Sebastián Casañas Salcedo

ING De sistemas

Profesor Gamarra

Taller #1

Actividad estructurada Parte 1:

Aplicación de búsqueda binaria en lista no ordenada Lista predefinida (desordenada por código):

¿Qué resultados inesperados observaste? ¿Por qué ocurre esto si el libro sí está en la lista?

R// Al ejecutar el codigo sin ningun tipo de cambio observamos que se encuentran desordenados los libros.

Aplicando el algoritmo de ordenamiento:

```
let librosOrdenados = librosDesordenados.sort((a, b) => a.codigo.localeCompare(b.codigo));
console.log(librosDesordenados)
```

Los libros como su algoritmo lo dice, se ordenan por su codigo de libro de menor a mayor.

```
{ codigo: 'L001', titulo: 'Algoritmos I', anio: 2019 },
{ codigo: 'L003', titulo: 'Introducción a la IA', anio: 2023 },
{ codigo: 'L004', titulo: 'Bases de Datos', anio: 2018 },
{ codigo: 'L005', titulo: 'Redes de Computadores', anio: 2022 },
{ codigo: 'L008', titulo: 'Ingeniería de Software', anio: 2021 },
{ codigo: 'L010', titulo: 'Análisis Numérico', anio: 2020 }
```

Pregunta de análisis posterior (después de ordenar y comparar):

¿Qué diferencias observaste entre los dos resultados? ¿Qué nos dice esto sobre las condiciones necesarias para aplicar correctamente la búsqueda binaria?

R// La diferencia entre los dos resultados es un ordenamiento notorio de los libros en base a su código. Es nesesario tener un orden especifico para obtener un periodo de tiempo mas corto a la hora de recibir el resultado de este

Las condiciones necesarias para aplicar busqueda binaria son:

- El arreglo o lista debe estar **ordenado** de forma ascendente o descendente.
- Sin orden, no se puede determinar en qué mitad buscar, lo que rompe la lógica del algoritmo.

Con esto llegamos a la conclusion de que si no tenemos un orden especifico en nuestra lista el algoritmo puede que no encuentre el elemento aun si se encuentra presente al igual que nos puede reflejar una posición incorrecta, lo cual conlleva a errores en el programa

Parte 2: Ordenamiento de lista casi ordenada

Pregunta de análisis posterior:
Cuando una lista está casi ordenada, ¿qué algoritmo de ordenamiento es más eficiente: burbuja o inserción? ¿Por qué? Justifica con base en el comportamiento observado.

R// El algoritmo de ordenamiento mas eficiente cuando una lista esta casi ordenada es el de inserción. Ya que si nos basamos en esta lista obtuvo 6 comparaciones y 1 intercambio. A diferencia del algoritmo burbuja que que obtuvo 15 comparaciones y 1 intercambio.

Con esto llegamos a la conclusión de que entre menos comparaciones mas rápido actual el programa y mas rápido se obtiene el resultado esperado. Por ende es mejor para esta lista el algoritmo de inserción .

```
Restarting 'parte2.js'
Resultado Burbuja:
[
  { codigo: 'L001', titulo: 'Algoritmos I', anio: 2019 },
  { codigo: 'L002', titulo: 'Estructuras de Datos', anio: 2020 },
  { codigo: 'L003', titulo: 'Introducción a la IA', anio: 2023 },
  { codigo: 'L004', titulo: 'Bases de Datos', anio: 2018 },
  { codigo: 'L005', titulo: 'Redes de Computadores', anio: 2022 },
  { codigo: 'L006', titulo: 'Sistemas Operativos', anio: 2017 }
]
Comparaciones: 15
Intercambios: 1
Resultado Inserción:
[
  { codigo: 'L001', titulo: 'Algoritmos I', anio: 2019 },
  { codigo: 'L002', titulo: 'Estructuras de Datos', anio: 2020 },
  { codigo: 'L003', titulo: 'Introducción a la IA', anio: 2023 },
  { codigo: 'L004', titulo: 'Bases de Datos', anio: 2018 },
  { codigo: 'L005', titulo: 'Redes de Computadores', anio: 2022 },
  { codigo: 'L006', titulo: 'Sistemas Operativos', anio: 2017 }
]
Comparaciones: 6
Intercambios: 1
Completed running 'parte2.js'. Waiting for file changes before restarting...
```