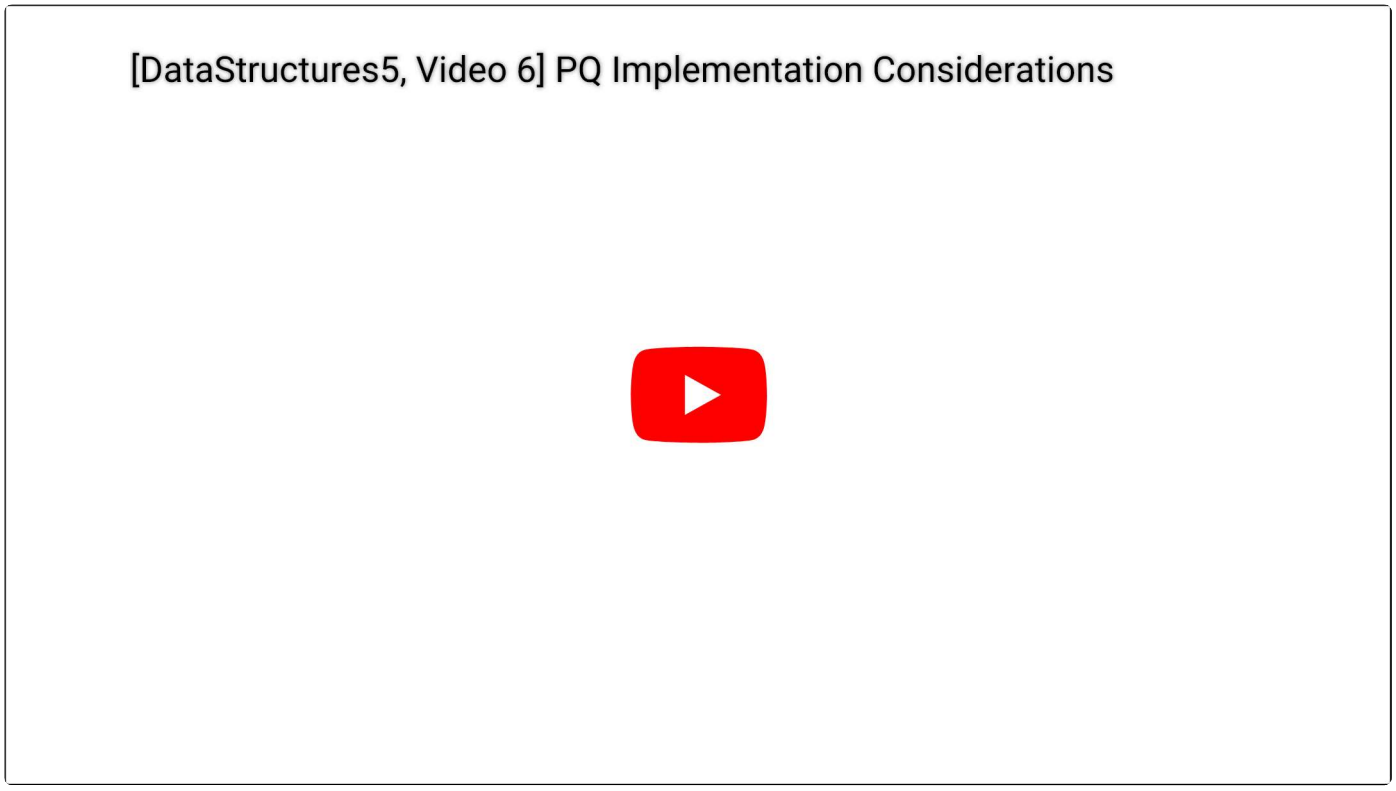
21.3 PQ Implementation



PQ Implementation Considerations

The actual implementation, which we will use and the book uses, is quite similar to the representation we discussed at the end of the previous chapter. The one difference is that we will leave one empty spot at the beginning of the array to simplify computation.

- `leftChild(k)` = $k * 2$
- `rightChild(k)` = $k * 2 + 1$
- `parent(k)` = $k/2$

Comparing to alternative implementations

Methods	Ordered Array	Bushy BST	Hash Table	Heap
`add`	$\Theta(N)$	$\Theta(logN)$	$\Theta(1)$	$\Theta(logN)$

Methods	Ordered Array	Bushy BST	Hash Table	Heap
`getSmallest`	$\Theta(1)$	$\Theta(logN)$	$\Theta(N)$	$\Theta(1)$

Awesome! We can see that we have improved our runtime and we have also solved the problem of duplicate elements. Couple notes:

- Heap operations are **amortized** analysis, since the array will have to resize (not a big deal)
- BST's can have constant time `getSmallest` if pointer is stored to smallest element
- Array-based heaps take around 1/3rd the memory it takes to represent a heap using approach 1A (direct pointers to children)

Exercise 13.3.1 Some lingering implementation questions:

1. How will the PQ know how to order the items? Say we had a PQ of dogs, would it order by weight or breed?
2. Is there a way to allow for a flexibility of orderings?
3. What could we do to make a MinPQ into a MaxPQ?

Previous
21.2 Heaps

Next
21.4 Summary

Last updated 1 year ago