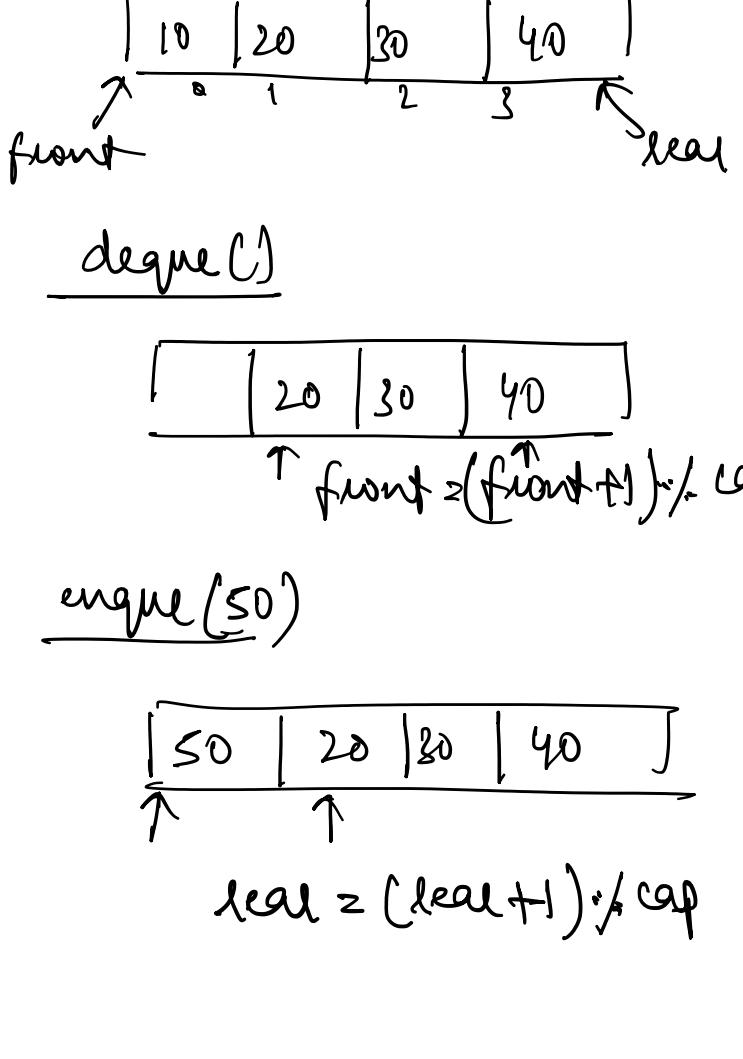


① Queue Implementation using Circular List → Advantages →  $O(1)$  time for all methods/operations  
 $q = \text{MyQueue}(\text{capacity})$ ;  $q.\text{enqueue}(x)$ ;  $q.\text{dequeue}()$ ;  $q.\text{getFront}()$ ;  $q.\text{getBack}()$ ;  $q.\text{isEmpty}()$

### # Circular List implementation of Queue

```
class MyQueue:
    def __init__(self, capacity):
        self.l = [None]*capacity
        self.cap = capacity
        self.size = 0
        self.front = 0
        self.back = 0
```

# class methods



$$\text{rear} = (\text{front} + \text{size} - 1) \% \text{cap}$$

```
def getFront(self):
    if self.size == 0:
        return None
    else:
        return self.l[self.front]

def getBack(self):
    if self.size == 0:
        return None
    else:
        rear = (self.front + self.size - 1) \% self.cap
        return self.l[rear]

def enqueue(self, x):
    if self.size == self.cap:
        return
    else:
        rear = (self.front + self.size - 1) \% self.cap
        rear = (rear + 1) \% self.cap
        self.l[rear] = x
        self.size = self.size + 1

def dequeue(self):
    if self.size == 0:
        return None
    else:
        rear = self.l[self.front]
        self.front = (self.front + 1) \% self.cap
        self.size = self.size - 1
        return rear

def isEmpty(self):
    if self.size == 0:
        return True
    else:
        return False

def isFull(self):
    if self.size == self.cap:
        return True
    else:
        return False

def size(self):
    return self.size
```

