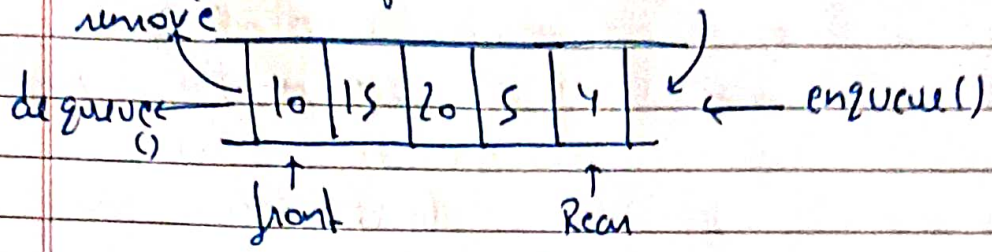
Queue → Pipe

- FIFO (first in first out)



- Removal operation done from front is called dequeue()
- Addition ——— Rear ——— enqueue()

- getFront() - to get front item
- getRear() ——— rear ———
- size() - Queue size return
- isEmpty() - return bool value, 1 if empty, 0 if empty.
- isFull() -

language Independent code

eg: Queue q

		o/p
q.enqueue(10)	10	
q.enqueue(20)	10 20	
q.enqueue(30)	10 20 30	
q.enqueue(40)	10 20 30 40	
print(q.dequeue())	20 30 40	10
print(q.dequeue())	30 40	20
print(q.getFront())		30
print(q.getRear())		40

we create Queue of Tickets

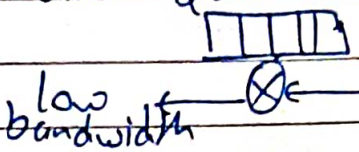
→ Application of Queue :

① Single Resource and Multiple Consumers (eg: Ticket के लिए Queue में लगते हैं, और पहले आया हुआ ticket) in computer also ~~that~~ process scheduling also, first come first serve follow

* Multiple Consumers के लिए Queue (क्योंकि जो पहले आएगा वो पहले process होगा)
* Synchronization in Queue
  of slow & fast device

② Synchronization between slow & fast devices.
eg: (Keyboard & printer work in synchronize way)
slow device fast
. character based are stored in Queue & CPU ~~process it or by one~~ comes takes a cycle & process data Quickly.

* In Router also Queue is used



here Queue stores some data b/z sender sends data fast & Receiver consumes slow. Queue stores data.

③ In OS (Semaphore (there are sleeping process waiting in a Queue when resource become available one of process wakes up and use the resource), FCFS (first come first serve), Spooling (used in printer b/z printer serves one page at a time), buffer for device like Keyboard.

④ In Mail Queue (mail server), in CN जहाँ भी Server होता वहाँ Queue भी होता क्योंकि consume करना है।

⑤ Variation → Deque, Priority Queue, Double Ended Priority Queue

We will study Array concept

↓

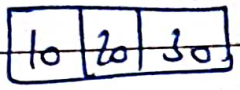
→ Array Implementation of Queue : → all O(1)

```
struct Queue
{   int *arr;
    int front, cap, size;
    Queue(int c)
    {   arr = new int[c];
        cap = c;
        front = 0;
        size = 0;
    }
    bool isFull() { return (cap == size); }
    bool isEmpty() { return (size == 0); }

    int getFront()
    {   if (isEmpty())
            return -1;
        else
            return front;
    }

    int getRear()
    {   if (isEmpty())
            return -1;
        else
            return (front + size - 1) % cap;
    }

    void enqueue(int x)
    {   if (isFull()) return;
        int rear = getRear();
        rear = (rear + 1) % cap;
        arr[rear] = x;
        size++;
    }
```

```
void dequeue()
{   if(isEmpty()) return;
    front = (front+1) % cap;
    size--;
}
```

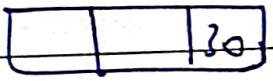
eg cap=3, size=0, front=0

enqueue(10), enqueue(20), enqueue(30)



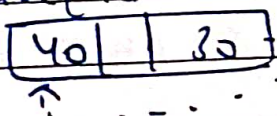
size=3, front=0

dequeue(), dequeue()



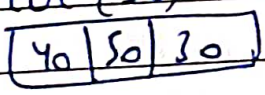
size=1, front=2

enqueue(40)



