STACK

No.	Problem Statement	Solution	Time complexity	Space complexity
1	**Valid Paranthesis**			
	Given a string s containing just the characters '(', ')', '{', '}', '[', ']', determine if the input string is valid. An input string is valid if: 1) Open brackets must be closed by the same type of brackets. 2) Open brackets must be closed in the correct order. 3) Every close bracket has a corresponding open bracket of the same type.	- Use **stack** to store opening brackets and match them with corresponding closing brackets	O(N)	O(1)
2	**Min Stack**			

	Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.	- Use 2 stacks :
		`stack<int> s`
		`stack<pair<int, int>> s_min`
		– [value, # occurences]
		– `s_min.top().first` will always have min value
		- Use stack 's' as a normal stack and push element into it
		- **Only** push into second stack if the new value to be inserted is **smaller** than current min
			O(1)	O(N)

3	**Evaluate Reverse Polish Notation**			
	You are given an array of strings tokens that represents an arithmetic expression in a reverse polish notation. Evaluate the expression. Return an integer that represents the value of the expression. Ex. tokens = ["2","1","+","3","*"] Output: 9 Explanation: ((2 + 1) * 3) = 9	- Use a Stack - Traverse the tokens vector - If number push into the stack - If operator apply to top 2 numbers, Push the result back to stack' - Note: keep in mind **n2**=`s.top(); s.pop();` **n1**=`s.top(); s.pop()`	O(N)	O(N)
4	**Generate Paranthesis**			

	Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.
	Ex. Ex. n = 3 -> ["((()))","(()())","(())()","()(())","()()()"], n = 1 -> ["()"]
		- **Backtracking**
		- Only if **#open > #close** then add ')'
		`if(open == n && close == n)`
		`ans.push_back(s)`
		`return`
		`if(open < n)` `helper(n, open+1, close, s+"(", ans)`
		`if(open > close)` `helper(n, open, close+1, s+")", ans)`
			O(2^N)	O(N)

5	**Daily Temperatures**			

	Given array of temperatures, return an array with number of days until it gets warmer.
	Ex. temperature = [73,74,75,71,69,72,76,73] -> [1,1,4,2,1,1,0,0]
		- Use a **stack** --> To keep track of the **indices** of temperatures that we **haven't** found a warmer day for yet
		`for(int i=0; i<n; i++)`
		`while(!s.empty() && temperatures[i] > temperatures[s.top()]){`
		`prev_day = s.top();`
		`ans[prev_day] = i - prev_day;`
		`s.pop();`
		`s.push(i);`
			O(N)	O(N)

6	**Car Fleet**			

	There are 'n' cars going to the same destination. The destination is 'target' miles away.
	position[i] = position of the ith car; speed[i] = speed of the ith car.
	A car can never pass another car ahead of it, but it can catch up to it and drive at the same speed. A car fleet is some non-empty set of cars driving at the same position and same speed. Note that a single car is also a car fleet. If a car catches up to a car fleet right at the destination point, it will still be considered as one car fleet. Return the number of car fleets that will arrive at the destination.
	Ex. target = 12, position = [10,8,0,5,3], speed = [2,4,1,1,3] --> output = 3
		- Keep a `vector<pair<int, double>> v` --> **[position, time to reach target from that position]**
		- Sort the vector in descending order
		- If another car needs less or equal time than cur, it can catch up this car fleet.
		- But If another car needs more time, it will be the new slowest car, and becomes the new lead of a car fleet.
		`for(auto p: v)`
		`if(p.second>max_time)`
		`ans++;`
		`max_time=p.second;`
			O(N)	O(N)

STACK				
No.	**Problem Statement**	**Solution**	**Time complexity**	**Space complexity**
7	**Largest Rectangle In Histogram**			

	Given array of heights, return area of largest rectangle
	Ex. heights = [2,1,5,6,2,3] -> 10 (5 x 2 at index 2 and 3)
	https://assets.leetcode.com/uploads/2021/01/04/histogram.jpg
		(This can't be solved using two pointers and is different from the problem number 4 of 'two pointers' because there when consider two lines to form a rectangle, we can ignore the lines in between which is not the case for this probelm.)
		- When a **lower** height is encountered, calculate the maximum area possible for the bars **taller** than the current bar
		- Stack will always have the **indices** of histograms in increasing order of their heights (i.e 4,3,2,1)
		`for(int i=0; i<n; i++)`
		`  start = i;`
		`  while(!s.empty() && heights[i] < s.top().second)`
		`  height = s.top().second;`
		`  width = i - s.top().first;`
		`  start = s.top().first;`
		`  ans=max(ans, height * width);`
		`  s.pop();`
		`  s.push({start, heights[i]});`
			O(N)	O(N)