- **What would you do differently if you had more time?**

An optimization that could be done is to do buckets of heaps to sort this data. We could push some IP ranges to different heaps. This would scale well to distributed computing to further speed up the process. This would be a better option if I could ask clarifying questions in an in-person interview.

A similar option would be to use a max heap for all data except the top 100. Then a min heap to store the top 100. This would make getting the top 100 operation O(constant). It would make insertion of a new item or updating an item slightly more complex since you'd have 2 heaps to check, but since it was a min-heap, you'd be able to check if the max of the max-heap after updating was greater than the min of the min-heap. Then we could move in that new value to the min-heap for the top 100. Time complexity would be the same O(n) for request_handled, constant for top100(), constant for clear()

- **What is the runtime complexity of each function?**

*request_handled(ip):*

Worst case we do heapify => O(n).

The heap restoring functions (heapify, heappush) can be removed as well from the request_handled function and called in top100() before using the heap since it is the only function that relies on the ordering of the heap. That means the max heap would be in a bad state. I preferred to leave it in the function to improve the code quality and retain the validity of the max heap for future updatability.

*top100():*

We grab out 100 items using heap pop => 100*(log(n)) => log(n)

We then call heapify => O(n)

Therefore, the upper bound of the time complexity is O(n)

*clear():*

We just clear out the references to the data => O(0)

- **How does your code work?**

We have a max heap that stores the most frequent IPs. The data in the heap is a custom frequency, ip pair class so it can be sorted by frequency.

We then maintain a hashmap/dictionary that stores the IP to that same data structure pairing. This way we can easily update the count of the correct item in the heap, without searching through the heap for it.

- **What other approaches did you decide not to pursue?**

Beyond the options I listed above that gave some improvements depending on exact use case, there were a lot of naive approaches that would work.

The obvious brute force to store all IPs in a dict then just scan through everything to find the top 100.

- **How would you test this?**

There would be 2 main tests categories that would need to be run:

1. Correctness
    - Unit tests would be written to confirm the functions were correct and not broken in the future.
2. Speed.
    - We had a requirement of 300ms to display the dashboard
        - This would need to be tested on the correct hardware with worse case 20mil IP addresses in play.
            - There is some example data showing up to 500k data points on my laptop well under 100ms response time on top100(). Please see example_data.txt for more detail.