→ Sliding window maximum :-
(finding maximum element in subarray of size k) :-

We have a array of size 'n' and we have to find maximum element in each subarray of size 'k'.

Method 1:- For this we can use most simple route, we take each subarray and traverse that subarray to find maximum in that subarray.

but this method has complexity of n^2.

Code :-

```
int main()
{
    int arr[7]={3, 7, 8, 12, 2, 1, 0, 9};
    int k = 3
    int n=sizeof(arr)/sizeof(arr[0]);
```

```
for(int i=0, j=k-1; j<n; i++, j++)
{
    int maxe=INT_MIN;
    for(int m=i; m<=j; m++)
    {
        maxe=max(maxe, arr[m]);
    }
    cout<<maxe<<" ";
}
return 0;
}
```

method 2:-

It is more effective method. It has time complexity of O(n). It does not seem so, as every element of array is inserted and deleted once.

It is a bit tricky algorithm.

We first create a dequeue of size k (dequeue as it helps in insert/dele. from both end and provide random access)

On dequeue in place of elements we store their index.

It can store k elements of current window but it will store only useful ones b/w those 'k' elements.

An element is useful if it is in current window/subarray of size 'k' and is greater than all other element on the left side of it in the current window.

We process all elements of array one by one and main deque in such a way that greatest element of current window is at front of <dq> and smallest element of current window at at back of <dq>.

algorithum:-

- make a dequeue of size 'k'.
- run loop for k elements i=0 to i=k-1
  - remove ith element from back of <dq> if current element is greater than <dq> and <dq> is not empty. do this until <dq> is empty or current element is smaller than back element.

- After that insert index of current element at back.

- run another loop for rest of n-k elements from i=k to i=n-1

- print element at front of <dq> as it is greatest element in previous window.

- now remove the element from dq, which are of previous window. for this check if index stored at front of <dq> is less than equal to i-k. if yes remove it.

- After removing elements, which are not part of current window, add new elements (current elements are remove elements those elemets less than or equal to current elements.

- add current element at back.

. At last step print maximum element of last window by printing front element of dq.

Code:-

```
int main(){
    int arr[] = {7, 10, 12, 6, 20, 13, 5, 23};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;

    deque<int> dq(k);
    int i;
    for(i=0; i<k; i++)
    {
        while((!dq.empty()) && (arr[i] >=
                arr[dq.back()])
        {
            dq.pop_back();
        }
        dq.push_back(i);
    }
    for(; i<n; i++)
    {
        cout<<arr[dq.front()]<<" ";
```

```
while((!dq.empty()) && (dq.front()<=
                        i-k))
{
    dq.pop_front();
}

while((!dq.empty()) && (arr[i]>=
                        arr[dq.back()]))
{
    dq.pop_back();
}
dq.push_back(i);
}
cout<<arr[dq.front()]<<" ";
return 0;
}
```

Method 3 - (we can use set also)

→ Generate numbers using given digits

We have to print 'n' numbers in increasing order such that numbers contains only '5' and '6'.

ex-

n = 10

output = 5 6 55 56 65 66 555 556 565 566

A ameture approach is run infinite loop and print 'n' numbers with ~~digits only~~ 5 and 6 only digit.

But it will have very high T.C. but we will you another better approach with O(n) T.C.

For this better approach we will use queue.

We store numbers as string in que.

We first store numbers / push them which are being used to generate new

numbers . in this case 5 and 6.

then we print front element of queue and ● before poping it. we add new elements as

