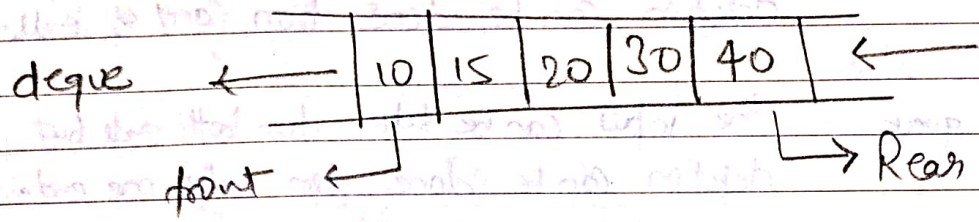
Queue

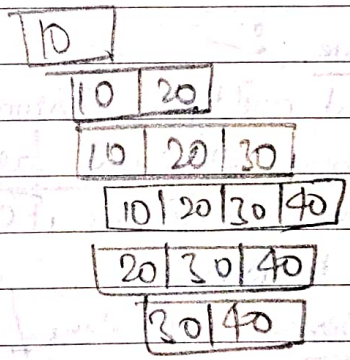
→ Queue is a linear data structure that is open at both ends and the operations are performed in first-in-first-out (FIFO) order



Operations :—

enqueue (x) — to insert the element
dequeue () — to delete the delete
getFront () — to get front element
getRear () — to get rear element
Size () — to know the size
isEmpty — to know whether it is empty or not → true / false

```
Queue q
q.enqueue (10)
q.enqueue (20)
q.enqueue (30)
q.enqueue (40)
print (q.dequeue ())
print (q.dequeue ())
print (q.getFront ())
print (q.getRear ())
```



o/p
→ 10
→ 20
→ 30
→ 40

JULY
18
'24
THURSDAY
29th Week
200-166

S	M	T	W	T	F	S
	1	2	3	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			

JUL 2024

Types of Queue :-

08

(1) input restricted queue — the input can be taken from only one end but
09 deletion can be done from any of ends.

(2) output restricted queue — the input can be taken from both ends but deletion can be done from only one end.

11

(3) Circular queue — In these where the last position is connected back
12 to the first positions.

(4) Double ended queue — insertion & deletion operations, both can be performed from both ends.

02

(5) Priority queue — Special queue where the elements are accessed
03 based on the priority assigned to them.

04 # Application of Queue :-

(*) Single Resources and multiple Consumers [Cpu Scheduling & Disk Scheduling]
05 (*) Synchronization b/w slow and fast devices.
(*) In operating System (semaphores, FCFS scheduling, spooling, buffers
06 for devices like keyboard).
(*) In Computer network (Routers / switches and mail queues)
07 (*) Variations : Deque, Priority queue and Doubly ended Priority queue.

Implementation in python.

(a) using list
(b) using collections.deque
(c) using queue.Queue → Multithreaded Environment.
(d) our own implementation.

Using list

```
q = []
q.append(10)  # [10]
q.append(20)  # [10,20]
q.append(30)  # [10,20,30]
print(q)
print(q.pop(0))  # [20,30]
q.append(40)  # [20,30,40]
print(q.pop(0))  # [30,40]
print(len(q))
print(q[0])
print(q[-1])
```

TC = O(1)

o/p —
[10, 20, 30]
2
30
40

TC = O(n)

Using Deque

```
from collections import deque
q = deque()  # deque([])
q.append(10)  # deque([10])
q.append(20)  # deque([10,20])
q.append(30)  # deque([10,20,30])
print(q)
print(q.popleft())  # deque([20,30])
q.append(40)  # deque([20,30,40])
print(q.popleft())  # deque([30,40])
print(len(q))
print(q[0])
print(q[-1])
```

TC = O(1)

o/p — deque([10,20,30])
2
30
40

TC = O(1) ✓

JULY
20 '24
29th Week
202-164
SATURDAY

S	M	T	W	T	F	S
	1	2	3	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			

JUL 2024

TC = O(1)

