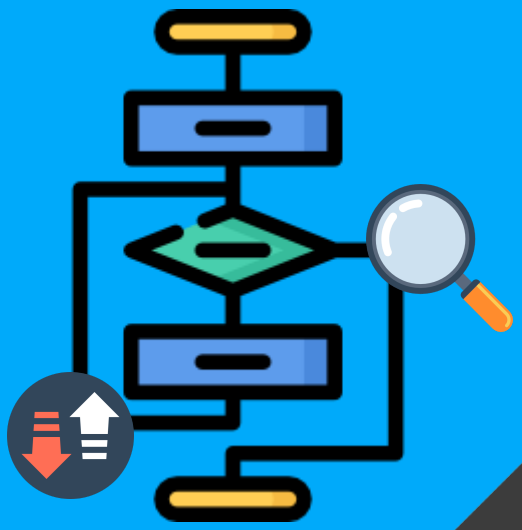
Java

//Algoritmos de búsqueda y ordenamiento



UTEC
Universidad Tecnológica

LICENICIATURA EN TECNOLOGÍAS
DE LA INFORMACIÓN

Algoritmos de Ordenamiento

Introducción

Un algoritmo de ordenamiento es un conjunto de instrucciones que especifican cómo organizar los elementos de una lista en un orden específico. El objetivo principal de los algoritmos de ordenamiento es colocar los elementos en la lista en una secuencia que sea más útil para futuras operaciones o para presentar los datos de manera más comprensible.

Utilidades de los algoritmos de ordenamiento en la vida cotidiana

1. **Organización de Datos Personales:** Los algoritmos de ordenamiento se pueden utilizar para organizar datos personales, como contactos en un teléfono móvil, lista de correos electrónicos o archivos en una computadora. Por ejemplo, ordenar los contactos alfabéticamente facilita la búsqueda rápida de un contacto específico.

2. **Listas de Compras:** Al ordenar los elementos de una lista de compras, se puede mejorar la eficiencia al recorrer el supermercado o tienda, ayudando a encontrar los productos más fácilmente y evitando la pérdida de tiempo.

3. **Clasificación de Documentos:** En entornos laborales, los algoritmos de ordenamiento se utilizan para clasificar documentos por fecha, prioridad o tipo. Esto ayuda a mantener una estructura organizada y facilita el acceso rápido a la información relevante.

4. **Ordenamiento de Libros y Películas:** En bibliotecas personales o en plataformas de streaming de contenido multimedia, los algoritmos de ordenamiento pueden ayudar a clasificar libros o películas por género, autor, año de publicación, calificación, etc.

5. **Presentación de Información:** En aplicaciones y sitios web, los algoritmos de ordenamiento se emplean para presentar información de manera más organizada y fácil de entender. Por ejemplo, en redes sociales, los mensajes se pueden ordenar por relevancia, fecha o interacción.

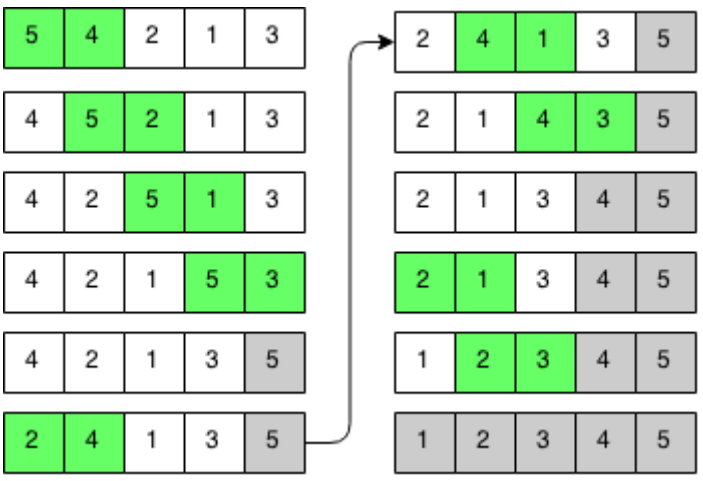
Detalles adicionales sobre los algoritmos de ordenamiento

