Batch PSP 54.77 $\longrightarrow$ 60%

Nov23_PSP_20March
Vijay V A
Yash Malviya
Manjunatha I
Sai Sharath
Harshil Dabhoya
Kevin Theodore E
Mayur Hadawale
Manikandan M
kameswarreddy Yeddula
Sarat Patel
ALLEN GEOSHAN M
Shaurya Srivastava
Suraj Devraye
sudhakar venkatachalam
Vigneshwaran K

Nov23_PSP_20March
barani r
Rajeev
Pranadarth S
MD JASHIMUDDIN
Phaneendra Gandla
Nitendra Rajput
Rsr Ram
SIJU SAMSON
Prabhakar
Prashant Kumar Soni
Pravin Raj
Pratham Singh
Robin Dhiman
Pushkar Deshpande
Tushar Desarda

Agenda

a) Basics and Implementation

b) Perfect number Question

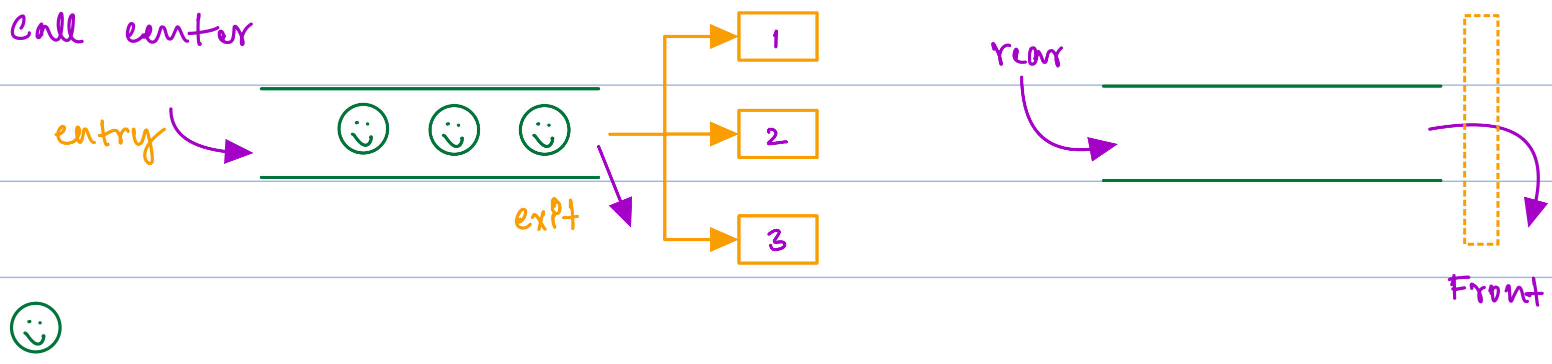
c) Doubly ended Queue

d) Sliding window maximum

Queue

Linear data structure where data is added in one end and removed in another end

Call center



FIFO → First in First out

Common Operations

Enqueue : add element to rear of the queue

Dequeue : remove and return the front element

isEmpty : checking queue is empty or not.

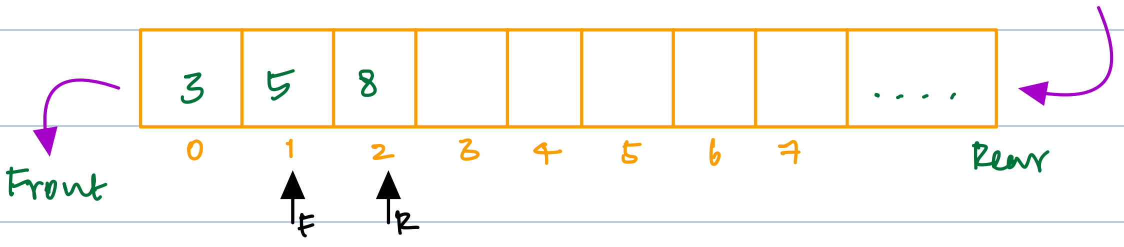
Peek : returns the value of element at front

Size : length of queue

Can a stack or Queue ever be full?

100 % Yes bounded queue

Implementation of Queue using Dynamic Arrays



F = 0   R = -1

Operations

enqueue (3)

enqueue (5)

enqueue (8)

dequeue ()

is empty ()

Do you visualize that at any point a Queue is a subarray in arraylist ?