### ② Implement Stack Using Queue

Idea: Two queues:  $q_1$  → To keep the actual items  
 $q_2$  → To be used as an auxiliary queue

- $\text{push}(x)$ :  $q_1 = [x]$ ,  $q_2 = []$
- $\text{push}(x)$ :  $q_1 = [x]$ ,  $q_2 = [x]$
- $\text{pop}()$ :  $q_1 = [x]$ ,  $q_2 = [x]$
- $\text{top}()$ :  $q_1 = [x]$ ,  $q_2 = [x]$
- $\text{size}()$ :  $q_1 = [x]$ ,  $q_2 = [x]$
- (a)  $q_1 = [x]$ ,  $q_2 = [x]$
- (b)  $q_1 = [x]$ ,  $q_2 = [x]$
- (c) Move everything from  $q_1$  to  $q_2$
- (d)  $q_1 = [x]$ ,  $q_2 = [x]$
- (e) Move everything back from  $q_2$  to  $q_1$
- (f)  $q_1 = [x]$ ,  $q_2 = [x]$
- (g) Remove front of  $q_1$
- (h)  $q_1 = [x]$ ,  $q_2 = [x]$
- (i)  $q_1 = [x]$ ,  $q_2 = [x]$

Idea for optimized solution:

Two queues:  $q_1$  → To keep the actual items  
 $q_2$  → to be used as an auxiliary queue

- $\text{push}(x)$ : (a) Enqueue  $x$  to  $q_1$
- (b) Move everything from  $q_1$  to  $q_2$
- (c) Swap the variables  $q_1$  and  $q_2$
- $\text{pop}()$ : remove front of  $q_1$
- $\text{top}()$ : returns front of  $q_1$
- $\text{size}()$ : return size of  $q_1$

```
class Stack:
    def __init__(self):
        self.q1 = deque()
        self.q2 = deque()

    def push(self, x):
        self.q1.append(x)
        while self.q1:
            self.q2.append(self.q1.popleft())
        self.q1, self.q2 = self.q2, self.q1

    def pop(self):
        if self.q1:
            self.q1.popleft()

    def top(self):
        if self.q1:
            return self.q1[0]

    def size(self):
        return len(self.q1)
```

### ③ Reverse a Queue

Input:  $q = [20, 10, 15, 30]$

Iterative Solution

- ① Create an empty stack
- ② Move all items of  $q$  to stack
- ③ Move all items back to  $q$

def reverseQueue(q):

st = []

$\left\{ \begin{array}{l} q \text{ is an object of } \\ \text{deque} \text{ from collections} \end{array} \right.$

Time:  $\Theta(n)$

Space:  $\Theta(n)$

while q:
 st.append(q.popleft())
 q.append(st.pop())

Recursive Solution

```
def recQ(q):
    if len(q) == 0:
        return
    x = q.popleft()
    recQ(q)
    q.append(x)
```

