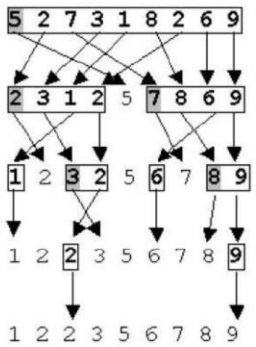
Actividad #3 de la Semana 3 Investigación sobre Algoritmos de búsqueda y Ordenamiento

Realiza una investigación detallada sobre dos algoritmos clave en el ámbito de la programación: QuickSort (algoritmo de ordenamiento) y Binary Search (algoritmo de búsqueda binaria). Explora sus características, funcionamiento y aplicaciones prácticas.

1. Algoritmo de ordenamiento Quick Sort:
El método de ordenamiento QuickSort es actualmente el más eficiente y veloz de los métodos de ordenación interna.
Este método es una mejora sustancial del método de intercambio directo y recibe el nombre de QuickSort por la velocidad con que ordena los elementos de un arreglo.
Es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar “n” elementos en un tiempo proporcional a “n Log n”.

- Explica cómo funciona la lógica de partición y el proceso de ordenamiento.
La idea central de este algoritmo consiste en lo siguiente: Se toma un elemento “x” de una posición cualquiera del arreglo. Se trata de ubicar a “x” en la posición correcta del arreglo, de tal forma que todos los elementos que se encuentran a su izquierda sean menores o iguales a “x” y todos los elementos que se encuentren a su derecha sean mayores o iguales a “x”. Se repiten los pasos anteriores, pero ahora para los conjuntos de datos que se encuentran a la izquierda y a la derecha de la posición correcta de “x” en el arreglo. Reubicar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada. Repetir este proceso de forma recursiva para cada sablista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados. Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido. Ejemplo:



Proporciona un ejemplo práctico de implementación en PHP.

```php
<?php
// Función que implementa el algoritmo QuickSort
function quicksort($arr) {
    // Si el array tiene 0 o 1 elementos, ya está ordenado
    if (count($arr) < 2) {
        return $arr;
    }

    // Elegimos el pivote (según indica la investigación), en este caso el primer elemento
    $pivote = $arr[0];

    // Separa los elementos menores o iguales, mayores al pivote
    $menor = [];
    $mayor = [];

    // Recorre el array a partir del segundo elemento
    for ($i = 1; $i < count($arr); $i++) {
        if ($arr[$i] <= $pivote) {
            $menor[] = $arr[$i]; // Menores o iguales al pivote
        } else {
            $mayor[] = $arr[$i]; // Mayores que el pivote
        }
    }

    // Recursión para ordenar las dos mitades
    return array_merge(quicksort($menor), [$pivote], quicksort($mayor));
}

// Ejemplo de uso
$array = [7, 2, 9, 4, 5, 3, 8, 1];
echo "Array original: ";
print_r($array);

// Ordenamos el array usando QuickSort
$sortedArray = quicksort($array);
echo "Array ordenado: ";
print_r($sortedArray);
?>
```

2. Algoritmo de búsqueda Binary Search:
Investiga y explora el algoritmo de búsqueda binaria.

- Describe cómo funciona la búsqueda binaria, incluida la lógica detrás de la búsqueda en un conjunto de datos ordenado.
La búsqueda binaria es un algoritmo eficiente para encontrar un elemento en una lista ordenada de elementos. Funciona al dividir repetidamente a la mitad la porción de la lista que podría contener al elemento, hasta reducir las ubicaciones posibles a solo una. Usamos la búsqueda binaria en el juego de adivinar en la lección introductoria.
Una de las maneras más comunes de usar la búsqueda binaria es para encontrar un elemento en un arreglo. Por ejemplo, el catálogo estelar Tycho-2 contiene información acerca de las 2,539,913 estrellas más brillantes en nuestra galaxia. Supón que quieres buscar en el catálogo una estrella en particular, con base en el nombre de la estrella. Si el programa examinara cada estrella en el catálogo estelar en orden empezando con la primera, un algoritmo llamado búsqueda lineal, la computadora podría, en el peor de los casos, tener que examinar todas las 2,539,913 de estrellas para encontrar la estrella que estás buscando. Si el catálogo estuviera ordenado alfabéticamente por nombres de estrellas, la búsqueda binaria no tendría que examinar más de 22 estrellas, incluso en el peor de los casos.
Los siguientes artículos discuten cómo describir cuidadosamente el algoritmo, cómo implementar el algoritmo en JavaScript y cómo analizar su eficiencia.

- Proporciona un ejemplo práctico de implementación en PHP.

```
<?php
/**
 * Función que implementa la búsqueda binaria.
 * Parámetros
 * $arr: El arreglo ordenado en el que se busca el valor.
 * $searchValue: El valor que se desea buscar.
 * Devuelve el índice del valor si se encuentra, o-1 si no se encuentra.
 */
function binarySearch($arr, $searchValue) {
    // Definir el límite inferior y superior del arreglo
    $low = 0;
    $high = count($arr)- 1;

    // Mientras el límite inferior sea menor o igual al límite superior
    while ($low <= $high) {
        // Calcular el índice del medio
        $middle = floor(($low + $high) / 2);

        // Comprobar si hemos encontrado el valor
```

```
        if ($arr[$middle] === $searchValue) {
            return $middle; // Retorna el índice donde se encuentra el valor
        }

        // Si el valor objetivo es mayor que el valor en el medio
        if ($arr[$middle] < $searchValue) {
            $low = $middle + 1; // El valor objetivo está en la mitad derecha
        } else {
            $high = $middle- 1; // El valor objetivo está en la mitad izquierda
        }
    }

    // Si llegamos aquí, significa que no se encontró el valor
    return-1;
}

// Ejemplo de uso:
$arraycito = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19];
$number = 7;

$result = binarySearch($arraycito, $number);

if ($result !==-1) {
    echo "El valor buscado $number se encuentra en el índice $result.";
} else {
    echo "El valor buscado $number no se encuentra en el arreglo.";
}

?>
```

- Analiza la eficiencia y ventajas de la búsqueda binaria en comparación con otros métodos de búsqueda, especialmente en grandes conjuntos de datos.

Complejidad Temporal

Búsqueda binaria: Tiene una complejidad temporal de O(log n), donde n es el número de elementos en la lista. Esto se debe a que en cada paso el algoritmo reduce el espacio de búsqueda a la mitad, eliminando la mitad de los elementos de la consideración.

Comparación con búsqueda lineal: La búsqueda lineal tiene una complejidad de O(n), ya que en el peor caso, el algoritmo debe revisar cada elemento de la lista hasta encontrar el objetivo o determinar que no está presente.

Por lo tanto, en grandes conjuntos de datos, la búsqueda binaria es mucho más eficiente que la búsqueda lineal. Por ejemplo, si tienes una lista de 1 millón de elementos, la búsqueda binaria

necesitaría a lo sumo 20 comparaciones ($\log_2(1{,}000{,}000) \approx 20$), mientras que la búsqueda lineal podría requerir hasta 1 millón de comparaciones.