Trabajo Práctico Integrador

Algoritmos de Búsqueda y Ordenamiento

Sebastián Nicolás Gossos Scribe
Marcelo Gomez Armoa

UTN
UNIVERSIDAD TECNOLÓGICA NACIONAL

¿Por qué los algoritmos de ordenamiento son importantes?

Los algoritmos de búsqueda son fundamentales porque:

- **Optimización de consultas**: Los datos ordenados permiten realizar operaciones como rangos, medianas y percentiles más rápidamente
- **Mejora de rendimiento**: Facilitan el acceso secuencial y reducen la fragmentación en memoria
- **Presentación de datos**: Los usuarios esperan ver información organizada (nombres alfabéticamente, fechas cronológicamente, etc.)

¿Cuál es la diferencia práctica entre O(n²) y O(n log n)?

O(n²) describe algoritmos donde cada elemento se compara con todos los demás, generando un crecimiento muy rápido del tiempo de ejecución a medida que aumentan los datos, como en los bucles anidados.

En cambio, **O(n log n)** aparece en algoritmos que dividen el problema y trabajan con cada parte de forma más eficiente, lo que permite procesar grandes cantidades de datos en mucho menos tiempo. Por eso, `O(n log n)` es preferido en tareas como la ordenación, ya que escala mejor cuando los datos crecen.

Ejemplo real con 1 millón de elementos

- **Ordenamiento por selección O(n²)**: ~400,000,000 operaciones (12 días de procesamiento)
- **QuickSort O(n log n)**: ~265,000 operaciones (20 segundos de procesamiento)

Comparación entre algoritmos de búsqueda

Algoritmo	Mejor Caso	Peor Caso	Caso Promedio	Complejidad Espacial	Eficiencia	Uso Recomendado
Búsqueda Lineal	O(1) (primer elemento)	O(n)	O(n)	O(1)	Baja	Listas/arrays no ordenados o muy pequeños; uso puntual.
Búsqueda Binaria	O(1) (elemento medio)	O(log n)	O(log n)	O(1)	Alta	Arrays/colecciones ordenadas; grandes volúmenes y pocas modificaciones.
Búsqueda de Interpolación	O(1) (distribución uniforme, cercano inicio)	O(n) (distribución muy desigual)	O(log log n) en casos ideales de distribución uniforme	O(1)	Alta si distribuciones uniformes	Datos ordenados y distribuidos uniformemente (por ejemplo, valores numéricos con rango conocido).
Búsqueda de Hash (Lookup)	O(1) promedio	O(n) (colisiones extremas)	O(1) promedio	O(n)	Muy alta (promedio)	Acceso rápido por clave cuando podemos usar tablas hash; caches, diccionarios.

¿Por qué los algoritmos de búsqueda son importantes?

Los algoritmos de búsqueda son fundamentales porque:

- **Eficiencia**: Permiten encontrar información rápidamente en grandes volúmenes de datos
- **Escalabilidad**: La diferencia entre O(n) y O(log n) es crítica con millones de registros
- **Recursos**: Menos tiempo de CPU y memoria significa menor costo operativo
- **Experiencia de usuario**: Búsquedas rápidas mejoran la usabilidad de aplicaciones

¿Por qué la búsqueda lineal no requiere una lista ordenada?

Búsqueda Lineal - **No requiere orden**

La búsqueda lineal funciona con cualquier lista porque busca elemento por elemento dentro de la lista hasta encontrarlo tiene una notación de O(n) por que en el peor de los casos tiene que revisar el elemento uno por uno.

```
def busqueda_lineal(lista, elemento):
    for i in range(len(lista)):
        if lista[i] == elemento:
            return i
    return -1
```

¿Por qué la búsqueda binaria requiere una lista ordenada?

Búsqueda Binaria - **Si requiere orden**

La búsqueda binaria funciona con listas ordenadas porque al dividir la lista en dos y buscar el elemento en la mitad, solo se puede saber en qué parte continuar si los datos están ordenados.

```
def busqueda_binaria(lista_ordenada, elemento):
  izquierda = 0
  derecha = len(lista_ordenada) - 1
  while izquierda ≤ derecha:
      medio = (izquierda + derecha) // 2
      if lista_ordenada[medio] == elemento:
          return medio
      elif lista_ordenada[medio] < elemento:
          izquierda = medio + 1
      else:
          derecha = medio - 1
  return -1
```

Comparación entre algoritmos de ordenamiento

Algoritmo	Mejor Caso	Peor Caso	Caso Promedio	Complejidad Espacial	Eficiencia	Uso Recomendado
Burbuja	O(n) (optimizado con detección de intercambio)	$O(n^2)$	$O(n^2)$	O(1)	Baja	Listas muy pequeñas o casi ordenadas, sólo para enseñanza o prototipos rápidos.
Selección	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	Baja	Listas muy pequeñas; rara vez en producción salvo fines didácticos.
Inserción	O(n) (casi ordenada)	$O(n^2)$	$O(n^2)$	O(1)	Media	Listas pequeñas o casi ordenadas; inserción incremental.
Ordenamiento Rápido (Quicksort)	O(n log n) (buen pivote)	$O(n^2)$ (pivote mal)	O(n log n)	O(log n) (recursión)	Alta	Arrays grandes en práctica, con pivote aleatorio o "median-of-three" para evitar peores casos.
Ordenamiento por Mezcla (Mergesort)	O(n log n)	O(n log n)	O(n log n)	O(n)	Alta	Datos grandes cuando hay memoria extra; estable; útil en escenarios donde queremos estabilidad o dividir y conquistar.