Caso de estudio

Estudio de Caso 2: Medición Empírica de Eficiencia de Algoritmos de Ordenamiento en Logs de Seguridad

Contexto: Un Centro de Operaciones de Seguridad (SOC) analiza miles de registros diarios generados por firewalls, servidores y sistemas de autenticación.

Cada registro contiene:

- Timestamp
- Dirección IP
- Tipo de evento
- Nivel de severidad

Para priorizar incidentes críticos, los registros deben ordenarse por severidad y fecha.

El desafío es determinar qué algoritmo de ordenamiento es más eficiente cuando el volumen de datos crece.

Problema

Comparar empíricamente el tiempo de ejecución de:

- **Bubble Sort**
- **Selection Sort**
- **Insertion Sort**

Aplicados a conjuntos de datos de diferentes tamaños (1 000, 5 000 y 10 000 registros simulados).

Simulación de Datos

```
import random

# Simulación de niveles de severidad
def generar_logs(n):
    return [random.randint(1, 5) for _ in range(n)]
```

Algoritmos Evaluados

Bubble Sort

```
def bubble_sort(lista):
    n = len(lista)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
    return lista
```

Selection Sort

```
def selection_sort(lista):
    n = len(lista)
    for i in range(n):
        minimo = i
        for j in range(i + 1, n):
            if lista[j] < lista[minimo]:
                minimo = j
        lista[i], lista[minimo] = lista[minimo], lista[i]
    return lista
```

Insertion Sort

```
def insertion_sort(lista):
    for i in range(1, len(lista)):
        clave = lista[i]
        j = i - 1
        while j >= 0 and lista[j] > clave:
            lista[j + 1] = lista[j]
            j -= 1
        lista[j + 1] = clave
    return lista
```

Medición Empírica

```
import time

tamaños = [1000, 5000, 10000]

for tamaño in tamaños:
    datos = generar_logs(tamaño)
```

```
    for algoritmo in [bubble_sort, selection_sort, insertion_sort]:
        copia = datos.copy()
        inicio = time.time()
        algoritmo(copia)
        fin = time.time()
        print(f"{algoritmo.__name__} con {tamaño} datos: {fin - inicio:.4f}
segundos")

    print("-" * 50)
```

Resultados Esperados (Ejemplo)

Tamaño	Bubble	Selection	Insertion
1 000	0.08 s	0.06 s	0.04 s
5 000	2.1 s	1.8 s	1.5 s
10 000	8.5 s	7.9 s	6.8 s

(Los valores pueden variar según el equipo.)

Análisis

- Los tres algoritmos presentan crecimiento cuadrático **O(n^2)**.
- La diferencia se vuelve significativa a medida que aumenta el volumen.
- Insertion Sort suele ser ligeramente más eficiente en listas parcialmente ordenadas.
- Ninguno es óptimo para grandes volúmenes reales de logs.

Aplicación en Ciberseguridad

En entornos reales:

- Los SOC pueden procesar millones de eventos por día.
- Un algoritmo ineficiente puede retrasar la priorización de incidentes críticos.
- La eficiencia impacta directamente en el tiempo de respuesta ante ataques.

Por ello, en sistemas reales se utilizan algoritmos más eficientes como:

- Merge Sort
- Quick Sort
- Timsort (implementado en sorted() de Python)

Conclusión

La medición empírica demuestra que, aunque los métodos básicos son útiles para aprendizaje y conjuntos pequeños, no son adecuados para sistemas de ciberseguridad a gran escala.

La selección del algoritmo debe basarse en:

- Volumen de datos
- Tiempo de respuesta requerido
- Recursos computacionales disponibles

En ciberseguridad, la eficiencia algorítmica es un componente estratégico en la defensa digital.

Caso de estudio