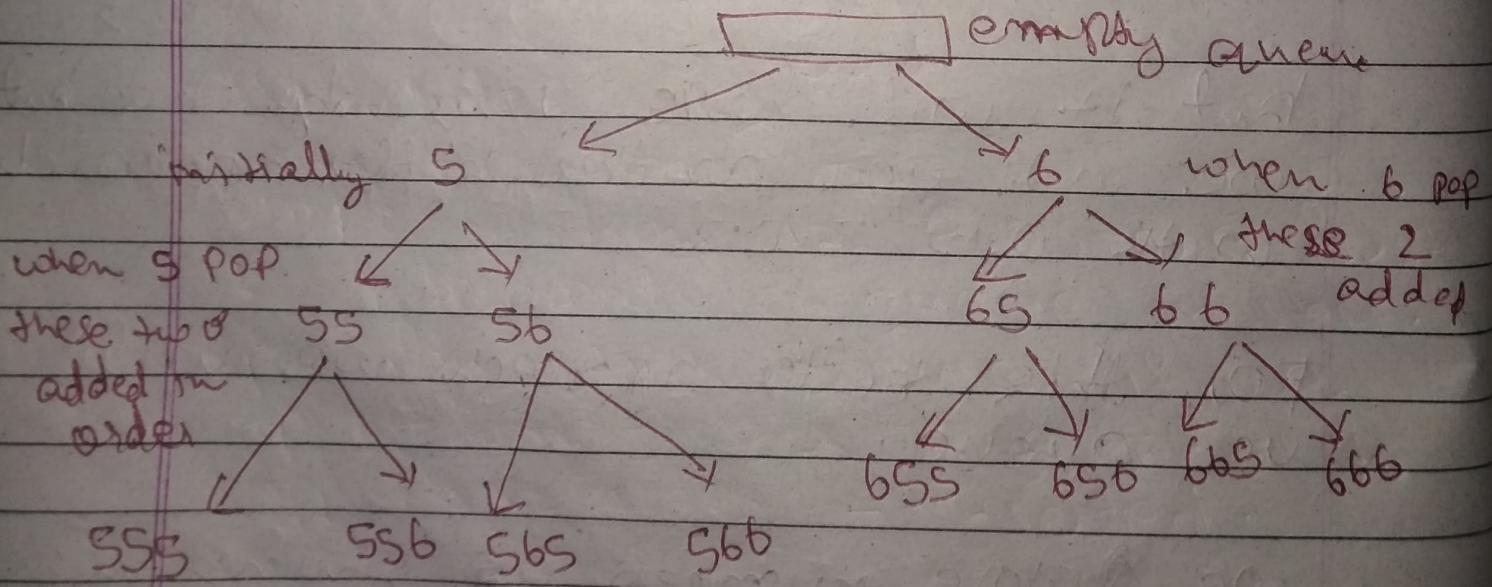
curr + 'digit 1'

curr + 'digit 2'

curr + 'digit as many there'

here digit 1 < digit 2 < - - - digit n

by doing so what we are doing is we are making a tree type struct



Code:-

```
#include<iostream>
#include<queue>
using namespace std;

int main()
{
    queue<string> q;
    q.push("5");
    q.push("6");
    int n=10;
    for(int i=0; i<n; i++)
    {
        string curr = q.front();
        cout<<curr<<" ";
        q.pop();
        q.push(curr+"5");
        q.push(curr+"6");
    }
    return 0;
}
```

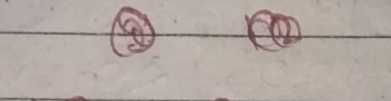
→ Reverse first 'k' items of a queue:-

we have a queue of size 'n' and a integer 'k' and we have to reverse first 'k' elements of queue.

ex-

```
queue<int> q={7,2,12,9,1,37};
int k=3;
```

output-



12 2 7 9 1 37

These 3 are reversed.

Simple Solution is store first 'k' elements of queue in stack. Specially stack because, if we pop() from stack, we get it in opposite order it inserted.

now, after storing it in stack, we push them one by one in queue back.

after that we get first n-k elements of queue one by one. poped out and again push them back, then we get our result.

ex-

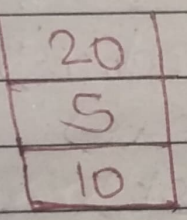
queue <int> q = { 10, 5, 20, 30, 45 };

k = 3

output - 20 5 10 30 45

→

store them is stack of size k=3



now q = {30, 45}

after pushing element back in queue from stack

q = {30, 45, 20, 5, 10}

now, first 2 element (n-k = 5-3 = 2) we have to remove and push them in back of queue.

So,

in first iteration

q = {45, 20, 5, 10, 30}

in second time

q = {20, 5, 10, 30, 45}

which is required result.

