Stack

LIFO: Last in First out.

Examples + terms:

- Stack of plates, Stack trace
- Stack overflow
- Stack undeflow

Operations:

- push(): Add Data at the end/top
- pop(): Remove data from the end/top
- peek(): Look at the element at the top without removing it
- empty()

In [ ]:

In [ ]:
```cpp
class Stack {
    int arr[5];
    int pos;

    public:
        Stack() {
           this->pos = -1;
        }

        void push(int value) {
            if (pos >= 5) {
                // exception
            }

            this->pos++;
            this->arr[this->pos] = value;
        }


        int pop() {
            if (pos == -1) {
                // exception
            }

            int value = this->arr[this->pos];
            this->pos--;

            return value;
        }

        bool empty() {
            return this->pos == -1;
        }

        int peek() {
            if (this->pos == -1) {
                // exception
            }

            return this->arr[this->pos];
        }

};
```

C++
Stack:
- push()
- pop()
- top()
- empty()

Stack using Vector:
- push: push_back()
- pop: pop_back()
- peek: back()

```
- empty: empty()
```

In [ ]:

Python

Stack using []:

- push: append()
- pop: pop()
- peek/top: a[-1]
- empty: len(a) == 0

In [ ]:

Java
Stack:

- push()
- pop()
- peek()
- empty()

In [ ]:
```
template <type T>
class Stack {

    public:
        void push_back(); # O(1)
        void pop(); # O(1)
        T peek(); # O(1)
        bool empty(); # O(1)
}
```

Queue

- FIFO: First in First out

Queue

- Enqueue: Insert/ Add /Push: O(1)
- Dequeue: Remove/ Delete/ Pop: O(1)
- Peek(): O(1)
- Empty(): O(1)

Dequeue

- Circular Queue
- Double Ended Queue

In [ ]:

Question
https://leetcode.com/problems/valid-parentheses/ (https://leetcode.com/problems/valid-parentheses/)

In [ ]:
```
()[]{}   y
{(){} (){} ([]){[][]{}}} y
{(){} (){} (]){[][]{}}}[n
()
```

In [ ]:
```
S = stack

for (i =0 ; i < brackets)
```

```
TC: O(N)
SC: O(N)
```C++
class Solution {
public:
```
```

```
    bool isValid(string s) {

        std::stack<char> brackets;

        for (auto c: s) {
            if (isOpen(c)) {
                brackets.push(c);
            } else {
                if (brackets.empty()) {
                    return false;
                }

                char closingBracket = brackets.pop();
                if(! isMatching(closingBracket, c))
                    return false;

            }
        }

        return brackets.empty();

    }

    bool isOpen(char bracket) {
        return bracket == '('  || bracket == '[' || bracket == '{';
    }

    bool isMatching(char open, char close) {
        switch(open) {
            case '(': return close  == ')';

            case '[': return close  == ']';

            case '{': return close  == '}';
        }

        return false;
    }
};
```

In [1]:
```
class Solution:
    def isValid(self, s: str) -> bool:
        res = []
        for el in s:
            if el in '({[':
                res.append(el)
            else:
                if len(res) == 0:
                    return False
                top = res.pop()
                if el == ')' and top != '(':
                    return False
                if el == '}' and top != '{':
                    return False
                if el == ']' and top != '[':
                    return False
        return len(res) == 0
```

In [ ]:
```
{[]}
{[ }]
```

In [ ]:
```

```

Question
Given an array of integers, return a result array where for each element, it is replaced with the next greater element on the right.

1 3 4 3 5 2 3 4 5 5 -1 -1

5 4 3 2 1 -1 -1 -1 -1 -1

Brute Force

- TC: O(N^2)
- SC: O(1)

Stack R->L

i = len - 1 stack = empty

while(i >= 0) { curr = arr[i]

```
    while (stack has a value which is < than curr) {
        stack.pop()
    }

    if (stack.empty()) {
        res[i] = -1;
    } else {
        res[i] = stack.top();
    }

    stack.push(curr);

}
```

Two approach:

- Value based (R->L)
- Index based (L->R)

```
vector<int> nextGreaterElement(vector<int> data) {
    stack<int> s;

    for(int i = data.size() - 1; i >= 0; i--) {
        int curr  = data[i];
        while(!s.empty() && s.top() < curr) s.pop();

        if (s.empty()) {
             data[i] = -1;
        } else {
             data[i] = s.top();
        }
        s.push(curr);
    }

    return data;
}
```

In [ ]:

In [ ]:

Question
https://leetcode.com/problems/next-greater-element-i/ (https://leetcode.com/problems/next-greater-element-i/)

```
class Solution {
public:
    vector<int> nextGreaterElement(vector<int>& nums1, vector<int>& nums2) {

        unordered_map<int, int> answers;
        vector<int> result;
        // run the next greater element algo and store results in hashmap
        // O(nums2.length)

        // for each element in nums1 lookup the answer in hashmap
        for (auto num: nums1) { // O(nums1.length)
            result.append(answers[num]); // O(1)
        }
    }
};
```

In [ ]:

In [ ]:

Question
https://leetcode.com/problems/minimum-add-to-make-parentheses-valid/ (https://leetcode.com/problems/minimum-add-to-make-parentheses-valid/)

In [ ]:
```python
class Solution:
    def minAddToMakeValid(self, s: str) -> int:
        ans = 0
        open = 0

        for c in s:
            if c == '(':
                open +=1
            else:
                if open == 0:
                    ans += 1
                else:
                    open -=1

        return ans + open
```

In [ ]:
```cpp
class Solution {
public:
    int minAddToMakeValid(string s) {

        std::stack<char> brackets;
        int ans = 0;
        for (auto c: s) {
            if (c == '(') {
                brackets.push(c);
            } else {
                if (brackets.empty()) {
                    ans += 1;
                } else {
                    brackets.pop();
                }
            }
        }

        return brackets.size() + ans;
    }
};
```

Question
https://leetcode.com/problems/next-greater-element-ii/ (https://leetcode.com/problems/next-greater-element-ii/)

In [ ]:

In [ ]:

In [ ]:

DIY:

- Implement a queue using a array.

In [ ]:

In [ ]: