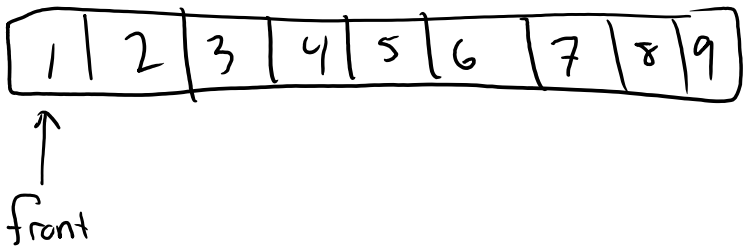
2019-02-26 Circular Queues

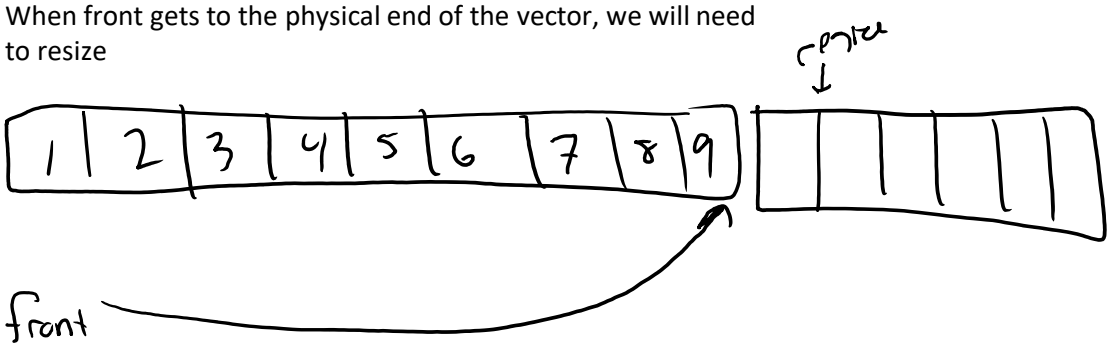
Tuesday, February 26, 2019 8:58 AM

Recall from last lecture

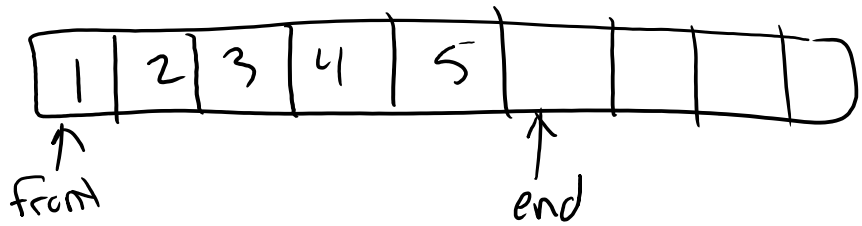
- Vector-based queues are quite a bit slower than LL queues because we have to shift all elements down by 1 on every dequeue
- A key aspect to data structure design is separating logical from physical constraints
- Logical constraint is conceptual restriction
  - For queues: items must come out in the order in which they're inserted
  - Implies that we need to track the front of the queue and the end of the queue and the order in which things are inserted
- Physical constraint are limitations on physical properties of a given data type
  - Vectors have a physical constraint that says the front of the vector is always at element 0.
  - LL's do not have such a physical constraint
- A circular queue asks: why must the "front" of the queue correspond to element 0 in the vector?
  - As such, the circular queue makes the "front" a logical pointer
- Example:



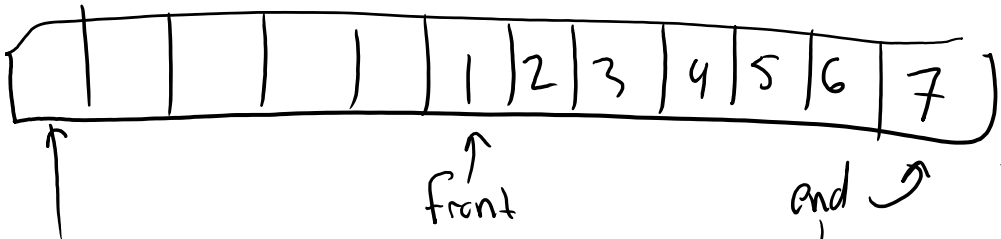
- With this setup, we no longer return data[0], but data[front]
- Thus, when performing a dequeue, we merely do a front++
- Issues with this scheme:
  - When front gets to the physical end of the vector, we will need to resize



  - Other issue: we may not ever be able to reuse the memory that exists before the front pointer
- Other observation of circular queue: the end can also be a logical pointer
- Thus, we dequeue from front pointer and enqueue from end pointer.



