```
const int DX[] = {-1, 0, 0, 1};
const int DY[] = {0, -1, 1, 0};
```

How do I implement a queue?

QUEUE: ABSTRACT DATA TYPE

→ we can produce the required behavior but store the data in different ways.

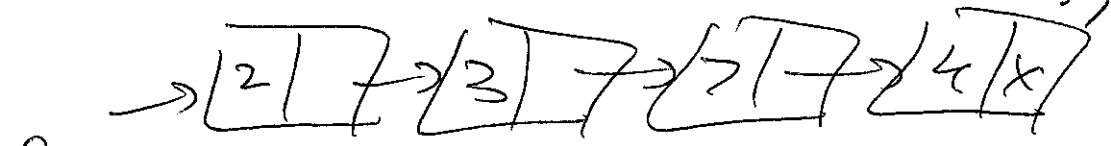
TWO IMPLEMENTATIONS

---

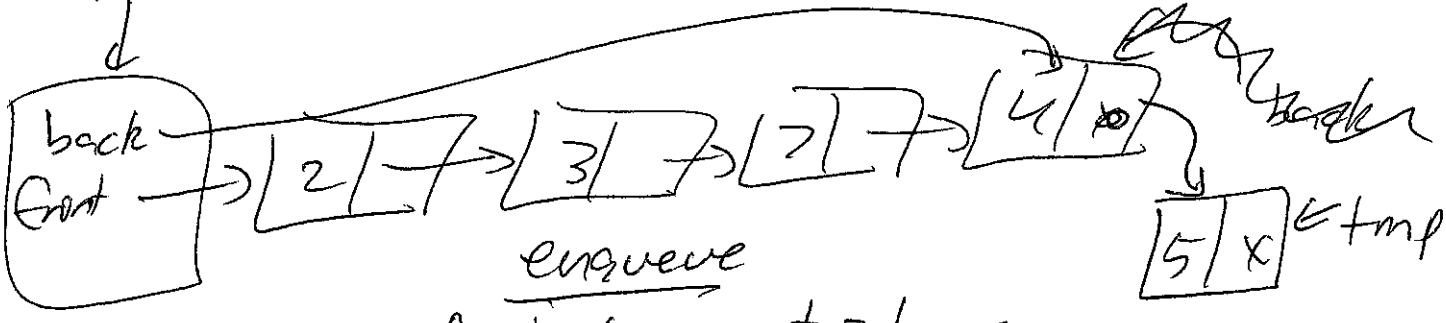
① Linked List

② Array

```
struct queue {
    struct node* front;
    struct node* back;
};
```

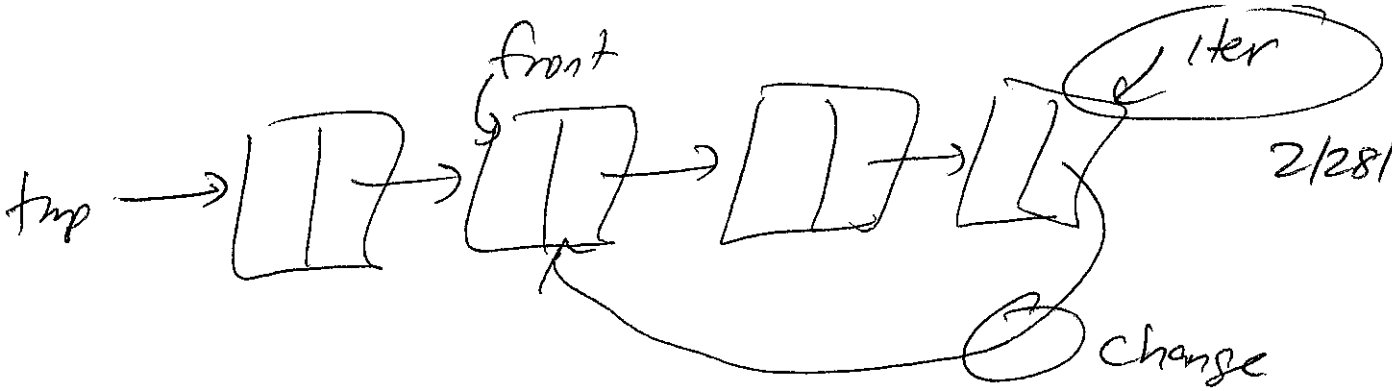


enqueue insert back for n element O(n), dequeue O(1)