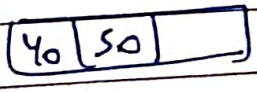
size=2, front=2

enqueue(50)



size=3, front=2

dequeue()



size=2, front=0

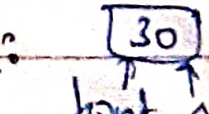
→ **Linked list implementation of Queue:**

```
struct Node
{   int data;
    Node *next;
    Node(int x)
    {  data=x;
       next=NULL;
    }
};
```


```
struct Queue
{  Node *front, *rear;
   int size;
   Queue()
   {  front = NULL;
      rear = NULL; size = 0; }
   void enqueue(int x)
   {
      Node *temp = new Node(x); size++;
      if(front == NULL)
      {  front = rear = temp;
         return;
      }
      rear->next = temp;
      rear = temp;
   }
   void dequeue()
   {  if(front == NULL) return;
      size--;
      Node *temp = front;
      front = front->next;
      if(front == NULL)
         rear = NULL;
      delete(temp);
   }
int void getSize() { return size; }
   int getFront { return front->data; }
   int getRear { return rear->data; }
```

deq can still start

Corner Case:



dequeue()

→ front = rear = NULL

enqueue(10)

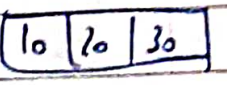
given front = rear = NULL

all O(1)

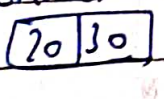
Queue in STL :

1. q.push(x) : add element x at the end of queue
2. q.pop() : delete last " of the queue
3. q.size() : return size of queue
4. q.empty() : return 1 if q emph or 0.
5. q.front() : return reference to the first element of the q
6. q.back() : ——————— last ———————
7. q.swap() : exchange content of 2 diffrent queue (q1.swap(q2)).
8. q.emplace() : new element is added to the end of queue

```
queue<int> q;
q.push(10);
q.push(20); q.push(30)
cout << q.front() << " " << q.back() << endl;
q.pop()
```



10 30



Traversing a Queue (we remove element from Queue)

```
while(q.empty() = false)
{   cout << q.front() << " "; }
    q.pop(); }
```

10 20 30

* Queue is implemented using Dequeue.
* Queue is also contains adaptor.

Reverse करना है - Stack

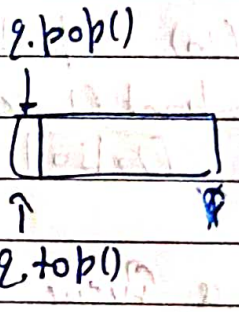
(I) Reverse a Queue :

I/P {10, 5, 9, 13} O/p : {13, 9, 5, 10}

a) Iterative (Using stack)

```
void f(queue<int> q)
{   stack<int> st;
    while(!q.empty())
    {   st.push(q.top());
        q.pop();
    }
    while(!st.empty())
    {   q.push(s.top());
        s.pop();
    }
}
```

b) Recursive




```
void reverse(queue<int>&q)
{
    if(q.empty())
        return;
    int x = q.top();
    q.pop();
    reverse(q);
    q.push(x);
}
```

(II) Gas Station

- There are n gas station in circular route.
- gas[i] → gas at i^{th} station
- cost[i] → cost to travel from i^{th} sta. to $(i+1)^{th}$ station
  of gas
- return index from which you can travel whole circle
  min

```
int f(vector<int> &g, vector<int> &c)
{  int total=0, fuel=0, ans=0;
   for(int i=0 ; i<g.size(); i++)
   {
      fuel += g[i]-c[i];
      if(fuel<0)
      {  fuel=0;
         ans=i+1; // new answer may be
      }
      total += g[i]-c[i];
   }
   return (total<0?-1:ans);
}
```

-1 तब आएगा जब total sum of petrol/gas less than distance/cost होगा।

eg: [1,2,3,4,5] → g o/p → index = 4
[3,4,5,1,2] → c

i=0 fuel=-2	i=1 fuel=-2	i=2	i=3	i=4
fuel<0	(-2<0) fuel=0	(-2<0) fuel=0	fuel=3	fuel=6
fuel=0	fuel=0	fuel=0	total=-6+3	total=-3+3
ans=1	ans=2	ans=3	=-3	=0
total=-2	total=-2+(-2)=-4	total=-4+(-2) =-6		

total<0 ✗ return ans=4

eg: [[ ]] — fuel<0 → new ans यहाँ होगा

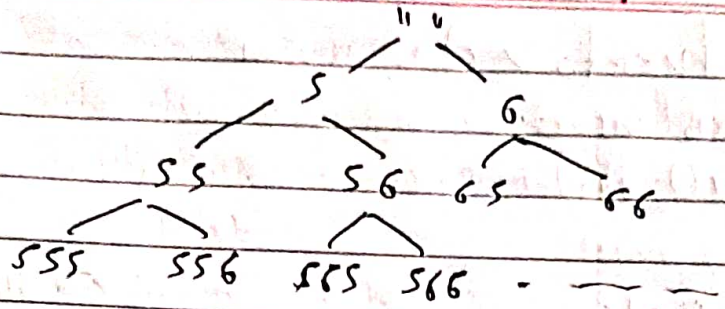
Let say ans=3 is answer but आगे जाके fuel<0 होजाता तो उसके पहले के index से ans नहीं होगा आगे तक जाएगा i++

