20. Valid Parentheses

Valid Paranthesis

```
class Solution:
    def isValid(self, s: str) -> bool:
        while '()' in s or '[]' in s or '{}' in s:
            s = s.replace('()', '').replace('[]', '').replace('{}', '')
        return s == ''
```

155. Min Stack

Min Stack

```
class MinStack:

    def __init__(self):
        """
        initialize your data structure here.
        """
        self.arr=[]


    def push(self, x: int) -> None:
        self.arr.append(x)


    def pop(self) -> None:
        if len(self.arr)>0:
            return self.arr.pop()


    def top(self) -> int:
        if len(self.arr)>0:
            return self.arr[-1]

    def getMin(self) -> int:
        if len(self.arr) > 0:
            return min(self.arr)
        return None



# Your MinStack object will be instantiated and called as such:
# obj = MinStack()
# obj.push(x)
# obj.pop()
# param_3 = obj.top()
# param_4 = obj.getMin()
```

225. Implement Stack using Queues

Implement Stack using Queues

```
class MyStack:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.arr=[]


    def push(self, x: int) -> None:
        """
        Push element x onto stack.
        """
        self.arr.append(x)


    def pop(self) -> int:
        """
        Removes the element on top of the stack and returns that element.
        """
        if len(self.arr)>0:
            return self.arr.pop()


    def top(self) -> int:
        """
        Get the top element.
        """
        if len(self.arr)>0:
            return self.arr[-1]


    def empty(self) -> bool:
        """
        Returns whether the stack is empty.
        """
        if len(self.arr)>0:
            return False

        return True



# Your MyStack object will be instantiated and called as such:
# obj = MyStack()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.top()
# param_4 = obj.empty()
```

496. Next Greater Element I

Next Greater Element 1

```
# a stack with monotonic decreasing
       monotonic_stack = []

       # dictionary:
       # key: number
       # value: next greater number of key
       dict_of_greater_number = {}

       # ---------------------------------------------

       # launch linear scan to build dict_of_greater_number
       for cur_number in nums2:

           # maintain a monotonic decreasing stack
           while monotonic_stack and cur_number > monotonic_stack[-1]:

               pop_out_number = monotonic_stack.pop()

               # next greater number of pop_out_number is cur_number
               dict_of_greater_number[pop_out_number] = cur_number

           monotonic_stack.append(cur_number)
       # ---------------------------------------------

       # solution output
       next_greater_element = []

       # get next greater element by dictionary
       for x in nums1:

           if x in dict_of_greater_number:
               next_greater_element.append( dict_of_greater_number[x] )

           else:
               next_greater_element.append(-1)

       return next_greater_element
```

682. Baseball Game

Baseball Game

```
class Solution:
    def calPoints(self, ops: List[str]) -> int:
        stack = []
        for c in ops:

            if c=="C":
                stack.pop()

            elif c=="D":
                stack.append(stack[-1]*2)

            elif c=="+":
                stack.append(stack[-1]+stack[-2])

            else:
                stack.append(int(c))
        return sum(stack)
```

844. Backspace String Compare

Backspace String Compare

```
class Solution:
    def backspaceCompare(self, S: str, T: str) -> bool:
        def reduce(X):
            x = ''
            for i, val in enumerate(X):
                if val != '#':
                    x += val
                else: # val == '#'
                    x = x[:-1]
            return x

        return reduce(S) == reduce(T)
```

1021. Remove Outermost Parentheses

Remove Outermost Paranthesis

```
class Solution:
    def removeOuterParentheses(self, S: str) -> str:
        pop, result = 0, []
        for x in S:
            if x==')':
                pop -= 1
            if pop>0:
                result.append(x)
            if x=='(':
                pop += 1
        return ''.join(result)
```

1047. Remove All Adjacent Duplicates In String

Remove all Adjacent Duplicates in String

```
class Solution:
    def removeDuplicates(self, S: str) -> str:
        i = 0
        while(i < len(S) - 1):
            if S[i] == S[i+1]:
                S = S[:i] + S[i+2:] # remove duplicated items
                if i != 0:
                    i -= 1 # reset idx if it is not at the beginning.
            else:
                i += 1 # keep searching
        return S
```

1441. Build an Array With Stack Operations

Build an Array with Stack Operation

```
class Solution:
    def buildArray(self, target: List[int], n: int) -> List[str]:
        ans = []
        for i in range(1,target[-1]+1):
            if i in target:
                ans.append("Push")
            else:
                ans.append("Push")
                ans.append("Pop")
        return ans
```