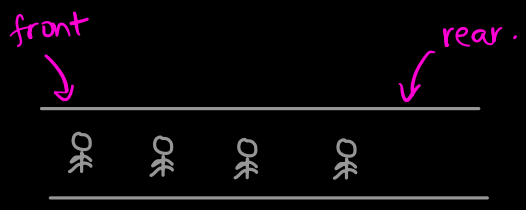
Today's content.

* Introduction to deques.
* Maximum element in every window of size 'k'.
* Generate k^{th} palindrome. (Perfect number)

Deques. : Double ended queues.

[insertion & deletion can be done on both the sides].



Operations possible for deque: Its implemented using doubly linked list.

(i) Push to front

(ii) Pop from front

(iii) Push to rear

(iv) Pop from rear

→ queue.

(v) Read front and rear.

(i) & (ii) => Stack

(iii) & (iv) => Stack.

Q1: Given ar[N] & k, find the max element in every window of size k.

Ex1: [10 1 9 3 7 6 5 11 8] , k=4
0 1 2 3 4 5 6 7 8

O/p : [10 9 9 7 11 11]

idea: For every subarray of length k, iterate and get max

TC: [N-K+1] * K , SC: O(1)

worst case: When $k = n/2$, $[N - N/2 + 1][N/2] \Rightarrow [N/2 + 1][N/2]$

$TC \simeq O(N^2)$

idea: sliding window

[10 | 1 9 3 7]
0 | 1 2 3 4

When the max ele is getting removed, we need to re-calculate max,

=> same as idea 1.

---

idea:

0	1	2	3	4	5	6	7	8	9	10	11	12
ar: [3	15	6	12	4	2	10	9	13	7	2	5	3]

, k = 4

container

~~3~~ ~~15~~ ~~6~~ ~~12~~ ~~4~~ ~~2~~ ~~10~~ ~~9~~ ~~13~~ 7 ~~2~~ 5 3

15 15 12 12 10 13 13 13 13 7

Operations: dequeue

push to rear
delete from rear
delete from front
read from & rear

Ex2:

[10 1 9 3 7 6 5 11 8]

0 1 2 3 4 5 6 7 8

container.

~~10~~ x ~~9~~ ~~3~~ ~~7~~ ~~6~~ ~~5~~ 11 8

10 9 9 7 11 11

What to do with equal elements?

(i)

0	1	2	3	4	5	
10	6	10	3	9	11	, K = 4

~~10~~ ~~6~~ ~~10~~ ~~3~~ 9

10

maintain equal elements also in the queue.

Steps:

(i) Try to insert first k elements into deque

delete from rear as long as `ar[i] > dequeue.rear()` ← while

push ar[i] to the rear

(ii) Try to insert elements from i = k to (n-1) indices.

delete from rear as long as `ar[i] > dequeue.rear()` ← while

```
if (ar[i-k] == dq.front())
    remove from front.
push ar[i] to the rear
```

```
void subarrayMax ( int ar[n], int k)
    Dequeue<int> dq;

    for (i=0; i<k ;i++)
        while (dq.size()>0 && ar[i] > dq.rear())
            dq.poprear()
        dq.push_rear(ar[i])

    print(dq.front());

    for (i=k; i<n ;i++)
        while (dq.size()>0 && ar[i] > dq.rear())
            dq.poprear()
        dq.push_rear(ar[i])
        if (ar[i-k] == dq.front())
            deq.popfront()
        print(dq.front());
```

Step 1.

Step 2.

Break.
8:29:00
8:39:00
8:40:00

TC: O(N) (∵ at the max 2n operations).
↳ n → push
n → pop.

SC: O(k)

i) Generate K^{th} even palindrome using the digits 1 4 2. [Perfect number].

K = 5,

2 digit palindrome : 11 22

4 digit palindrome : 1111 1221 (11 → 2112, 22 → 2222)

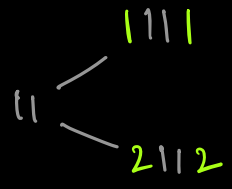
6 digit palindrome : 111111 112211 ------

Observation:

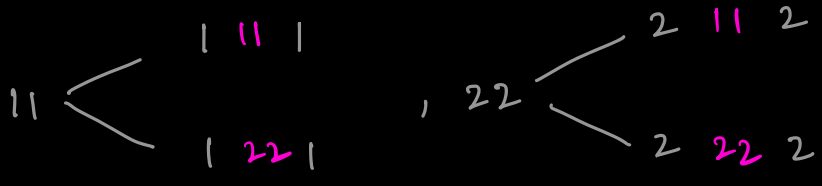
i) If you've a palindrome, you can add same character at the beg & end. It'll still be a palindrome.

base no's.