- **Eficiencia:** La eficiencia de un algoritmo de ordenamiento se evalúa principalmente por su complejidad temporal (cuánto tiempo tarda en ejecutarse en función del tamaño de la entrada) y su complejidad espacial (cuánta memoria adicional requiere).
- **Estabilidad:** Algunos algoritmos de ordenamiento, como el algoritmo de burbuja y el de inserción, conservan el orden relativo de elementos iguales. Esto se conoce como estabilidad del algoritmo.
- **In-place vs. Externos:** Algunos algoritmos de ordenamiento modifican la lista de entrada (in-place), mientras que otros requieren memoria adicional (externos) para almacenar datos temporales durante el proceso de ordenamiento.

Burbuja (Bubble Sort)

Este algoritmo compara elementos adyacentes y los intercambia si están en el orden incorrecto. Tiene una complejidad de tiempo de O(n^2).

Como podemos observar en la siguiente imagen, las filas de valores representan el array en cada iteración de su recorrido para establecer el orden. En cada iteración podemos presenciar con el color verde cuáles son los valores que se están comparando en ese momento, en la primera iteración compara el primer valor (5) con el segundo (4), al darse cuenta que están en una posición incorrecta lo que hace es intercambiarlos. Y así seguirá sucesivamente con el resto de elementos.



Ejemplo de código de un algoritmo de ordenamiento de burbuja

```java
public class BubbleSort {
  public static void bubbleSort(int[] arr) {
    int n = arr.length;
    for (int i = 0; i < n - 1; i++) {
      for (int j = 0; j < n - i - 1; j++) {
        if (arr[j] > arr[j + 1]) {
          int temp = arr[j];
          arr[j] = arr[j + 1];
          arr[j + 1] = temp;
        }
      }
    }
  }
}
```

Para probar el algoritmo de ordenamiento de burbuja, podemos crear un caso de prueba con un conjunto de números desordenados y luego verificar si el algoritmo ordena correctamente la lista.
Te brindamos un ejemplo de un caso de prueba para el algoritmo de ordenamiento de burbuja: https://git.utec.edu.uy/fip/fip-m3-example-bubble_sort

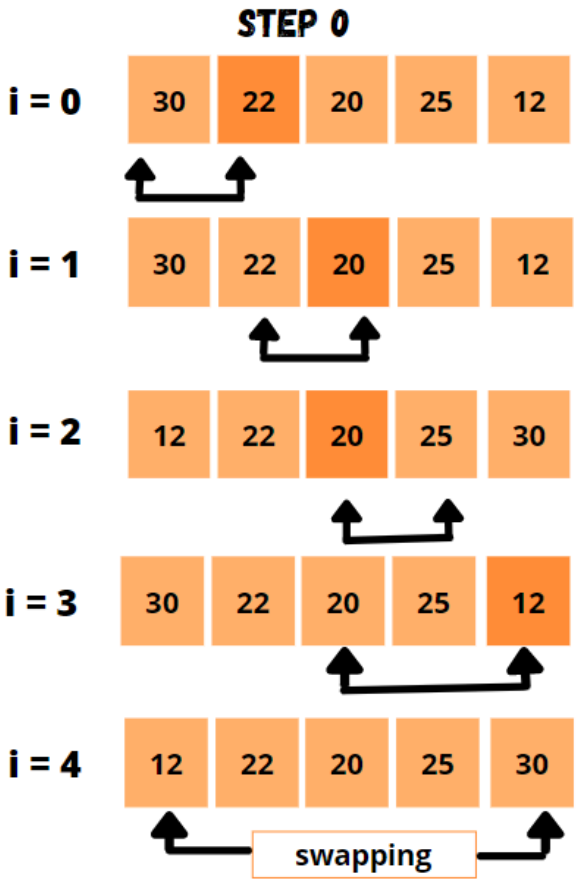
Selección (Selection Sort)

Este algoritmo busca el elemento más pequeño y lo coloca en la posición correcta. Tiene una complejidad de tiempo de O(n^2).

