Stack Questions

Basic Level:
Implement a Stack using Two Queues: Create a stack data structure using two queues.
Implement a Queue using Two Stacks: Design a queue using two stacks.
Sort a Stack: Sort the elements of a stack in ascending order using recursion.
Evaluate Postfix Expression: Evaluate a given postfix expression using a stack.
Check Balanced Parentheses with Multiple Types: Extend the parentheses validation problem to include angle brackets <> and test with various strings.
Check for Redundant Brackets: Detect redundant or unnecessary brackets in a mathematical expression.
Infix to Postfix Conversion: Convert an infix expression (with operators +, -, *, /) into its postfix equivalent.
Next Smaller Element: Find the next smaller element for each element in an array using a stack.
Min Stack: Design a stack that supports push, pop, and retrieving the minimum element in constant time.
Find Largest Rectangle in Binary Matrix: Find the largest rectangle of 1's in a binary matrix.

explain problem with example in simple way with proper input and output and give java optimized simple code with time complexity

Memory updated

Let's go over each of the problems one by one with examples and Java code.

1. Stack using Two Queues

Problem: Implement a stack using two queues. The operations `push`, `pop`, and `top` should behave as in a stack (LIFO).

Example:

- Input:
  - `push(1)`, `push(2)`, `top()`, `pop()`, `top()`
- Output:
  - `top()` -> 2, `pop()` -> 2, `top()` -> 1

Java Code:

```java
import java.util.LinkedList;
import java.util.Queue;

class StackUsingTwoQueues {
    Queue<Integer> q1 = new LinkedList<>();
    Queue<Integer> q2 = new LinkedList<>();

    public void push(int x) {
        q2.add(x);
        while (!q1.isEmpty()) {
            q2.add(q1.remove());
        }
        Queue<Integer> temp = q1;
        q1 = q2;
        q2 = temp;
    }

    public int pop() {
        if (q1.isEmpty()) return -1;
        return q1.remove();
    }

    public int top() {
        if (q1.isEmpty()) return -1;
        return q1.peek();
    }

    public boolean isEmpty() {
        return q1.isEmpty();
    }
}
```

- **Time Complexity**: `push()` -> O(n), `pop()` and `top()` -> O(1).

---

2. Queue using Two Stacks

Problem: Implement a queue using two stacks with operations `enqueue`, `dequeue`.
Example:

- Input:
  - `enqueue(1)`, `enqueue(2)`, `dequeue()`, `dequeue()`
- Output:
  - `dequeue()` -> 1, `dequeue()` -> 2

Java Code:

```java
    import java.util.Stack;

    class QueueUsingTwoStacks {
        Stack<Integer> s1 = new Stack<>();
        Stack<Integer> s2 = new Stack<>();

        public void enqueue(int x) {
            s1.push(x);
        }

        public int dequeue() {
            if (s2.isEmpty()) {
                if (s1.isEmpty()) return -1;
                while (!s1.isEmpty()) {
                    s2.push(s1.pop());
                }
            }
            return s2.pop();
        }
    }
```

- **Time Complexity**: Amortized O(1) for both `enqueue` and `dequeue`.

---

3. Sort a Stack using Recursion

Problem: Sort the elements of a stack in ascending order using recursion.
Example:

- Input: `push(3)`, `push(1)`, `push(2)`
- Output: Stack becomes `[1, 2, 3]`

Java Code:

java

```
import java.util.Stack;

class SortStack {
    public void sortStack(Stack<Integer> s) {
        if (!s.isEmpty()) {
            int x = s.pop();
            sortStack(s);
            insertSorted(s, x);
        }
    }

    private void insertSorted(Stack<Integer> s, int x) {
        if (s.isEmpty() || x > s.peek()) {
            s.push(x);
        } else {
            int temp = s.pop();
            insertSorted(s, x);
            s.push(temp);
        }
    }
}
```

- **Time Complexity**: O(n^2) where n is the number of elements in the stack.

---

4. Evaluate Postfix Expression

Problem: Evaluate a postfix expression.
Example:

- Input: `5 1 2 + 4 * + 3 -`
- Output: 14

Java Code:

java

```
import java.util.Stack;

class PostfixEvaluation {
    public int evaluatePostfix(String expression) {
        Stack<Integer> stack = new Stack<>();
        for (char ch : expression.toCharArray()) {
            if (Character.isDigit(ch)) {
                stack.push(ch - '0');
            } else {
                int val1 = stack.pop();
                int val2 = stack.pop();
                switch (ch) {
                    case '+': stack.push(val2 + val1); break;
                    case '-': stack.push(val2 - val1); break;
                    case '*': stack.push(val2 * val1); break;
                    case '/': stack.push(val2 / val1); break;
                }
            }
        }
        return stack.pop();
    }
}
```

- **Time Complexity**: O(n), where n is the length of the expression.