Time:  $\Theta(n)$

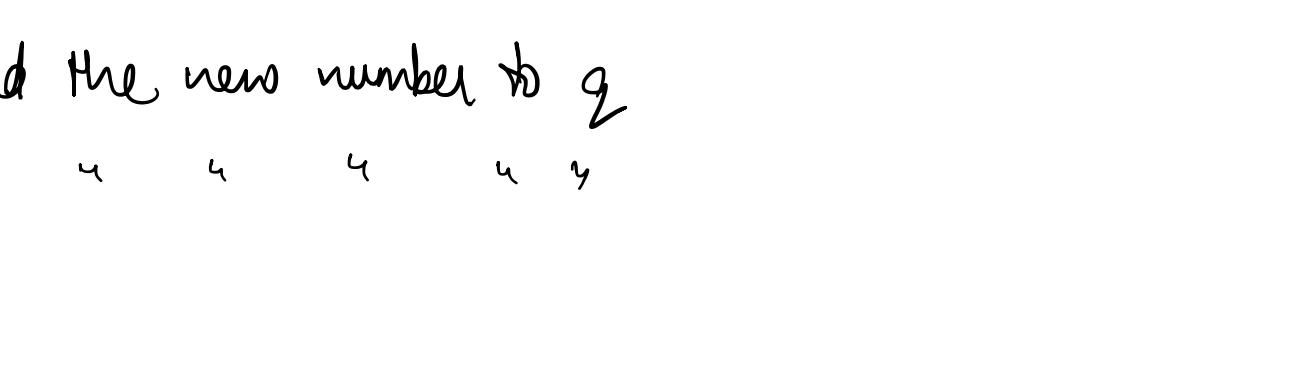
Space:  $\Theta(n)$

Native → Traverse through all natural numbers while we have not generated the  $n$  numbers

⇒ For every traversed number, check if it has digit as given only. If yes, print the number and increment the count.

### Idea for Efficient Alg

- ① Create an empty queue,  $q$
- ② Add 5 and 6 to  $q$
- ③ Run a loop  $n$  times:
  - ⇒ take out an item from  $q$
  - ⇒ print the item
  - ⇒ append 5 to the item and add the new number to  $q$
  - ⇒  $n = 6 \dots 11 \dots 16 \dots 21 \dots 26 \dots 31 \dots 36 \dots 41 \dots 46 \dots$



```
def printFirstN(n):
    q = deque()
    q.append('5')
    q.append('6')
    for i in range(n):
        curr = q.popleft()
        print(curr, end=" ")
        q.append(curr + '5')
        q.append(curr + '6')
```

### Better Implementation

```
def printFirstN(n):
    q = deque()
    q.append("5")
    q.append("6")
    q.append("56")
    i = 0
    while (i + len(q)) < n:
        curr = q.popleft()
        print(curr, end=" ")
        q.append(curr + "5")
        q.append(curr + "6")
    i += 1
    print(q.popleft(), end=" ")
```

### ④ Design a data structure with min/max operations

⇒ Design a data structure that supports following operations in  $O(1)$  times

- ① insertMin( $x$ )
- ② insertMax( $x$ )
- ③ getMin()
- ④ getMax()
- ⑤ extractMin()
- ⑥ extractMax()

```
from collections import deque
class MyDS:
    def __init__(self):
        self.dq = deque()

    def insertMin(self, x):
        self.dq.appendleft(x)

    def insertMax(self, x):
        self.dq.append(x)

    def getMax(self):
        return self.dq[-1]

    def extractMin(self):
        return self.dq.popleft()

    def getMin(self):
        return self.dq[0]

    def extractMax(self):
        return self.dq.pop()
```

Time:  $\Theta(1)$

Space:  $\Theta(n)$

Efficient: use doubly ended queue

```
def printKMin(arr, k):
    dq = deque()
    for i in range(k):
        dq.append(arr[i])
    for i in range(k, len(arr)):
        while dq and arr[i] >= arr[dq[-1]]:
            dq.pop()
        dq.append(arr[i])
        print(arr[dq[0]], end=" ")
    for i in range(k, len(arr)):
        while dq and dq[0] <= i - k:
            dq.popleft()
        while dq and arr[i] >= arr[dq[-1]]:
            dq.pop()
        dq.append(arr[i])
        print(arr[dq[0]], end=" ")
```

Time:  $\Theta(n)$

Space:  $\Theta(n)$

### ⑤ Find CircularTour

Input:  $\text{petrol}[] = \{4, 8, 7, 4\}$

subarray size  $\Rightarrow k$

$\text{dist}[] = \{6, 5, 3\}$

$\text{dist}[] = \{5, 6, 3\}$

$\text{dist}[] = \{5, 10, 12\}$

$\text{dist}[] = \{-1, 12, 8\}$

$\text{dist}[] = \{8, 9, 4\}$

$\text{dist}[] = \{5, 10, 12\}$

$\text{dist}[] = \{5, 1$