoyamos en sistemas o librerías que ya los tienen implementados.

Es importante entender cómo funcionan, cómo evaluarlos, y cómo utilizarlos. Algo que se aplica también a los algoritmos que implementamos nosotros mismos.

Calcular el orden de complejidad de un algoritmo puede ser complicado, pero hay varias pistas que nos pueden ayudar:

- Si el algoritmo encuentra el valor deseado sin necesidad de iterar sobre los datos, entonces es probable que sea O(1)

- Si realiza una iteración sobre los datos (bucle for) entonces probablemente sea O(n)

    - Para n = 1 ejecutamos x instrucciones
    - Para n = 5 ejecutamos 5x instrucciones

- Si debe realizar una iteración anidada dentro de otra (un bucle dentro de otro bucle) entonces probablemente sea O(n2)

    - Para n = 1 ejecutamos x instrucciones
    - Para n = 5 ejecutamos 25x instrucciones (52)

El orden de complejidad nos da una función genérica sobre cuántas instrucciones hay que ejecutar dependiendo de los datos de entrada. O dicho de otro modo, cómo crece el número de órdenes a ejecutar con respecto a cómo crece el número de datos de entrada.

Entrega

- Cada ejercicio deberá entregarse en un archivo JavaScript (.js) subido a la plataforma del curso.
- La entrega deberá ir acompañada de al menos un diagrama de flujo en formato imagen (se recomienda usar un software de diagramas como Draw.io o LucidChart) que represente el código de la solución de un ejercicio. El ejercicio a escoger se deja a criterio del alumno. Si se entregan más diagramas todos serán corregidos, pero al menos debe entregarse uno. Si no se entrega ningún diagrama la entrega se dará por insatisfecha.

Evaluación

- Se evaluarán los siguientes puntos:
- Que el código funcione (que no de error).
- Que el código satisfaga el enunciado.
- Que el código no tenga redundancias
- Que el código esté correctamente indentado.
- Que el diagrama sea claro, correcto, y represente el código fielmente.

Ejercicios

0.8 Estructuras de datos y algoritmos

Ejercicios

Ejercicio 1
Implementa una función que reciba un texto y devuelva las palabras que se repiten en dicho texto

Ejemplo:

```
$> node ejercicio1.js

Introduce un texto: Recordar es fácil para quien tiene memoria, olvidar es
difícil para quien tiene corazón

Se repiten las siguientes palabras: es, para, quien, tiene

$>
```

Ejercicio 2

Implementa una función que reciba dos arrays de números y devuelva un array con los elementos que están en el primer array pero que no están en el segundo. Pruébala con valores aleatorios.

Ejemplo:

```
$> node ejercicio2.js

Array 1: [1, 2, 3, 4, 5]
Array 2: [3, 5]

La diferencia de ambos conjuntos es [1, 2, 4]

$>
```

Ejercicio 3

Implementa una función que reciba una cadena y devuelva un Map con la frecuencia de cada carácter en la cadena (las veces que se repite).

Ejemplo:

```
$> node ejercicio3.js

Introduce un texto: Hola mundo

Mapa de frecuencias:
Map { 'h' => 1, 'o' => 2, 'l' => 1, 'a' => 1, ' ' => 1, 'm' => 1, 'u' => 1, 'n'
=> 1, 'd' => 1 }

$>
```

Ejercicios

0.8 Estructuras de datos y algoritmos

Ejercicio 4

Escribe una función que reciba dos cadenas y verifique si son anagramas utilizando un Map.

Ejemplo:

```
$> node ejercicio4.js

Introduce una palabra: amor
Introduce otra palabra: roma

Las palabras son anagramas.

$>
```

Ejercicio 5

Implementa una función que invierta el orden de una palabra usando una pila (stack).

Ejemplo:

```
$> node ejercicio5.js

Introduce una palabra: amor

La palabra invertida es “roma”.

$>
```

Ejercicio 6:

Implementa el algoritmo de ordenación bubble sort y comprueba cuánto tarda en ejecutarse sobre un array de números aleatorios de tamaño 100.000.

Ejemplo:

```
 $> node ejercicio6.js

El algoritmo bubble sort ha tardado 12 segundos en ordenar 100000 elementos.

$>
```

Ejercicios

0.8 Estructuras de datos y algoritmos

Ejercicio 7

Implementa el algoritmo de ordenación quicksort y comprueba cuánto tarda en ejecutarse sobre un array de números aleatorios de tamaño 100.000

Ejemplo:

```
$> node ejercicio7.js

El algoritmo quicksort ha tardado 0.6 segundos en ordenar 100000 elementos.

$>
```

Ejercicio 8

Implementa una función que reciba un número y devuelva si dicho número es primo o no.

Ejemplo:

```
$> node ejercicio8.js

Introduce un número: 1009

1009 es primo.

$>
```

Ejercicio 9

Implementa una función que, dado un texto con paréntesis, indique si dichos paréntesis están balanceados (se cierran todos los que se abren),

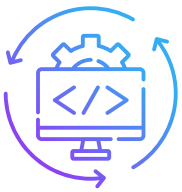
Ejemplos:

```
$> node ejercicio9.js

Introduce un texto: (()(()))

Están balanceados.

$>
```




```
$> node ejercicio9.js

Introduce un texto: )(()()))

No están balanceados.

$>
```

Ejercicio 10

Implementa una función que verifique si una secuencia de paréntesis, corchetes, y llaves es válida.
Es decir, se abren tantos como se cierran y cada secuencia está siempre contenida.

Ejemplos:

```
$> node ejercicio10.js

Introduce una secuencia: [()]{}()

Secuencia correcta.

$>
```

```
$> node ejercicio9.js

Introduce una secuencia: [(]{)}

Secuencia incorrecta.

$>
```