En el siguiente ejemplo de la imagen podemos observar como es la lógica por detrás de este algoritmo, lo primero que hace es recorrer todo el array buscando el valor más pequeño. Este array del ejemplo consta de 5 elementos, por lo tanto, con cuatro comparaciones ya logra identificar cuál es el valor más pequeño, para eso realiza las iteraciones 0, 1, 2 y 3, con sus respectivas comparaciones.

Cuando alcanza la iteración 4 realiza el llamado **swapping** que se trata de un intercambio de valores, el valor más pequeño que fue encontrado pasa a ocupar la primera posición del array.

Y luego, el procedimiento continúa de igual manera, pero sin tomar en cuando aquella posición que ya fue ordenada.



Ejemplo de código de un algoritmo de ordenamiento de selección

Java

```java
public class SelectionSort {
  public static void selectionSort(int[] arr) {
    int n = arr.length;
    for (int i = 0; i < n - 1; i++) {
      int minIndex = i;
      for (int j = i + 1; j < n; j++) {
        if (arr[j] < arr[minIndex]) {
          minIndex = j;
        }
      }
      int temp = arr[minIndex];
      arr[minIndex] = arr[i];
      arr[i] = temp;
    }
  }
}
```

Compartimos un caso de prueba para el algoritmo de ordenamiento por selección: https://git.utec.edu.uy/fip/fip-m3-example-selection_sort

Algoritmos de Búsqueda

Un algoritmo de búsqueda es un conjunto de instrucciones diseñadas para encontrar un elemento específico dentro de una colección de datos. El objetivo principal de los algoritmos de búsqueda es determinar si un elemento dado está presente en la colección y, en caso afirmativo, determinar su ubicación.

Tipos de Algoritmos de Búsqueda

- **Búsqueda Secuencial:** Este algoritmo examina cada elemento de la lista en orden secuencial hasta encontrar el elemento buscado o llegar al final de la lista.

- **Búsqueda Binaria:** Solo puede aplicarse a listas ordenadas. Divide repetidamente la lista en dos mitades y verifica si el elemento buscado está en la mitad izquierda o derecha, reduciendo así el espacio de búsqueda a la mitad en cada iteración.

- **Búsqueda por Interpolación:** Similar a la búsqueda binaria, pero utiliza una estimación más precisa de la posición del elemento basada en el valor del elemento y la distribución de los datos.

Utilidades de los Algoritmos de Búsqueda en la Vida Cotidiana

1. **Búsqueda de Elementos en Listas o Bases de Datos:** Los algoritmos de búsqueda se utilizan comúnmente en aplicaciones informáticas para encontrar elementos específicos en grandes conjuntos de datos, como buscar un contacto en la lista de contactos de un teléfono móvil o buscar una palabra clave en un motor de búsqueda en línea.

2. **Sistemas de Gestión de Inventarios:** En empresas y almacenes, los algoritmos de búsqueda se utilizan para localizar productos dentro del inventario, lo que ayuda a agilizar el proceso de envío y recepción de productos.

3. **Búsqueda de Archivos en Sistemas Operativos:** Los algoritmos de búsqueda se utilizan en sistemas operativos para encontrar archivos o carpetas en el sistema de archivos, lo que facilita la navegación y gestión de archivos para los usuarios.

4. **Búsqueda de Información en Bibliotecas Digitales:** En bibliotecas digitales y bases de datos académicas, los algoritmos de búsqueda permiten a los usuarios encontrar rápidamente artículos, libros o documentos relevantes según sus consultas de búsqueda.