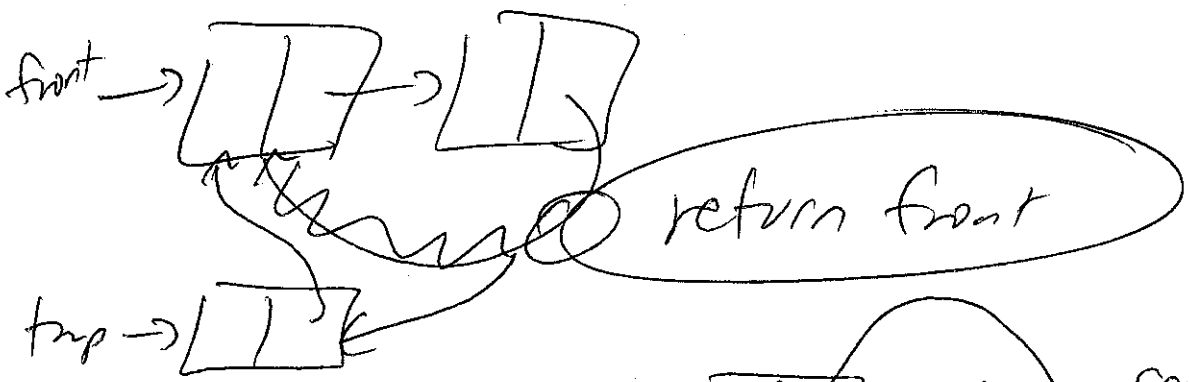
enqueue

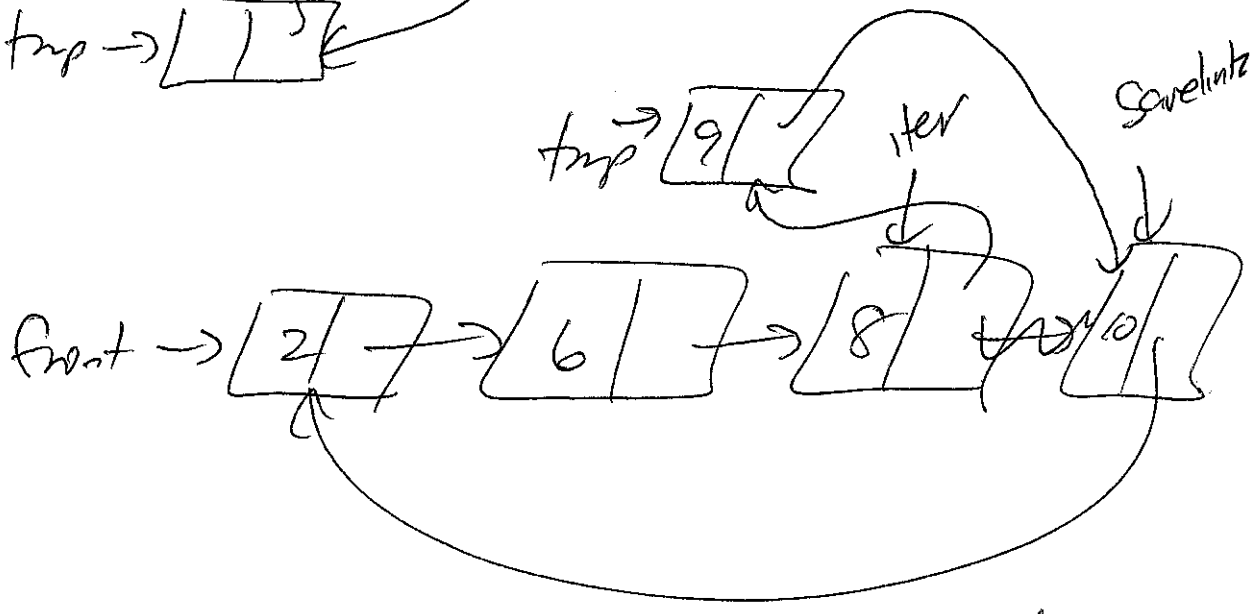
```
myQ->back->next = tmp;
myQ->back = tmp;
```