Using Linked list

```
Class Node:
    def __init__(self, k):
        self.key = k
        self.next = None

Class MyQueue
    def __init__(self):
        self.front = NONE
        self.rear = NONE
        self.s2 = 0

    def size(self):
        return self.s2

    def isEmpty(self):
        return (self.s2 == 0)

    def getFront(self):
        return self.front.key

    def getRear(self):
        return self.rear.key

    def enque(self, n):
        temp = Node(n)
        if self.rear == NONE:
            self.front = temp
        else:
            self.rear.next = temp
        self.rear = temp
        self.s2 = self.s2 + 1

    def deque(self):
        if self.front == None:
            return NONE
        else:
            res = self.front.key
            self.front = self.front.next
            if self.front == NONE:
                self.rear = NONE
            self.s2 = self.s2 - 1
            return res
```

Implement stack using Queue:—

We will take 2 queue:
q_1 : To keep the actual items.
q_2 : To be used as an auxilary queue.

push(x): (a) Move everything from q_1 to q_2.
(b) Enqueue x to q_1
(c) move everything back from q_2 to q_1

pop(x): Remove front of q_1
top() : return front of q_1
size() : return size of q_1

push(10): $q_1 = [10]$, $q_2 = []$

push(20):
(a) $q_1 = []$, $q_2 = [10]$
(b) $q_1 = [20]$, $q_2 = [10]$
(c) $q_1 = [20, 10]$, $q_2 = []$

push(30):
(a) $q_1 = []$, $q_2 = [20, 10]$
(b) $q_1 = [30]$, $q_2 = [20, 10]$
(c) $q_1 = [30, 20, 10]$, $q_2 = []$

In optimize Solution of Queue implementation is
Instead of this we use
push(x): (a) Enqueue x to q_2.
(b) move everything from q_1 to q_2.
(c) Swap the variable q_1 and q_2.

Code

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

    def push(self, x):
        self.q2.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]
        else:
            return None

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

TC = O(n)

pop() : O(1)

top() : O(1)

size() : O(1)

JULY
22
'24
30th Week
204-162
MONDAY

S	M	T	W	T	F	S
	1	2	3	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			

JUL 2024

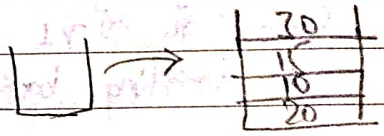
Reverse a Queue

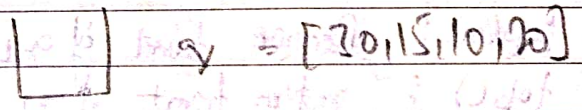
i/p: q = [20, 10, 15, 30]
o/p: q = [30, 15, 10, 20]

M-1
```
from collections import deque
def reverseQueue(q):
    st = []
    while q:
        st.append(q.popleft())
    while st:
        q.append(st.pop())
```
TC = O(n)
SC = O(n)

M-2
```
def revQ(q):
    if len(q) == 0: return
    n = q.popleft()
    revQ(q)
    q.append(n)
```
TC = O(n)
SC = O(n)

Iterative Solution
① create an empty stack.
② move all item of q to stack.
③ move all items back to the q.