(III) Generate number with given digits :

Given a no. n, print n no. in increasing order such that all these numbers have digits in set(5,6).

eg: n=10

o/p 5, 6, 55, 56, 555, 556, 565, 566, 5555, 5556




```
vector<string> f(int n)
{
    vector<string> ans;
    queue<string> q;
    q.push("5");
    q.push("6");
    for(int i=0; i<n; i++)
    {
        string curr = q.top();
        ans.push_back(curr);
        q.pop();
        q.push(curr+"5");
        q.push(curr+"6");
    }
}
```

n=4

i=0 | 5 | 6 | | ← q | 5 | | ← y

| 6 | 55 | 56 |

i=1 | 6 | 55 | 56 | | 5 | 6 |

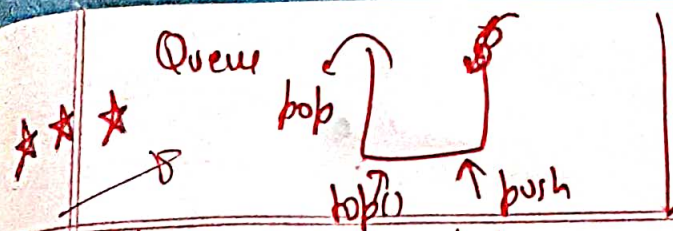
| 55 | 56 | 65 | 66 |

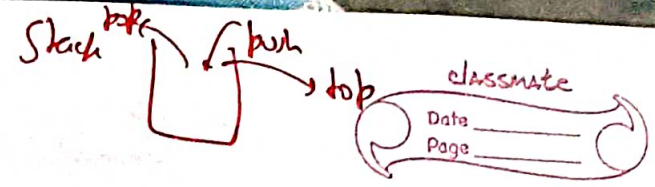
i=2 | 55 | 56 | 65 | 66 | | 5 | 6 | 55 |

| 56 | 65 | 66 | 555 | 556 |

i=3 | 56 | 65 | 66 | 555 | 556 | | 5 | 6 | 55 | 56 | ← Ans

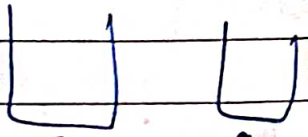
| 65 | 66 | 555 | 556 | 565 | 566 |





→ Stack Implementation using a ~~Single~~ Queue :

a) Using Two Queue



push(x)
add x to Q2
Q1 → Q2 (element by element)
swap Q1 ⇌ Q2

O(n)

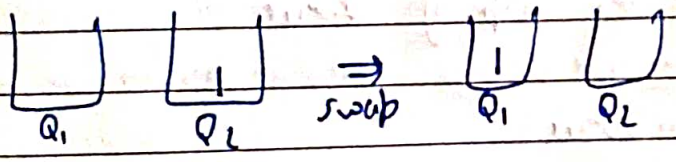
S(n) O(n)

pop()
remove top of Q1

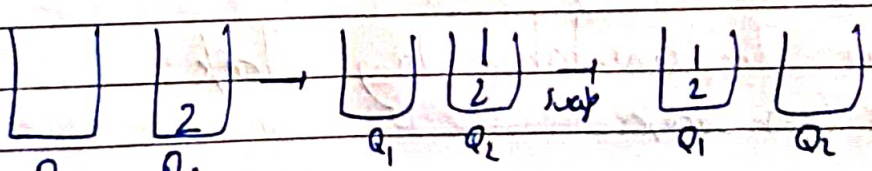
TC → O(1)

eg: push(1), (push(2), push(3), push(4), pop() , top(), pop()

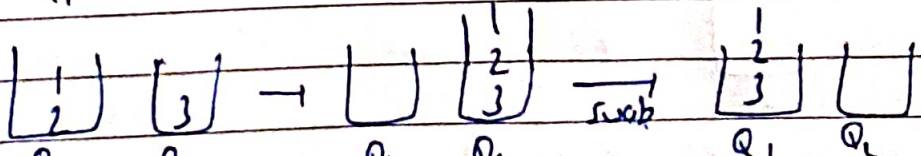
push(1)



