Caso de estudio

Estudio de Caso 1: Ordenamiento de Registros y Logs de Seguridad en un Centro de Monitoreo

Contexto: Una institución educativa cuenta con un Centro de Operaciones de Seguridad (SOC) que monitorea accesos a sus servidores, intentos fallidos de autenticación y actividades sospechosas.

Diariamente se generan miles de registros (logs) con información como:

- Fecha y hora
- Dirección IP
- Tipo de evento
- Nivel de severidad

Para priorizar incidentes críticos, el equipo necesita ordenar los registros por nivel de severidad y por fecha.

Problema: Los registros llegan desordenados, por lo que el analista debe:

1. Ordenarlos por nivel de severidad (1–5).
2. En caso de empate, ordenarlos por fecha más reciente.
3. Procesar los eventos más críticos primero.

Ejemplo de Datos:

```
logs = [
   {"fecha": "2026-02-20 08:15", "ip": "192.168.1.10", "severidad": 3},
   {"fecha": "2026-02-20 08:10", "ip": "192.168.1.25", "severidad": 5},
   {"fecha": "2026-02-20 08:18", "ip": "192.168.1.30", "severidad": 2},
   {"fecha": "2026-02-20 08:12", "ip": "192.168.1.40", "severidad": 5},
]
```

Aplicación del Ordenamiento

Opción 1: Algoritmo básico (Insertion Sort adaptado)

```
def insertion_sort_logs(logs):
   for i in range(1, len(logs)):
      clave = logs[i]
      j = i - 1

      while j >= 0 and (
```

```
            logs[j]["severidad"] < clave["severidad"] or
            (logs[j]["severidad"] == clave["severidad"] and logs[j]["fecha"] < clave["fecha"])
        ):
            logs[j + 1] = logs[j]
            j -= 1

        logs[j + 1] = clave

    return logs

ordenados = insertion_sort_logs(logs.copy())

for log in ordenados:
    print(log)
```

- Ordena primero por severidad descendente
- Luego por fecha más reciente

Resultado Esperado

Los registros con severidad 5 aparecerán primero, ordenados por fecha más reciente. Luego severidad 3, después severidad 2, etc.

Análisis Técnico

Importancia del Ordenamiento en Ciberseguridad

- Permite priorizar incidentes críticos.
- Facilita análisis forense.
- Mejora tiempos de respuesta ante amenazas.
- Reduce riesgo de ignorar eventos graves.

Costo Computacional

- Algoritmos básicos como Bubble, Selection e Insertion → $O(n^2)$
- No son adecuados para millones de registros.
- En entornos reales se utilizan algoritmos más eficientes como:
    - Merge Sort → O(n log n)

- Quick Sort → O(n log n)
- Timsort (usado en Python con sorted())

Reflexión Final

En sistemas de ciberseguridad, el ordenamiento no es solo una operación técnica, sino un mecanismo crítico de priorización.
Un algoritmo ineficiente puede retrasar la detección de un ataque activo.

La elección del algoritmo debe considerar:

- Volumen de datos
- Tiempo de respuesta requerido
- Recursos del sistema

Caso de estudio