



**Tecnológico  
de Monterrey**

Campus Estado de México

Reflexión integradora 3

Samuel Rincón V. A01752573

Programacion de estructuras de datos y algoritmos fundamentales

05/13/2025

En esta actividad, fue de suma importancia el uso de estructuras de datos jerárquicas, para procesar grandes volúmenes de datos es clave para hacerlo de manera eficiente. El uso de un Binary Heap permitió que se identificaran las IPs con mayor número de accesos sin necesidad de recorrer todo el arreglo cada vez. La naturaleza jerárquica de esta estructura facilita operaciones de prioridad (máximo o mínimo) en  $O(\log n)$ , esto es muy eficaz comparado con estructuras secuenciales como listas o vectores.

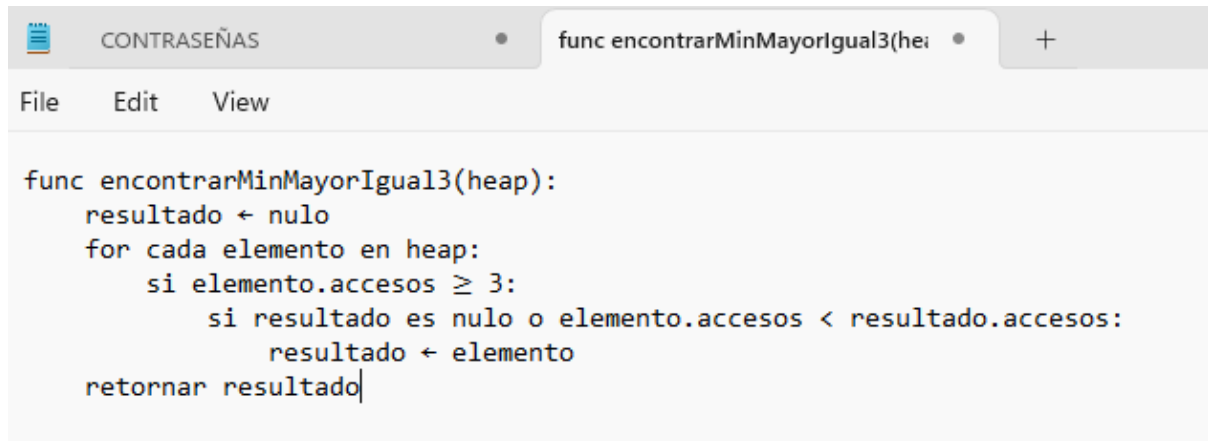
Cuando se quiera priorizar o acceder rápidamente al elemento con mayor o menor valor (colas de prioridad), es indispensable este tipo de estructura para mantener bajo el costo computacional.

Un BST también puede mantener datos ordenados pero presenta varias desventajas frente a un binary heap en esta entrega. Las operaciones de inserción y eliminación en un BST tienen una complejidad promedio de  $O(\log n)$ , pero se pueden degradar a  $O(n)$  si el árbol se desbalancea. El binary heap garantiza  $O(\log n)$  tanto en el push como en pop, manteniendo la eficacia.

Además, el Binary Heap está diseñado específicamente para optimizar la obtención del elemento de mayor prioridad, que en este reto se traduce a la IP con más accesos. Esta operación (getTop) tiene una complejidad constante  $O(1)$ .

- **push:  $O(\log n)$**   
Inserta un nuevo elemento en el heap y lo reordena hacia arriba si es necesario.
- **pop:  $O(\log n)$**   
Elimina el elemento raíz (de mayor prioridad) y reordena el heap hacia abajo.
- **top:  $O(1)$**   
Devuelve el elemento de mayor prioridad sin modificar la estructura.
- **buildHeap:  $O(n)$**   
Construye el heap desde un arreglo de elementos en tiempo lineal amortizado.

El uso del Binary Heap permitió cumplir los requerimientos de manera eficiente incluso con más de 7000 registros.



```
func encontrarMinMayorIgual3(heap):  
    resultado ← nulo  
    for cada elemento en heap:  
        si elemento.accesos ≥ 3:  
            si resultado es nulo o elemento.accesos < resultado.accesos:  
                resultado ← elemento  
    retornar resultado
```

Este fue el pseudocódigo para buscar IP con menor número de accesos  $\geq 3$


Este algoritmo tiene una complejidad de  $O(n)$ , debe recorrer todo el heap para encontrar el mínimo con la condición deseada. Un min-heap podría optimizar la búsqueda pero se eligió el max-heap por el enfoque en obtener las IPs con mayor número de accesos.


