ADS 2021: Week 5 Solutions

Solutions for week 5 of Algorithms and Data Structures.

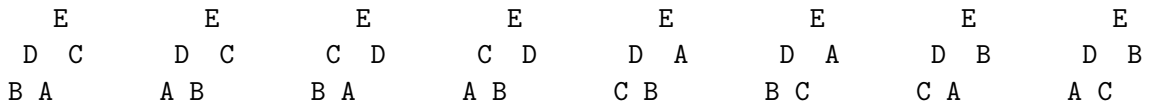
2.4.4 - Green Yes, an array that is sorted in decreasing order is a max-oriented heap.

2.4.7 - Green

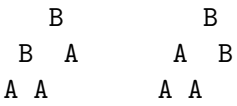
K	Can appear	Cannot appear
2	2,3	1,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20, 21,22,23,24,25,26,27,28,29,30,31
3	2,3,4,5,6,7	1,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23, 24,25,26,27,28,29,30,31
4	2,3,4,5,6,7,8,9,10,11,12,13,14,15	1,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31

2.4.9 - Green

ABCDE:



AAABB:



2.4.15 - Yellow For each element in the heap, check if both children (at position (2*index) and ((2*index) + 1)) are smaller than the current index. Make sure to not go out of bounds.

2.4.21 - Yellow Use the generic MaxPQ API on page 309 in the book and make a custom class to use as a key:

```
// Pseudocode
class PQKey<T> implements Comparable<T>
  T item;
  int priority;

  compareTo(other):
    return this.priority > other.priority
```

Then keep track of how many elements have been added so far. Let's call that variable *nextIndex* and increment it on every *push()* and *enqueue()*. Note that it is not decremented in *pop()* or *dequeue()*.

- Stack
  - Push: Add the item as a Key with priority *nextIndex*, meaning an item pushed later always has the highest priority.
  - Pop: RemoveMax

- Queue:
  - Enqueue: Add the item as a Key with priority -*nextIndex*, meaning an item enqueued earlier always has the highest priority.
  - Dequeue: RemoveMax
  - Alternatively, use a MinPQ and use priority *nextIndex* just as the stack.

2.4.27 - Yellow In a MaxPQ, the min can only change if a smaller item is inserted or the queue becomes empty. The minimal element will always be the last removed from the priority queue through calls to *delMax()*. The solution is therefore simple, keep a variable *private Key minItem*, update it on calls to *insert* if *minItem* is currently *null* or the new value is smaller, set it to *null* if *delMax* removes the last element of the queue, and return *minItem* from *min()*. Note that we may want to be careful about returning the minimal Key unless it is immutable, as callers may otherwise change its value and ruin the heap order of the priority queue. The Key returned from *delMax* has already been removed from the queue prior to being returned and does therefore not suffer such risks.

2.4.29 - Red The idea here is to use 2 priority queues, an indexMinPQ and an indexMaxPQ. To insert, we call insert on both PQs associating the item with the same index in both. This is done in logarithmic time.

To find the max or min, we just call max() on the maxPQ and min() on the minPQ, respectively. This is done in constant time.

To delete the maximum, we call delMax() on the maxPQ, which deletes the maximum item and returns its index. We then use this index to call delete(index) on the minPQ. Both of these operations are done in logarithmic time. Similarly, to delete the minimum we call delMin on the minPQ and use the index to delete from the maxPQ.