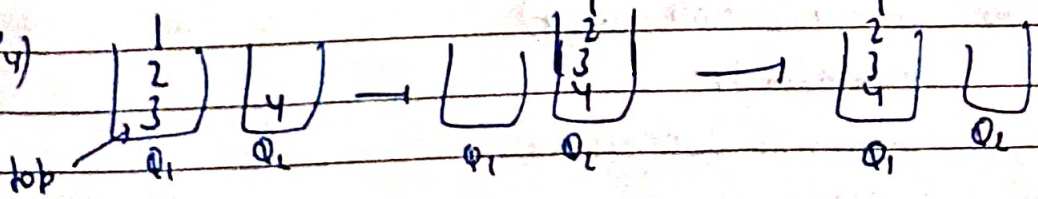
push(2)



push(3)

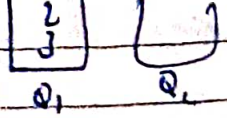


push(4)



top() ~~top()~~ when Q1.top() → 4

pop()



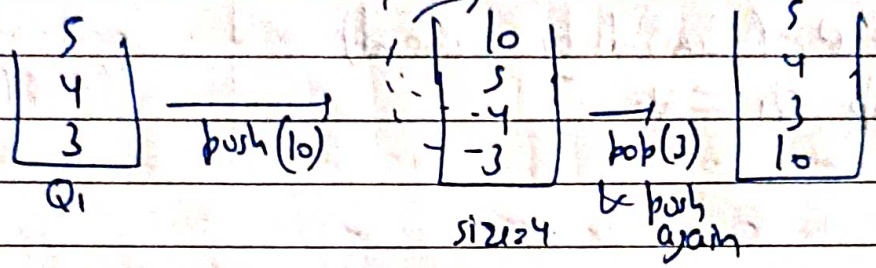
simply remove top() of Q1 (Q1.pop())

~~top()~~

b) Using one Queue

push(x) | pop()

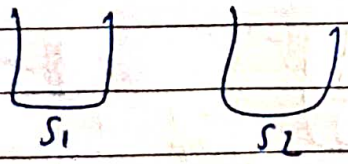
O(n) simply push in Queue
S-O(n) Q.push(x)
then push the elements i from (size-1) again in Queue



O(1) → pop()
simply Q.pop() removes top of Queue
→ top()
Q.top() return top of Queue

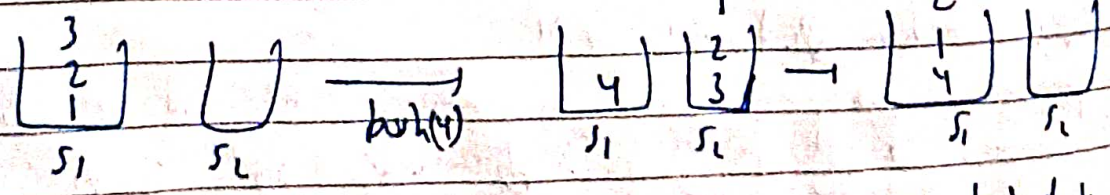
→ Queue Implementation Using Stack:

a) Using Two Stack.



push(x) O(n) S-O(2n)
do S1 → S2 (all element)
push(x) in S1
S2 → S1 (all element)

Eg



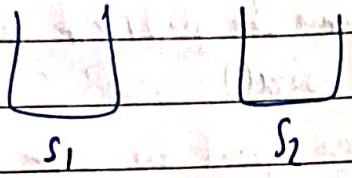
4 is pushed in Queue

O(1)

pop()	top()
S_1.pop()	S_1.top()

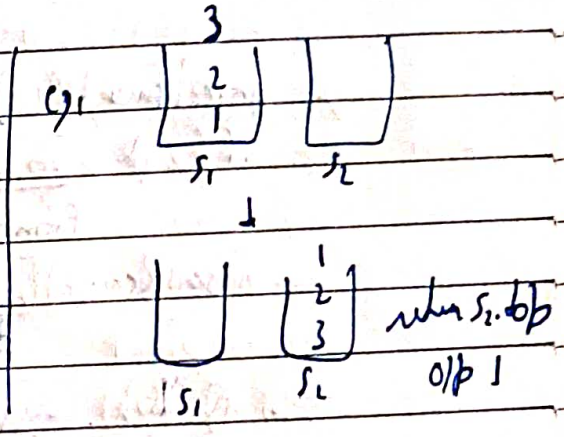
b) ~~push(x)~~ Using two Stack only code & operation is less

push(x)
simply push to S_1 O(1)



~~pop()~~
~~top() = O(n)~~

- If S_2 == empty
  push S_1 in S_2 then return S_2.top()
- If S_2 != empty
  return S_2.top()



pop()

- If S_2 == empty
  push S_1 in S_2 then ~~return~~ do S_2.pop()
- If S_2 != empty
  simply S_2.pop()

* top() & pop() are O(1) amortized complexity (means maximum वाला O(1) होगा but हमेशा नहीं)