100% Yes

pseudo code

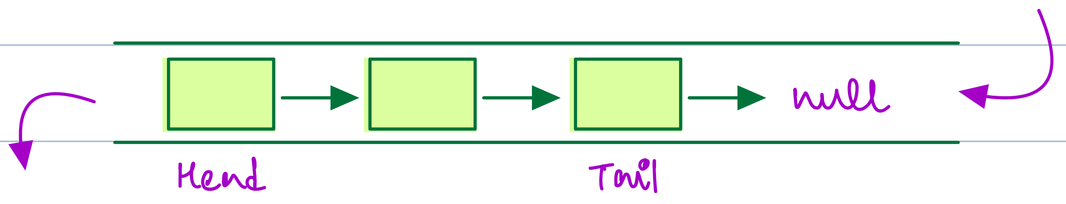
```
// Initialization

  f = 0,  r = -1

void  enqueue ( x ) {
    r += 1;
    arr [r] = x;
}

boolean  is empty ()
    return  f > r;
```

```
int  dequeue () {
    if (queue. isEmpty())
        return -1;
    int tmp = arr [f];
    f += 1;
    return tmp;
}
```

T.C = O(1)

S.C = O(n)

Implement Queue using linked list



Do we need to maintain two pointers?

100% Yes, we maintain Head (F)

Tail (R)

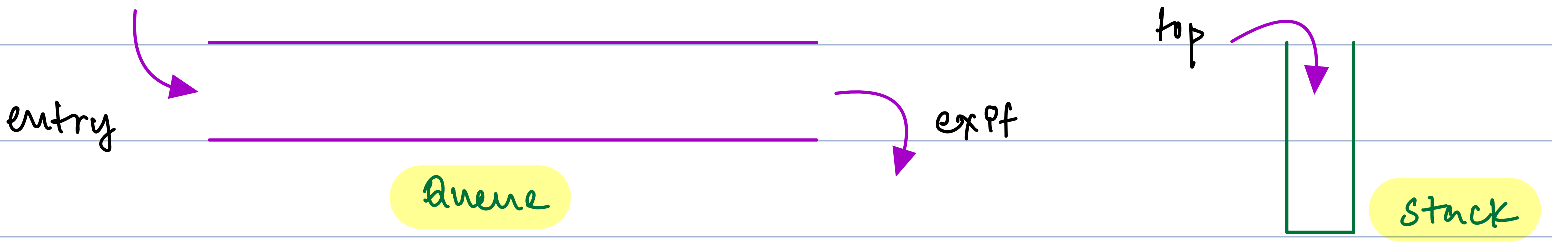
```
// Initialize
    Head = null    Tail = null

// enqueue at tail
    // Handle Head and Tail null
    if (Head == null)    Head = Tail = nnode;
    Tail.next = nnode;   Tail = nnode

// dequeue from head
    if (Head == null) return -1;
        Head = Head.next;
```

T.C = O(1)

```
// is empty
    return Head == null
```

Implement Queue using two stacks ※ ※.

enqueue,	push
dequeue,	pop
is empty	is empty



Operations

enqueue (1)

enqueue (2)

enqueue (3)

dequeue ()

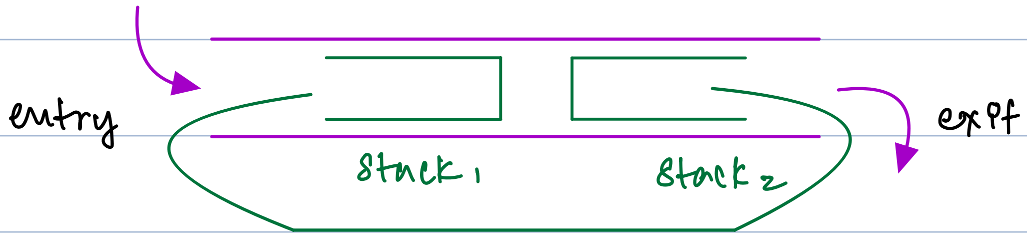
enqueue (4)

dequeue ()

dequeue ()

dequeue ()

Visualize



Dequeue ()