Hubo diferentes retos al momento de hacer esta entrega pero uno de los principales retos que se tuvo fue evitar errores de ejecución como el segmentation fault presente en las entregas anteriores. Se prestó especial atención al uso de punteros, manejo de archivos, compilación y estructuras dinámicas.


Otro de los retos fue asegurarse de que el ordenamiento por IP ignorara correctamente los puertos, se separó la IP de la cadena antes de realizar cualquier comparación, extrayendo únicamente la parte antes de los dos puntos ( : ). Esta lógica fue aplicada en el comparador del Heapsort, lo que permitió ordenar adecuadamente todos los registros.

Además, se decidió incluir toda la lógica relacionada con la estructura del Binary Heap directamente en el archivo main.cpp, en lugar de separar su implementación en un archivo de encabezado (.h). Esta decisión permitió evitar errores de vinculación (linker errores) al compilar en entornos como Reptil o compiladores con configuración básica. También simplificó el manejo de dependencias y facilitó las pruebas rápidas, sin perder modularidad gracias al uso de clases definidas dentro del mismo archivo.


## CHECKLIST


Se abre el archivo bitacoraHeap.txt y se almacenan los datos en un vector. 

Se implementa y ejecuta correctamente el Heap Sort por IP (sin puerto). 


Se guarda el resultado ordenado en bitacora\_ordenada.txt. 


Se cuenta el número de accesos por IP y se almacena en un Binary Heap. 

Se obtienen las 10 IPs con más accesos y se guardan en ips\_con\_mayor\_acceso.txt. 

Se encuentra la IP con menor número de accesos  $\geq 3$  y se muestra correctamente. 

Se analizó el uso de estructuras jerárquicas. 

Se justificó el uso de Binary Heap frente a BST. 

Se analizó la complejidad de operaciones (push, pop, top). 

Se escribió pseudocódigo y su complejidad. 

Se incluyeron fuentes. 

## REFERENCIAS

1. cplusplus.com. (s.f.). *ifstream* - C++ Reference. Recuperado de <http://www.cplusplus.com/reference/fstream/ifstream/>
2. GeeksforGeeks. (s.f.). *Extract IP address and port from a given string in C++*. Recuperado de <https://www.geeksforgeeks.org/extract-ip-address-and-port-from-a-given-string-in-cpp/>
3. cppreference.com. (s.f.). *std::unordered\_map* - cppreference.com. Recuperado de [https://en.cppreference.com/w/cpp/container/unordered\\_map](https://en.cppreference.com/w/cpp/container/unordered_map)
4. GeeksforGeeks. (s.f.). *Heap Sort*. Recuperado de <https://www.geeksforgeeks.org/heap-sort/>
5. cplusplus.com. (s.f.). *priority\_queue* - C++ Reference. Recuperado de [https://cplusplus.com/reference/queue/priority\\_queue/](https://cplusplus.com/reference/queue/priority_queue/)
6. DigitalOcean. (s.f.). *Priority Queue in C++ STL*. Recuperado de <https://www.digitalocean.com/community/tutorials/priority-queue-in-cpp-stl>
7. GeeksforGeeks. (s.f.). *Sorting a vector in C++*. Recuperado de <https://www.geeksforgeeks.org/sorting-a-vector-in-c/>
8. cplusplus.com. (s.f.). *vector* - C++ Reference. Recuperado de <https://cplusplus.com/reference/vector/vector/>
9. Stack Overflow. (2008). *How to split a string in C++*. Recuperado de <https://stackoverflow.com/questions/236129/how-to-split-a-string-in-c>
10. YouTube. (2020). *Heap Sort Algorithm | Data Structures and Algorithms*. [Video]. Recuperado de <https://www.youtube.com/watch?v=YXcgD0MGj0c>
11. cppreference.com. (s.f.). *std::make\_heap* - cppreference.com. Recuperado de [https://en.cppreference.com/w/cpp/algorithm/make\\_heap](https://en.cppreference.com/w/cpp/algorithm/make_heap)
12. Khan Academy. (s.f.). *Binary heaps*. Recuperado de <https://www.khanacademy.org/computing/computer-science/algorithms/heap/a/binary-heaps>
13. Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.
14. Stroustrup, B. (2013). *The C++ Programming Language* (4th ed.). Addison-Wesley.

15. Microsoft Learn. (s.f.). *ifstream class*. Recuperado de <https://learn.microsoft.com/en-us/cpp/standard-library/ifstream-class>
16. Programiz. (s.f.). *C++ Vectors*. Recuperado de <https://www.programiz.com/cpp-programming/vectors>
17. Josuttis, N. M. (2012). *The C++ Standard Library: A Tutorial and Reference* (2nd ed.). Addison-Wesley Professional.