1. Stack using Queue
   1. <https://leetcode.com/problems/implement-stack-using-queues/solution/>
   2. Push = O(1) and pop = O(n)
      1. 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

* + 1. Pop : Delete and insert Q.size() - 1 times

Delete one more time and return it

* 1. Push = O(n) and pop = O(1)
     1. 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

* + 1. Push : Delete and insert Q.size() - 1 times

Insert element

1. Queue using Stack
   1. <https://leetcode.com/problems/implement-queue-using-stacks/>
2. Largest Histogram in Ractange :
   1. <https://leetcode.com/problems/largest-rectangle-in-histogram/discuss/28902/5ms-O(n)-Java-solution-explained-(beats-96)>
   2. 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]);  } |
| --- |

* 1. 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;  } |
| --- | --- |

1. Postfix Evaluation
   1. <https://leetcode.com/problems/evaluate-reverse-polish-notation/submissions/>
   2. Visit each token of string

If token is operator

* + 1. Pop two element evaluate it and push it back

Else

* + 1. Push token

1. Minimum number of bracket reversals needed to make an expression balanced
   1. <https://www.geeksforgeeks.org/minimum-number-of-bracket-reversals-needed-to-make-an-expression-balanced/>
   2. Remove all pairs {} after that -- >}}}...{{{
   3. Ans = m/2 + n/2 , m - no of } , n - no of {
2. Length of the longest valid substring
   1. geeksforgeeks.org/length-of-the-longest-valid-substring/

| 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);  }  } |
| --- |



1. Get min in O(1) using stack
   1. <https://www.interviewbit.com/problems/min-stack/>
   2. The idea is to store the next min below that element in the stack so that if we remove any element min value can be updated by the 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

1. Sliding window maximum
   1. <https://leetcode.com/problems/sliding-window-maximum/>
   2. Use next greater array and two pointer concept
2. Rain Water Trapped
   1. <https://leetcode.com/problems/trapping-rain-water/>
   2. Each height[i] will contribute min(leftMax[i-1],rightMax[i+1]) - height[i] amount of water.
3. Parentheses
   1. Min parenthesis add to make string valid
      1. <https://leetcode.com/problems/minimum-add-to-make-parentheses-valid/>
      2. Keep a count inc and dec
      3. If count == -1

Count = 0

ans ++

Ret ans + count

* 1. <https://www.interviewbit.com/problems/redundant-braces/>
  2. <https://leetcode.com/problems/minimum-remove-to-make-valid-parentheses/>

If ‘(‘ add to stack

If ‘)’ {

If stack is empty

Make s[i] = \*

Else

pop()

}

All index in stack make s[i] = \*

* 1. <https://leetcode.com/problems/minimum-insertions-to-balance-a-parentheses-string/>

Make cases for ( and ) parenthesis

Count = 0

If stack is empty

)( → +2

) → +2

)) → +1

Else

)( → +1

) → +1

)) → 0

Return count + st.size()/2

* 1. https://leetcode.com/problems/valid-parenthesis-string/

St1 -> ‘(‘ and st2 -> ‘\*’

If s[i] == ‘)’ check for st1 and then st2

At the end make pair from st1 and st2 but the index should be less . ‘\*\*((‘ is invalid so

* 1. <https://leetcode.com/problems/minimum-remove-to-make-valid-parentheses/>