1. If I have data in st2
2. If not, move all data from st1 to st2 ← O(n)
3. If yes, blindly remove from stack 2 ← O(1)

pseudo code

```
// is empty ()
boolean is empty ()
        return st1. isEmpty () && st2. isEmpty()

// enqueue
void enqueue (int x) {
        st1. push (x);
}

// dequeue
int dequeue () {
        if (Q. isEmpty ()) return -1; // underflow
        if (st2. isEmpty ()) move ()
        return st2. pop ()
}

void move () {                          10:15 → 10:20
        while (!st1. isEmpty ())
            st2. push ( st1. pop ());
}
```

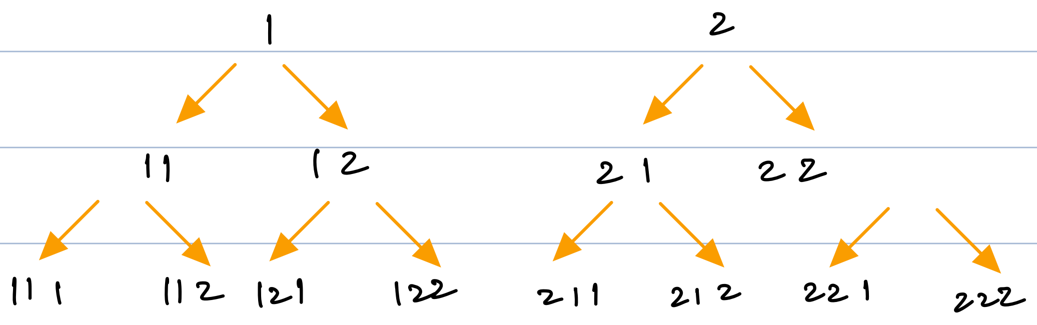
Question

Given an integer N, find Nth number that can be formed using 1 & 2

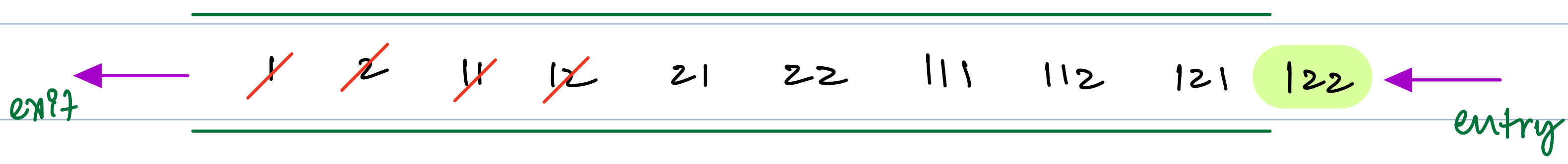
N=

1	2	11	12	21	22	111	112	121	122	211
1	2	3	4	5	6	7	8	9	10	11

Visualize



First come first serve



N=10

pseudo code

```
int  getPerfectNumber (int N) {
    if (N <= 2) return N;
    // intialize Queue
     q.enqueue (1);   q.enqueue (2);
     i=3
     while (i<=N) {
             x = q.dequeue ();
             n1 = x * 10 + 1;
             n2 = x * 10+2;
              if (i==N) return n1;
              if (i+1==n) return n2;

             q.enqueue (n1);
             q.enqueue (n2);
             i= i+2;
     }
}
```

Doubly ended Queue

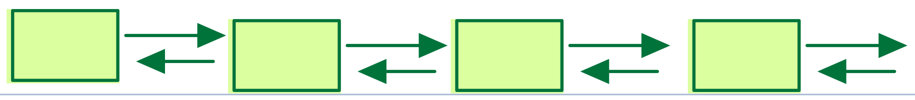
Linear data structure that allows entry and exit from both front and rear ends

Operations possible

enqueue_front, enqueue_rear

dequeue_front, dequeue_rear

is empty ()