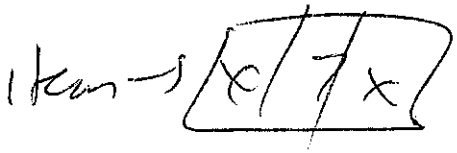
2/28/17 ②

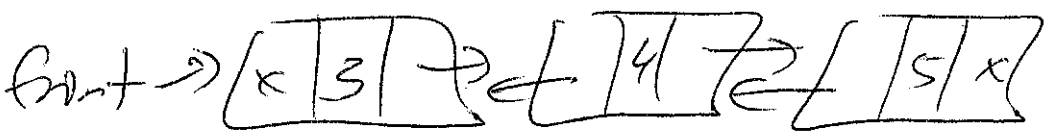
```
while (iter -> next != front)
    iter = iter -> next;
```



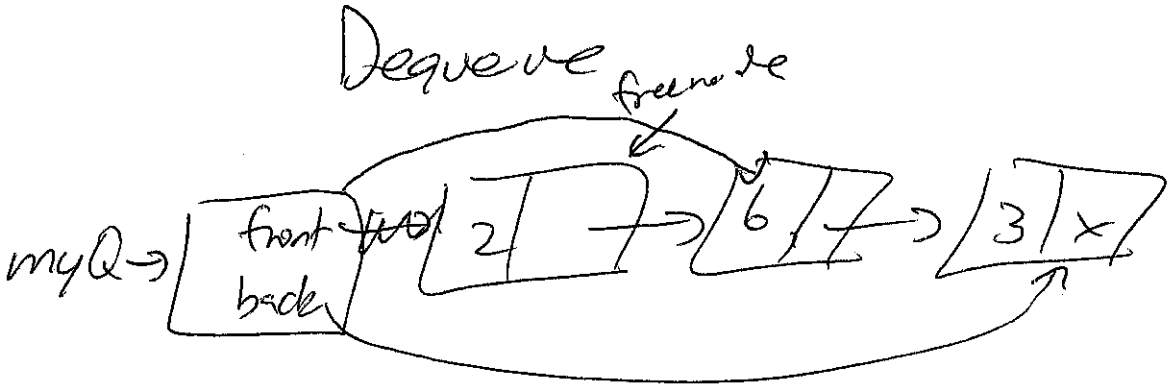



```
tmp -> next = Savelink;
iter -> next = tmp
return front;
```





Dequeue




```
int ~~Node*~~ retval = myQ->front->data;
node* freenode = myQ->front;
myQ->front = myQ->front->next;
free(freenode);
return retval;
```

(x,y) → struct w/ 2 components.

[0-(r-1)][0-c-1]
r x c

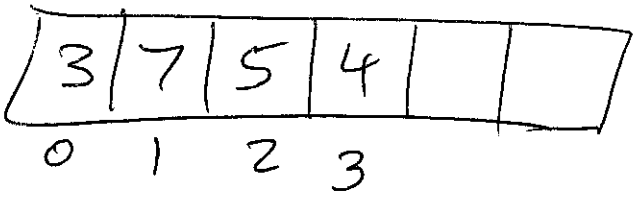
n*x+y

```
val = n*x+y
row = val / n;
col = val %n;
```

```
val = c*x+y
row = val / c;
col = val %c;
```

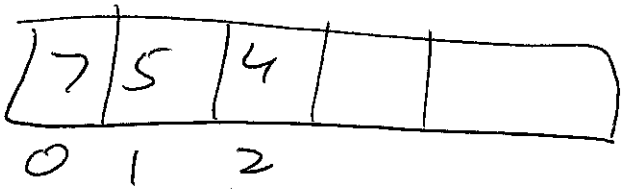
QUEUES

1. Linked List Implementation
2. Array Implementation

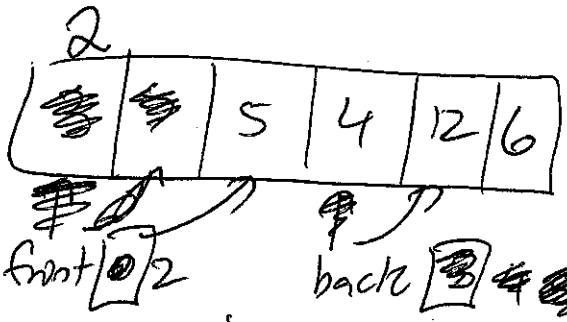


enqueue in order... O(1)

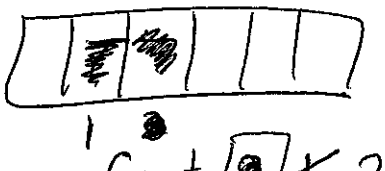
Imagine a deque.