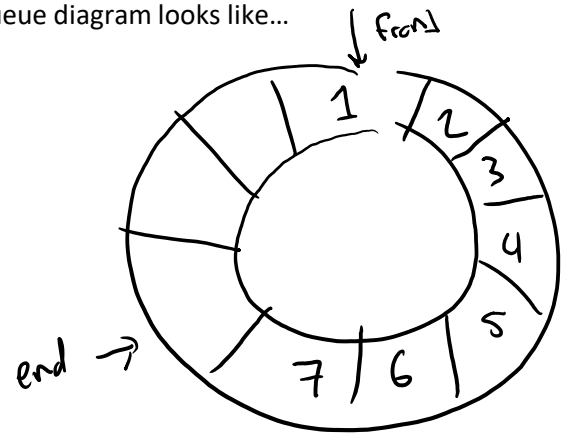
- What happens on an enqueue on the following circular queue?

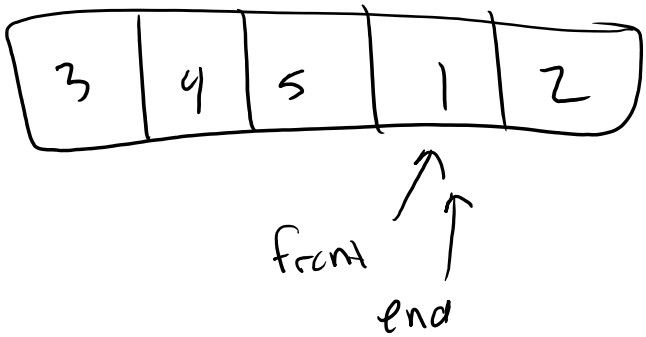


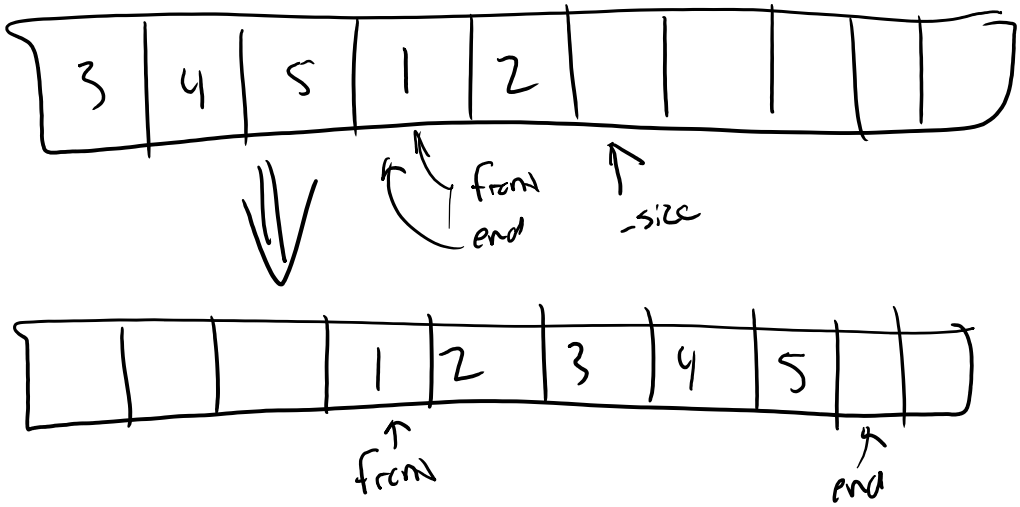
- "wrap" end pointer around to physical front

- In this scheme, we are able to reuse space from previous dequeues without having to resize
- Question: when will we still have to resize?
- Answer: when end and front point to the same thing
- WARNING: this occurs when END and FRONT point to the same thing
- A true circular queue diagram looks like…

redrawing diagram from above…






3
4
5
1
2
from
end
size
1
2
3
4
5
from
end