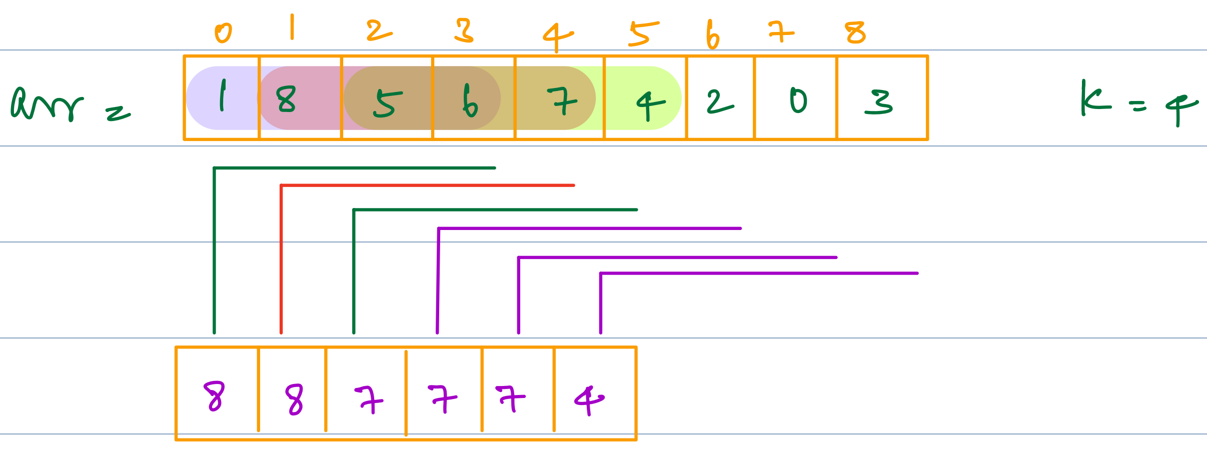
Head tail

For implementing a Doubly ended Queue you will use a Doubly linked list

Question

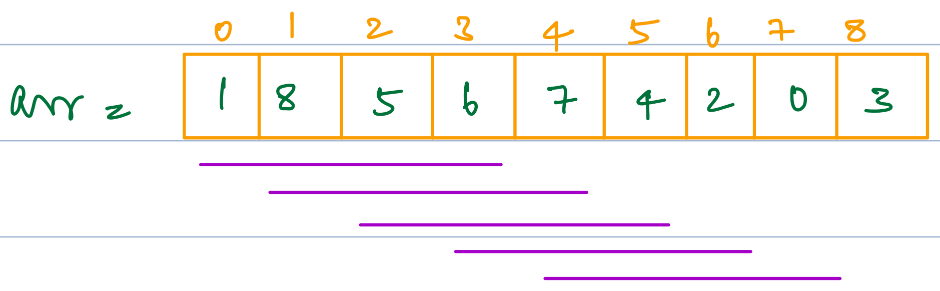
Given an integer array, and a window size k find the max element for each window

example



Observation 1

sliding window because of fixed size window



~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5 ~~6~~ ~~7~~ 8

potential ans

arr[1], arr[1], arr[4], arr[4], arr[5]

Observation 1

We use a doubly ended queue for identifying the max element in a window

Observation 2

Store index of array instead of value to verify what we are removing from the window

pseudo code

```
def    findMaxInWindow ( arr, k) {
        // Initialize empty dequeue
1st wind-ow
        for i=0 → k-1;
            while ( ! q.isEmpty () && A[i] >= A [rear])
                q. dequeue rear();
            q. enqueue Rear (i) ;
        print ( A [q. front ()] ;

        for i = k ⟶ n-1;
            if ( ! q. isEmpty() && q. front () = i-k)
                q. dequeue Front(); // remove element
```

rest of the window | not part of window

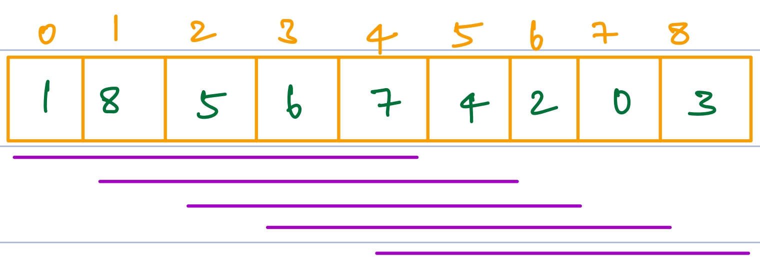
```
while (! q.isEmpty() && A[i] >= A[rear])
        q.dequeueRear();
q.enqueueRear(i);
// ans is in A[front]
```

Dry Run

arr =

0	1	2	3	4	5	6	7	8
1	8	5	6	7	4	2	0	3

k = 5



~~0~~ ~~1~~ ~~2~~ ~~3~~ 4 5 ~~6~~ ~~7~~ 8

Front rear

arr[1] , arr[1] , arr[4] , arr[4] , arr[4]