Guía de laboratorio

Laboratorio de Contenido 3: Heap Sort en sistemas de prioridad

Explicación básica

Heap Sort utiliza una estructura llamada heap (montículo binario). Construye un Max-Heap y extrae repetidamente el elemento mayor. Garantiza O(n log n) en todos los casos y usa O(1) espacio adicional.

Objetivos

- Comprender la estructura heap.
- Analizar su rendimiento garantizado.
- Aplicarlo a colas de prioridad.

Caso práctico

Un sistema gestiona incidentes críticos y debe procesar primero los de mayor prioridad.

Problema

Ordenar prioridades:

[4, 9, 1, 7, 3, 8]

Ejemplo paso a paso

Construcción del Max-Heap:

[9,7,8,4,3,1]

Extracciones sucesivas:

[1,3,4,7,8,9]

Implementación en Python

```
def heapify(arr, n, i):
    largest = i
    left = 2*i + 1
    right = 2*i + 2

    if left < n and arr[left] > arr[largest]:
        largest = left

    if right < n and arr[right] > arr[largest]:
        largest = right

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
```

```
        heapify(arr, n, largest)

    def heap_sort(arr):
        n = len(arr)

        for i in range(n//2 - 1, -1, -1):
            heapify(arr, n, i)

        for i in range(n-1, 0, -1):
            arr[i], arr[0] = arr[0], arr[i]
            heapify(arr, i, 0)

        return arr

    datos = [4, 9, 1, 7, 3, 8]
    print(heap_sort(datos))
```

Conclusiones

Heap Sort es ideal cuando se requiere rendimiento garantizado en el peor caso. Es ampliamente utilizado en sistemas de prioridad y gestión de eventos críticos en ciberseguridad.