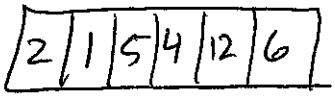
everyone steps forward O(n).




```
back = (back+1) % size;
front = (front+1) % size;
```



empty



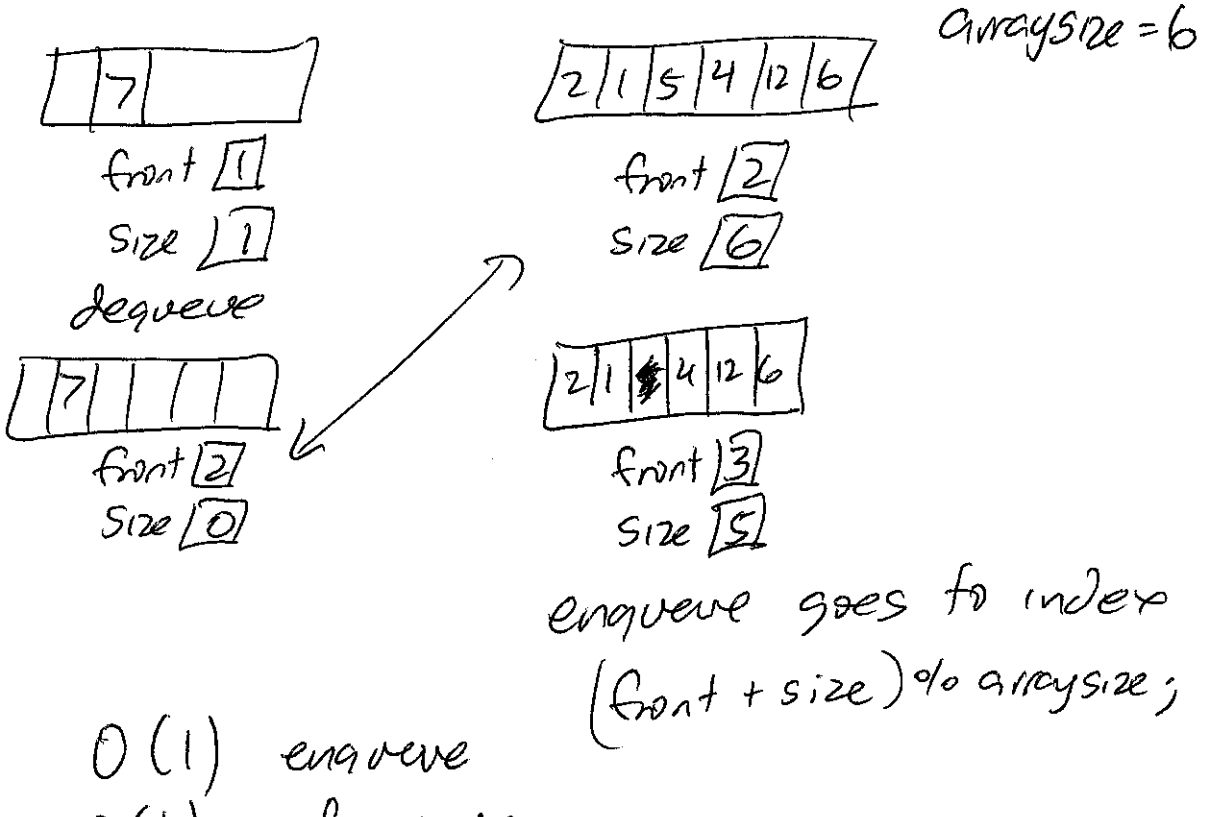
full

thorny issue

TWO SOLUTIONS

(1) if queue empty set front = -1, back = -1 as sentinel values.