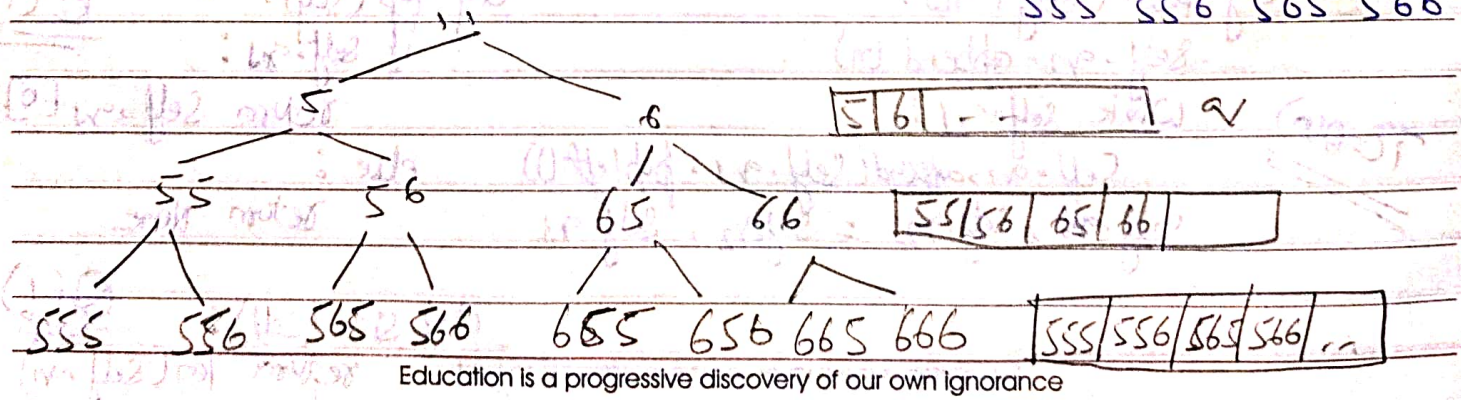


Naive Sol^n
① Traverse through all natural No while we have not generated the n numbers.
② for every Traversed No check if it has digit as 5 and 6 only. If yes, Print the No. and increasing the Count.

Generate number with Given digits.

→ Given a value K, generate all well-ordered numbers of length k.
→ Well ordered means that digit should be in increasing order.

eg i/p: [5,6], n=10
o/p: 5 6 55 56 66 555 556 565 566



Steps

1. Create an empty queue, q
2. Add 5 and 6 to q.
3. Run a loop n times.
   - → Take out an items from q.
   - → Print the item.
   - → Append 5 to the item and add the new numbers to q.
   - → " 6 " " " " " " " " " " ".

```
def PrintFirst(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")
```

Design a Data structure with max/min operation.

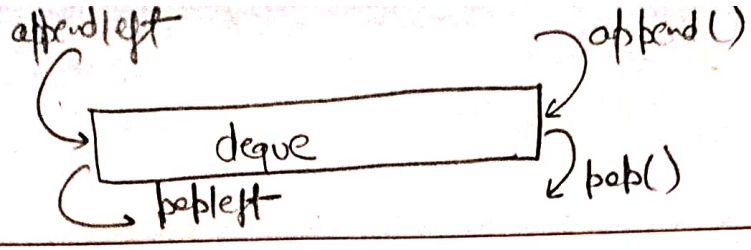
→ Design a data structure that support following operation in O(1) Time complexity.

	i/p:	o/p:
1. insertMin(n)	insertMin(5)	
2. insertMax(n)	insertMax(10)	2
3. getMin()	insertMin(3)	15
4. getMax()	insertMax(15)	1
5. extractMin()	insertMin(2)	20
6. extractMax()	getMin()	
	getMax()	
	insertMin(1)	
	getMin()	

Each one sees what he carries in his heart.

insertMax(20)
getMax()

