

### Reflexión Actividad 3.3

#### **Importancia de usar estructuras jerárquicas en este reto**

Durante este reto me di cuenta de que trabajar con estructuras jerárquicas como el Binary Heap hace todo más práctico, especialmente cuando se necesita priorizar información, como en este caso donde estamos ordenando IPs o buscando elementos que se repiten. Estas estructuras permiten organizar los datos de forma que acceder al dato más “importante” (por ejemplo, el mayor o menor) sea rápido. En un proyecto real donde hay muchísimos registros, esto es clave para que el programa no se tarde ni se trabe.

#### **¿Por qué es mejor usar Binary Heap en lugar de BST?**

Elegimos Binary Heap porque está más optimizado para este tipo de tareas. Un BST (árbol binario de búsqueda) también permite organizar datos, pero puede desequilibrarse y eso hace que algunas operaciones se vuelvan lentas. En cambio, el Binary Heap mantiene su forma balanceada por naturaleza. Eso significa que operaciones como insertar o sacar el elemento de mayor prioridad siempre tardan lo mismo, sin importar cuántos datos tengamos. En mi experiencia, fue mucho más fácil trabajar con heap para organizar y acceder a los registros.

#### **Complejidad computacional y cómo impacta**

Estas son las complejidades de las funciones clave que se usaron :

push (insertar elemento):  $O(\log n)$   
pop (sacar el más importante):  $O(\log n)$   
getTop (ver el más importante):  $O(1)$

Esto quiere decir que incluso si tenemos una cantidad grande de datos, el tiempo que se tarda el programa en hacer estas operaciones no crece de manera brusca. Esa eficiencia hace que el programa sea más estable y rápido, lo cual es muy útil si queremos que funcione bien con muchos registros.

¿Cómo saber si una red está infectada?

Una idea que tuvimos mientras resolvía esto fue que si hay muchos intentos de conexión desde una misma IP en muy poco tiempo, eso podría ser una señal de que algo raro está pasando, como un ataque. También si una IP aparece muchas veces en la bitácora en horarios inusuales o tratando de acceder a muchas direcciones distintas. Si se usa un heap, se puede detectar fácilmente cuál IP se repite más o cuál tiene más prioridad en cuanto a actividad sospechosa.

### **Algoritmos de ordenamiento y búsqueda**

Aunque usamos Heapsort en esta actividad, también investigamos otros algoritmos. Por ejemplo:

Quicksort: rápido en la mayoría de los casos ( $O(n \log n)$ ) pero puede fallar en el peor caso ( $O(n^2)$ )

Mergesort: siempre es  $O(n \log n)$ , pero usa más memoria

Heapsort: lo bueno es que mantiene  $O(n \log n)$  y usa menos memoria que Mergesort, además de estar ligado directamente al Binary Heap

Para buscar elementos, si los datos ya están ordenados, la búsqueda binaria es muy eficiente ( $O(\log n)$ ). Pero si no lo están, hay que usar búsqueda secuencial, que tarda más ( $O(n)$ ).

### **Conclusión**

Siento que trabajar con un Binary Heap fue lo más acertado para este reto. Me ayudó a entender mejor cómo usar estructuras de datos no solo porque son “lo que toca” en la materia, sino porque realmente se aplican a problemas del mundo real. A veces uno piensa que esto es muy teórico, pero al analizar una bitácora o detectar patrones de red, todo esto cobra sentido.

GeeksforGeeks. (2014, November). *Binary Heap*. GeeksforGeeks.

<https://www.geeksforgeeks.org/dsa/binary-heap/>

Olivera, A. M. (2020, May 11). *Algoritmos de Búsqueda y Ordenamiento*. Medium.

<https://medium.com/@mise/algoritmos-de-b%C3%BAsqueda-y-ordenamiento-7116bcea03d0>

GeeksforGeeks. (2024, January 24). *Heap Data Structure*. GeeksforGeeks.

<https://www.geeksforgeeks.org/dsa/heap-data-structure/>

*Quicksort vs Mergesort vs Heapsort – Aula en la nube*. (s. f.).

<https://aulaenlanube.com/los-mejores-algoritmos-ordenacion-quicksort-mergesort-heapsort/%7D>

AlgoDaily. (s. f.). *AlgoDaily - Daily coding interview questions. Full programming interview prep course and software career coaching*.

<https://algodaily.com/lessons/merge-sort-vs-quick-sort-heap-sort>

---

Mariana Aguilar Requena  
A01569276

