1. Stack using Queue
    a. https://leetcode.com/problems/implement-stack-using-queues/solution/
    b. Push = O(1) and pop = O(n)
        i. Insertion order : 1 2 3 4
        Stack : 1 2 3 4 - top
        Queue :

Push	Pop
1	F 1 2 3 4 R
1 2	F 4 1 2 3 R Delete and insert 3 times
1 2 3	F 1 2 3 R Delete one time , returns 4
F 1 2 3 4 R	
	Pop
	F 1 2 3 R
	F 3 1 2 R Delete and insert 2 times
	F 1 2 R Delete one time , returns 3

        ii. Pop : Delete and insert Q.size() - 1 times
        Delete one more time and return it
    c. Push = O(n) and pop = O(1)
        i. Insertion order : 1 2 3 4
        Stack : 1 2 3 4 - top
        Queue :

Push		Pop	
1	1	4 3 2 1	
1 2	2 1	3 2 1	=> 4
2 1 3	3 2 1	2 1	=> 3
3 2 1 4	4 3 2 1	1	=> 2

        ii. Push : Delete and insert Q.size() - 1 times
        Insert element
2. Queue using Stack
    a. https://leetcode.com/problems/implement-queue-using-stacks/

3. Largest Histogram in Ractange :
    a. https://leetcode.com/problems/largest-rectangle-in-histogram/discuss/28902/5ms-O(n)-Java-solution-explained-(beats-96)
    b. Next smaller left side using stack

```
vector<int> ans(A.size());
   stack<int> st;
   for(int i=0;i<A.size();i++){
       while(!st.empty() && st.top() >= A[i] ) st.pop();
       if(st.empty()) ans[i] = -1;
       else ans[i] = st.top();
       st.push(A[i]);
   }
```

    c. Using dp

```
int leftsmaller[n], rightsmaller[n];
   for(int i=0;i<n;i++){
       int p = i - 1;
       while(p >= 0 && A[p] >= A[i])
           p = leftsmaller[p];
       leftsmaller[i] = p;
   }
```

```
for(int i=n-1;i>=0;i--){
      int p = i + 1;
      while(p <= n-1 && A[p] >= A[i])
          p = rightsmaller[p];
      rightsmaller[i] = p;
  }
```

4. Postfix Evaluation
    a. https://leetcode.com/problems/evaluate-reverse-polish-notation/submissions/
    b. Visit each token of string
    If token is operator
        i. Pop two element evaluate it and push it back
    Else
        ii. Push token

5. Minimum number of bracket reversals needed to make an expression balanced
    a. https://www.geeksforgeeks.org/minimum-number-of-bracket-reversals-needed-to-make-an-expression-balanced/
    b. Remove all pairs {} after that -- >}}}...{{{
    c. Ans = m/2 + n/2 , m - no of } , n - no of {

6. Length of the longest valid substring
    a. `geeksforgeeks.org/length-of-the-longest-valid-substring/`
    b.

```
for (int i=0; i<n; i++)
    {
        // If opening bracket, push index of it
        if (str[i] == '(')
          stk.push(i);
        else // If closing bracket, i.e.,str[i] = ')'
        {
            // Pop the previous opening bracket's index
            stk.pop();

            // Check if this length formed with base of
            // current valid substring is more than max
            // so far
            if (!stk.empty())
                result = max(result, i - stk.top());

            // If stack is empty. push current index as
            // base for next valid substring (if any)
            else stk.push(i);
        }
    }
```

    c.

7. Get min in O(1) using stack
    a. https://www.interviewbit.com/problems/min-stack/
    b. The idea is to store the next min below that element in stack so that if we remove any element min value can be updated by next element.
    So when we do push operation and if element x is smaller than current element update min element. Push min element and push new element x

8. Sliding window maximum
    a. https://leetcode.com/problems/sliding-window-maximum/

    b. Use next greater array and two pointer concept

9. Rain Water Trapped
    a. https://leetcode.com/problems/trapping-rain-water/
    b. Each height[i] will contribute min(leftMax[i-1],rightMax[i+1]) - height[i] amount of water.