Priority queues are a data structure which has three operations:

- put(k, v)

: puts the given value with the given key into the queue

- peek()

: returns the value with the lowest key

- pop()

: removes the value with the lowest key

In simple terms, they keep data in a sorted order in a way that allows insertions and specifically optimized for accessing the lowest item at any time. They are very useful in a lot of applications, including:

- On-chain market order books, where you want to match incoming orders with the best available order
- Nth price auctions
- Various token sale models
- Validator induction in proof of stake systems

The usual way to implement a priority queue is aheap, which has O(log(n)) overhead for a put

and a pop

. The problem with implementing heaps on ethereum is that ethereum's only available data structure is a trie, which already

has O(log(n)) overhead. Hence, the de-facto overhead of a heap in ethereum is the cumbersome O(log^2(n)).

There is a better way. We can store a dataset as a doubly linked list, ie. a series of objects of the form [prevkey, value, postkey]

(prevkey and postkey can easily be stored in one storage slot, so this is two storage slots max). We store a pointer to the first value. Peek and pop are easy: just look at that pointer.

Insertion now becomes trickier: a naive insertion would require walking through the entire list and doing O(n) reads and eventually doing O(1) writes to insert the new element (4 storage keys: 1 for the prev item's postkey, 2 for the new item, 2 for the next item's prevkey). In the current ethereum, reading is much cheaper than writing, so this is actually quite reasonable for lists up to a few hundred items in size. However, we can make it even cheaper by doing that computation off-chain, and requiring the submitter to attach a witness

to their transaction - the function in the contract for inserting would require an additional input which specifies the position where the new element would be inserted. This gives us O(1) reads and O(1) writes, so O(log(n)) overhead in total on top of the trie.