JULY
24 '24
30th Week
206-160
WEDNESDAY



S	M	T	W	T	F	S
	1	2	3	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			

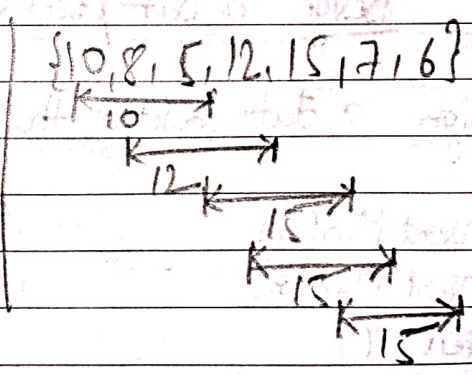
JUL 2024

```
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 extractMin(self):
        return self.dq.popleft()
    def extractMax(self):
        return self.dq.pop()
    def getMin():
        return self.dq[0]
    def getMax():
        return self.dq[-1]
```

```
d = MyDS()          deque[]
d.insertMin(10)     deque([10])
d.insertMin(5)      deque([5,10])
d.insertMax(20)     deque[5,10,20]
d.insertMin(3)      deque[3,5,10,20]
print(d.extractMin(1))   # 3
print(d.extractMax(1)    # 20
print(d.getMin())   # 5
print(d.getMax())   # 10
```

Maximum in all subarrays of size K.

i/p: arr[] = {10, 8, 5, 12, 15, 7, 6}, K = 3,
o/p: 10 12 15 15 15



General Formula = (n-K+1)
(n-K+1) → arr length
K → size of window.

Naive Solution

```
def printMaxK(arr, K):
    for i in range(n-K+1):
        res = arr[i]
        for j in range(i+1, i+K):
            res = max(res, arr[j])
        print(res, end = " ")
```

TC = O(n*K)
Aux space = O(1)

Efficient : O(n) K=3, {10, 8, 5, 12, 15, 7, 6}

08 i=0, dq [10]
i=1, dq [10|8]
09 i=2, dq [10|8|5] — 10
i=3, dq [ ] — 12

i=4, dq [15] — 15
i=5, dq [15|7] — 15
i=6, dq [15|7|6] — 15

10 ↳ Whenever we see an item we remove all the smaller than it from deque.

11
```
def printKMax(arr, K):
12      dq = deque()
        for i in range(K):
            while dq and arr[i] >= arr[dq[-1]]:
01              dq.pop()
02          dq.append(i)
        print(arr[dq[0]], end=" ")
03      for i in range(K,n):
            while dq and dq[0] <= i-K:
04              dq.popleft()
            while dq and arr[i] >= arr[dq[-1]]:
05              dq.pop()
            dq.append(i)
06          print(arr[dq[0]], end=" ")
```

arr = [20, 40, 30, 35, 60] (0 1 2 3 4)

K=3
i=0: [0| |]
i=1: [1| |]
i=2: [1|2|] print(40)

Inside the second loop:
i=3: [1|3|], print(40)
i=4: [4| |] print(60)

T.C = O(n)

07 # First Circular Tour.

→ Given information about N petrol pumps (says arr[]) that are present in a circular path. The information consists of the distance of the next petrol pump from the current one (in arr[i][1]) and the amount of petrol stored in that petrol pump (in arr[i][0]). Consider a truck with infinite capacity that consumes 1 unit of petrol to travel 1 unit distance. The task is to find the index of the first starting point such that the truck can visit all the petrol pumps & come back to that starting point.

JULY

26

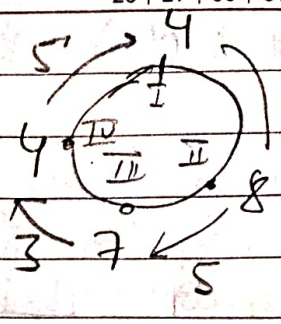
FRIDAY

'24

30th Week
208-158

S	M	T	W	T	F
	1	2	3	4	5
7	8	9	10	11	12
14	15	16	17	18	19
21	22	23	24	25	26
28	29	30	31		

i/p: petrol[] = {4, 8, 7, 4} { o/p: 2 {
dist[] = {6, 5, 3, 5}



Solve

```
def printTour(pet, dist):
    n = len(pet)
    for i in range(n):
        curr_pet = 0
        end = i  # i = start
        while True:
            curr_pet += pet[end] - dist[end]
            if curr_pet < 0:
                break
            end = (end + 1) % n
            if end == start:
                return start + 1
    return -1
```