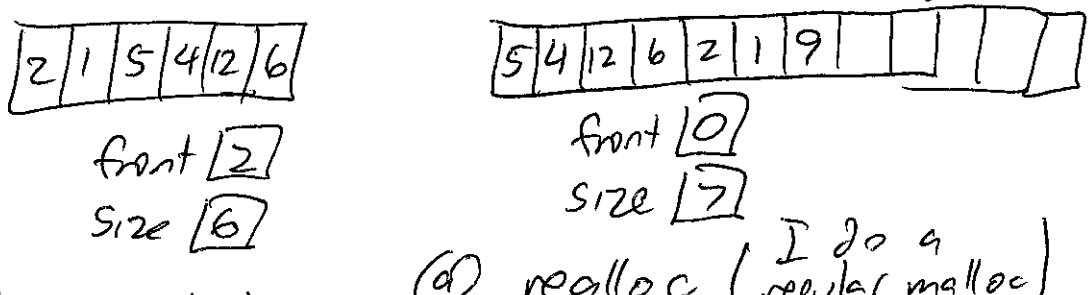
(2) DON'T STORE ~~FRONT~~ FRONT + BACK, STORE ~~FRONT~~ FRONT + SIZE



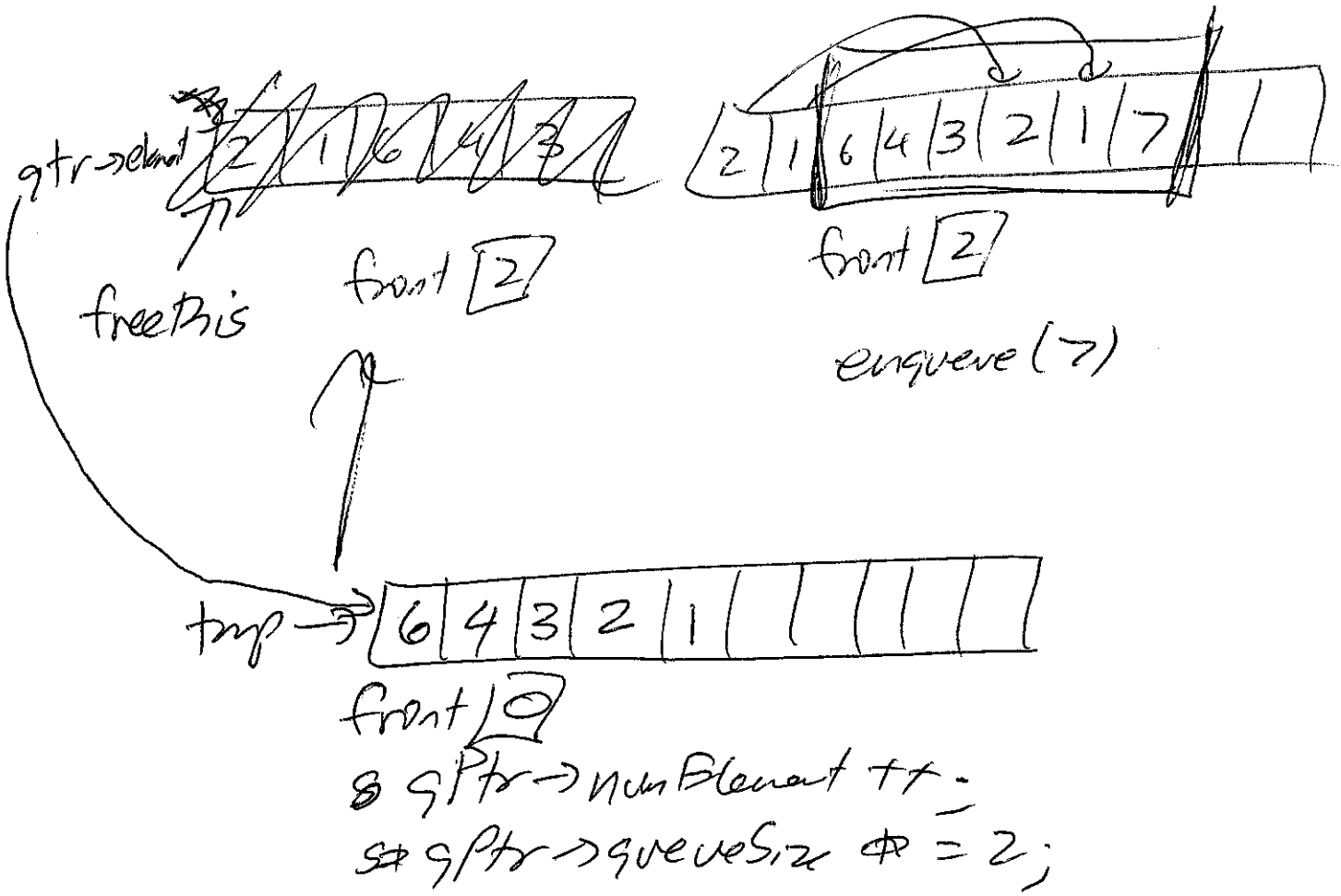
O(1) enqueue
O(1) dequeue

---

(3) WHAT IF QUEUE FILLS UP?



(a) realloc (I do a regular malloc)
(b) move front to 0
(c) copy accordingly
(d) size++;




```
qPtr->numElements++;
qPtr->queueSize *= 2;
```