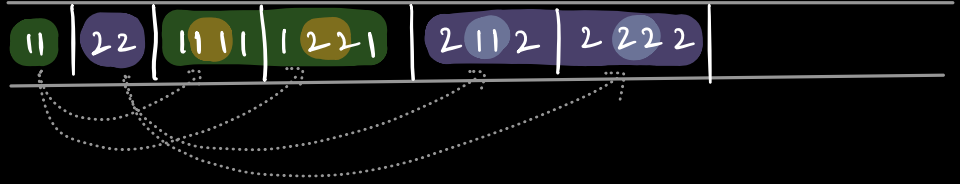
11 22



ii) If you've a palindrome, you can add some character in the middle, It'll still be a palindrome.

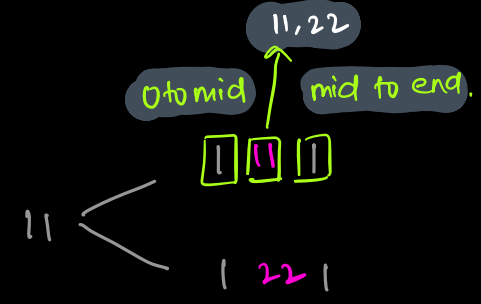


Use a queue.



The slower you plan, the faster you code.

Code:




```
String  Kth Palindrome (int k)
    Queue <String>  q;
    q.add("11"), q.add("22");
    for (i=1; i<k; i++)
        ele = q.pop();
        mid = (ele.length())/2;
        q.add( substring(ele, 0, mid) + "11" + substring(ele, mid, ele.length))
        q.add( substring(ele, 0, mid) + "22" + substring(ele, mid, ele.length))
    return  q.front()
```

Use StringBuilder in Java.

TC: O(K * max no. of ele in string)

SC: O(K)

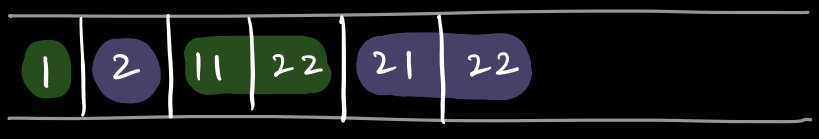
↑ small optimization.

2 digit palindrome : 11 22

4 digit palindrome : 1111 1221 2112 2222

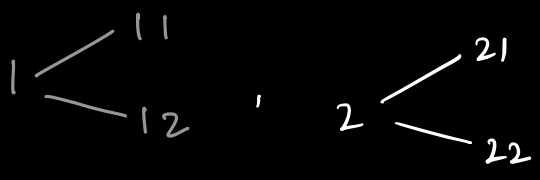
6 digit palindrome : 111111 112211 - - - - -

Generate only first half of palindrome : 1121 1



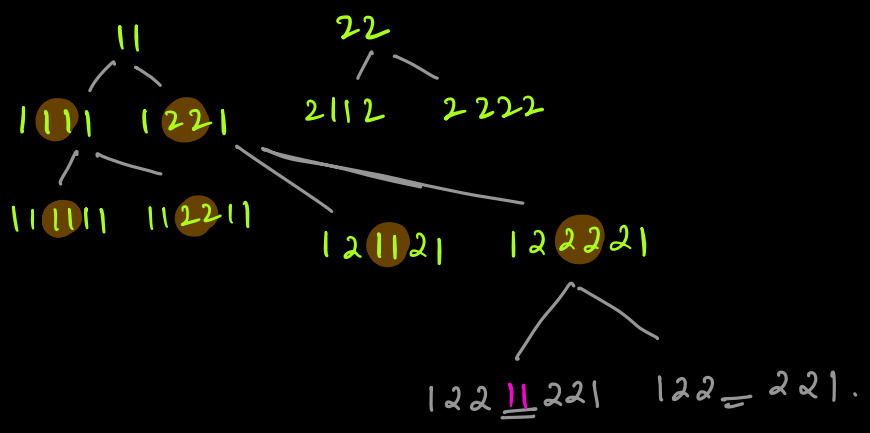
Base values.

[1] [2].



Code: TODO.

Doubts



1 22 11 22 1

TO-DOs.

(i) Share both backgrounds

(ii) Share python notes.