Detalles Adicionales sobre los Algoritmos de Búsqueda

- **Eficiencia:** La eficiencia de un algoritmo de búsqueda se evalúa principalmente por su tiempo de ejecución, es decir, cuánto tiempo tarda en encontrar el elemento deseado en función del tamaño de la colección de datos.

- **Complejidad:** Los algoritmos de búsqueda pueden tener diferentes complejidades en función de la estructura de datos y el método utilizado para la búsqueda.

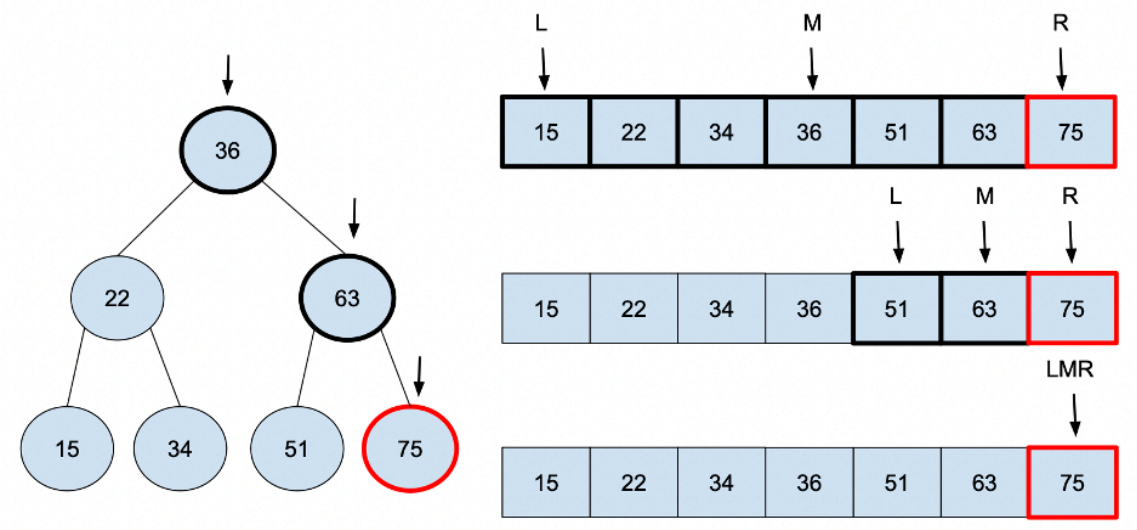
- **Relevancia del Contexto:** La elección del algoritmo de búsqueda adecuado depende del contexto específico, como si los datos están ordenados o no, la cantidad de datos a buscar, etc.

Búsqueda Binaria (Binary Search)

Este algoritmo busca un elemento en una lista ordenada dividiendo repetidamente el espacio de búsqueda a la mitad. Tiene una complejidad de tiempo de O(log n).

En el siguiente ejemplo, podemos observar como tenemos un array ordenado de menor a mayor, y en él se quiere encontrar la posición del valor 75. Por lo tanto, lo primero que hace es

dividir el array en Left (Izquierda), Right (derecha) y Medium (Medio). Analiza si el medio es mayor o menor al valor buscado y dependiendo del resultado tomará la decisión de si sigue buscando pero con los elementos a su izquierda, o con los elementos a su derecha o también podría haber encontrado su valor en el medio.



Ejemplo de este algoritmo:

```java
public class BinarySearch {
  public static int binarySearch(int[] arr, int target) {
    int left = 0;
    int right = arr.length - 1;
    while (left <= right) {
      int mid = left + (right - left) / 2;
      if (arr[mid] == target) {
        return mid;
      } else if (arr[mid] < target) {
        left = mid + 1;
      } else {
        right = mid - 1;
      }
    }
    return -1; // Si el elemento no se encuentra en la lista
  }
}
```

Compartimos un caso de prueba para el algoritmo de búsqueda binaria: https://git.utec.edu.uy/fip/fip-m3-example-binary_search