Code :-

```
#include<iostream>
#include<queue>
#include<stack>
using namespace std;

int main()
{
    queue<int> q={10,7,2,18,30,26};
    int k= 4;
    stack<int> s;
    for(int i=0;i<k;i++)
    {   int x = q.front();
        s.push(x);
```

```
        q.POP();
    }
for(int i=0; i<k; i++)
    {
        int x = S.top();
        q.Push(x);
        S.POP();
    }

for(int i=0; i<q.size()-k; i++)
    {
        q.Push(q.front());
        q.POP();
    }
return 0;
}
```

NOTE! - We have not handled cases like if queue is empty or k > q.size() or k is zero or less. For this just add one if condition at top.

→ Balance paranthesis problem

In this problem we have given a String of paranthesis '(', '[', '{', '}', ']' and ')'.

ex. -

String s = "[( )]";

now, we have to check if string is balanced or not.

String is balanced if

- String contain equal no of open and close bracket.

ex. -

"[(( ))]" → balanced

"[[({ ( ) }]]" → unbalanced

- Bracket that open first will must closed later.

ex. -

"[(])" → unbalanced, as '[' is open before '(' so it must closed after inner bracket ')'.

"[] (()) " -> balanced

approach:-

We will solve this problem using stack, as in this problem LIFO rule is required.

- we first make a stack of char.

input the char if it is one of opening brackets '(', '[' or '{'

- else, it is not, then check if the closing brackets is equal to top() bracket of stack. (as it will ~~control~~ keep control of the fact that last inner bracket opened will be closed first also). also pop() top bracket after that.

if closing bracket is not same as top() bracket of stack, return false.

after loop is over then check, if stack is empty or not, if it is empty return true, else false.

it will keep control on the fact that no of opening and closing bracket should be same.

Code :-

```
#include<iostream>
#include<stack>
#include<string>

bool equal(char a, char b)
{
    return ((a=='(' && b==')') || (a=='{' && b=='}')
            || (a=='[' && b==']'));
}

bool check(string c)
{
    stack<char> s;
    for(int i=0; i<c.length(); i++)
    {
        if(c[i]=='(' || c[i]=='{' || c[i]=='[')
            s.push(c[i]);
        else{
            if(equal(c[i], s.top())==true)
            {
                s.pop();
                continue;
            }
```

```
        else
            return false;
        }
    }
    return (s.empty() == true);
}

int main()
{
    string c = "[()]";
    if(chek(c) == true)
        cout<<"string is balanced";
    else
        cout<<"string is not balanced";
    return 0;
}
```

Here to check if c[i] and s.top() is same or not we have used equal function not '==' equal to operator as these operator not work in this case.

→ Stock span problem :-

In this problem we have given value of a stock as integer element in array.

Every element represent values of stock at that day. We have to find span of that stock.

Span of a stock means number of continuous days before it, where value of stock is less than or equal to todays value, also count todays values. So, minimum span of stock will be 1.

ex-

Stock [ ] = {70, 80, 60, 30, 45, 90, 60, 100};

- Output :- 1, 2, 1, 1, 2, 6, 1, 8

now, as you can see to calculate span of given stock say stock[i], what we are doing is we are searching left of that element and find a element having more value than given element

if find it then we calculate difference b/w current element index and max element index, that will be span of that element.

and if no greater element is left is find in left of given element that means span is equal to (i+1), where 'i' is index of given element.

now, nieve approach can be will be to traverse the array and for each element we again traverse whole element in it's left.
it will cost O(n²) T.C.

another, effective approach is to store index of array element in another container and maintain it in such way that it's top element gives index of first max element in left than current element.

FOR this we will use stack container, as it keep track recent element.

approach:-

We store '0' as first element in stack corresponding to first element in array.

now, ~~we check, if~~ we traverse the array from ~~i~~=1 to ~~i~~=n-1. ~~insert it in stack only if less than top element~~

when we at index i of array, we first remove all smaller element then it, from stack, then

calculate Spam according to fact that stack is empty (no greater element than given element) or we get max element and print Span.

Then push 'i' to stack.

Code :-

```
Void span(int stack[])
{
    stack <int> S;
    S.push(0);
    cout<< " 1 ";
    for(int i=0; i<n ; i++)
    {
        while((!S.empty()) && arr[i]>=
                arr[S.top()])
        {
            S.POP();
        }
    int spam = (S.empty())?(i+1):(i-S.top());
    cout<< spam << " ";
    S.push(i);
    }
}
```