Estado	Finalizado
Comenzado	jueves, 20 de marzo de 2025, 20:24
Completado	jueves, 20 de marzo de 2025, 20:58
Duración	34 minutos 32 segundos
Calificación	**3.3** de 10.0 (**32.5**%)

Pregunta 1

Incorrecta

Se puntúa -0.3 sobre 1.0

¿Qué es una variable?

a. Un nombre que representa la dirección de un dato en memoria.

b. Un nombre que se asigna a un valor.

c. Todas las anteriores ✕

d. Un valor en cualquier lenguaje de programación.

e. Un nombre que representa un dato.

Respuesta incorrecta.

La respuesta correcta es: Un nombre que representa la dirección de un dato en memoria.

Pregunta 2

Correcta

Se puntúa 1.0 sobre 1.0

Los arreglos son mucho mas eficientes por que reducen el espacio en memoria, espacios que se ocuparían al usar múltiples declaraciones de variables.

Verdadero

Falso ✓

Bien

La respuesta correcta es 'Falso'

Pregunta 3

Correcta

Se puntúa 1.0 sobre 1.0

¿La siguiente función **retorna** la suma de dos números que se reciben cómo parámetro?

```
const add = (a, b) => {
  console.log(a + b);
};
```

- Verdadero
- Falso ✓

La respuesta correcta es 'Falso'

Pregunta 4

Correcta

Se puntúa 1.0 sobre 1.0

La notación Big O se encarga de medir solamente la eficiencia temporal.

- Verdadero
- Falso ✓

La respuesta correcta es 'Falso'

Pregunta 5

Incorrecta

Se puntúa -0.2 sobre 1.0

Determine la complejidad temporal del siguiente algoritmo:

```
1  int sum = 0;
2
3  for (int val : arrayA) {
4    sum += val;
5  }
6
7  for (int val : arrayB) {
8    sum += val;
9  }
```

- a. O (2 n)
- b. O (n + n)
- c. O (n) ×
- d. O (A + B)

Respuesta incorrecta.

La respuesta correcta es: O (A + B)

Pregunta 6

Incorrecta

Se puntúa -0.1 sobre 1.0

Cual es la complejidad del siguiente algoritmo:

```
const array = [1, 2, 3, 4, 12];

for (i = 1; i < array.length; i++) {
  let current = array[i];
  let j = i - 1;

  while (j >= 0 && array[j] > current) {
    array[j + 1] = array[j];
    j--;
  }

  array[j + 1] = current;
  i = i + 1;
}
```

a. O (2 n)

b. O (nlogn)

c. O (n ^ 2) ✕ El arreglo ya esta organizado por lo tanto es O (n)

d. O (n)

Respuesta incorrecta.

La respuesta correcta es: O (n)

Pregunta 7

Incorrecta

Se puntúa -0.2 sobre 1.0

Para el siguiente arreglo:

const disorderArray = [33, 4, 3, 1, 21, 100, 23];

Que método de ordenamiento escogería usted, teniendo en cuenta el rendimiento temporal.

a. Quick sort ✕ Cualquiera, para N pequeños todos los algoritmos son rápidos

b. Cualquiera

c. Inserción

d. Burbuja

Respuesta incorrecta.

La respuesta correcta es: Cualquiera

Pregunta 8

Finalizado

Se puntúa 1.0 sobre 3.0

Calcule la complejidad temporal del siguiente algoritmo. Justifique cada decisión.

Importante: Respuestas no legibles no serán tenidas en cuenta.

```python
def find_index(sorted_list, target):
    left = 0
    right = len(sorted_list) - 1

    while left <= right:
        mid = (left + right) // 2
        if sorted_list[mid] == target:
            return mid
        elif sorted_list[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1
```

El algoritmo utiliza la busqueda binaria, lo cual implica en dividir en conjunto de datos en la mitad en cada interación

```
// Inicialización de los índices que delimitan los extremos del intervalo
let left = 0; // Complejidad: O(1), inicialización de una variable
let right = sorted_list.length - 1; // Complejidad: O(1), cálculo de longitud del arreglo

// Bucle que se repite mientras el intervalo sea válido (no vacío)
while (left <= right) { // Complejidad: O(log n), el bucle se ejecuta log🤏 veces
  // Cálculo del índice medio del intervalo
  let mid = Math.floor((left + right) / 2); // Complejidad: O(1), operación matemática constante

  // Verificación: ¿el elemento medio coincide con el objetivo?
  if (sorted_list[mid] == target) { // Complejidad: O(1), comparación constante
    return mid; // Complejidad: O(1), retorno inmediato
  }

  // Si el elemento medio es menor que el objetivo, buscamos en la mitad derecha
  else if (sorted_list[mid] < target) { // Complejidad: O(1), comparación constante
    left = mid + 1; // Complejidad: O(1), asignación de valor constante
  }

  // Si el elemento medio es mayor que el objetivo, buscamos en la mitad izquierda
  else {
    right = mid - 1; // Complejidad: O(1), asignación de valor constante
  }
}
// Si salimos del bucle, significa que el objetivo no está en el arreglo
return -1; // Complejidad: O(1), retorno constante
```

El bucle principal (while) realiza $\log_2(n)$ iteraciones en el peor de los casos, dado que cada iteración reduce el tamaño del intervalo de búsqueda a la mitad.

Cada línea dentro del bucle tiene una complejidad de O(1), ya que implica operaciones simples como asignaciones, comparaciones, y cálculos matemáticos.

Por lo tanto, la complejidad total es O(log n), al combinar el número de iteraciones del bucle con las operaciones constantes en cada iteración.

Comentario:
No era explicar el algoritmo, era calcular la complejidad para ello

Debía plantear la ecuación:

(n/(2^k)) = 1

◄ Avisos

Ir a...

Parcial 1 práctico ►