---

5. Check Balanced Parentheses with Multiple Types

Problem: Check if the given expression has balanced parentheses `{}`, `[]`, `()`, and `<>`.
Example:

- Input: `{[()<>]}`
- Output: `true`

Java Code:

```java
import java.util.Stack;

class BalancedParentheses {
    public boolean isBalanced(String expression) {
        Stack<Character> stack = new Stack<>();
        for (char ch : expression.toCharArray()) {
            if (ch == '(' || ch == '{' || ch == '[' || ch == '<') {
                stack.push(ch);
            } else {
                if (stack.isEmpty()) return false;
                char open = stack.pop();
                if ((ch == ')' && open != '(') || (ch == '}' && open != '{') ||
                    (ch == ']' && open != '[') || (ch == '>' && open != '<')) {
                    return false;
                }
            }
        }
        return stack.isEmpty();
    }
}
```

- **Time Complexity**: O(n).

---

6. Check for Redundant Brackets

Problem: Detect redundant or unnecessary brackets in a mathematical expression.
Example:

- Input: `((a+b))`
- Output: `true` (redundant brackets)

Java Code:

```java
import java.util.Stack;

class RedundantBrackets {
    public boolean checkRedundantBrackets(String expression) {
        Stack<Character> stack = new Stack<>();
        for (char ch : expression.toCharArray()) {
            if (ch == ')') {
                char top = stack.pop();
                boolean flag = true;
                while (top != '(') {
                    if (top == '+' || top == '-' || top == '*' || top == '/') {
                        flag = false;
                    }
                    top = stack.pop();
                }
                if (flag) return true;
            } else {
                stack.push(ch);
            }
        }
        return false;
    }
}
```

- **Time Complexity**: O(n).

---

7. Infix to Postfix Conversion

Problem: Convert an infix expression to postfix.
Example:

- Input: `` `(a+b)*(c+d)` ``
- Output: `` `ab+cd+*` ``

Java Code:

```java
import java.util.Stack;

class InfixToPostfix {
    public String infixToPostfix(String expression) {
        StringBuilder result = new StringBuilder();
        Stack<Character> stack = new Stack<>();
        for (char ch : expression.toCharArray()) {
            if (Character.isLetterOrDigit(ch)) {
                result.append(ch);
            } else if (ch == '(') {
                stack.push(ch);
            } else if (ch == ')') {
                while (!stack.isEmpty() && stack.peek() != '(') {
                    result.append(stack.pop());
                }
                stack.pop();
            } else {
                while (!stack.isEmpty() && precedence(ch) <= precedence(stack.peek())) {
                    result.append(stack.pop());
                }
                stack.push(ch);
            }
        }
        while (!stack.isEmpty()) {
            result.append(stack.pop());
        }
        return result.toString();
    }

    private int precedence(char ch) {
        switch (ch) {
            case '+':
            case '-':
                return 1;
            case '*':
            case '/':
                return 2;
            default:
                return -1;
        }
    }
}
```

- **Time Complexity**: O(n).

---

8. Next Smaller Element

Problem: Find the next smaller element for each element in an array.
Example:

- Input: `[4, 8, 5, 2, 25]`
- Output: `[2, 5, 2, -1, -1]`

Java Code:

```java
import java.util.Stack;

class NextSmallerElement {
    public int[] nextSmallerElement(int[] arr) {
        int[] result = new int[arr.length];
        Stack<Integer> stack = new Stack<>();
        for (int i = arr.length - 1; i >= 0; i--) {
            while (!stack.isEmpty() && stack.peek() >= arr[i]) {
                stack.pop();
            }
            result[i] = stack.isEmpty() ? -1 : stack.peek();
            stack.push(arr[i]);
        }
        return result;
    }
}
```

- **Time Complexity**: O(n).

---

9. Min Stack

Problem: Design a stack that supports push, pop, and retrieving the minimum element in constant time.

Example:

- Input:
    - `push(3)`, `push(5)`, `min()`, `push(2)`, `min()`
- Output:
    - `min()` -> 3, `min()` -> 2

Java Code:

```java
import java.util.Stack;

class MinStack {
    Stack<Integer> stack = new Stack<>();
    Stack<Integer> minStack = new Stack<>();

    public void push(int x) {
        stack.push(x);
        if (minStack.isEmpty() || x <= minStack.peek()) {
            minStack.push(x);
        }
    }

    public int pop() {
        if (stack.peek().equals(minStack.peek())) {
            minStack.pop();
        }
        return stack.pop();
    }

    public int getMin() {
        return minStack.peek();
    }
}
```

- **Time Complexity**: O(1) for all operations.

---

10. Find Largest Rectangle in Binary Matrix

Problem: Find the largest rectangle of 1’s in a binary matrix.
Example:

- Input:

```
1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0
```

- Output: 6 (largest rectangle has area 6)

Java Code:

java

```java
class LargestRectangleInMatrix {
    public int maximalRectangle(int[][] matrix) {
        if (matrix.length == 0) return 0;
        int[] heights = new int[matrix[0].length];
        int maxArea = 0;
        for (int[] row : matrix) {
            for (int i = 0; i < row.length; i++) {
                heights[i] = row[i] == 0 ? 0 : heights[i] + 1;
            }
            maxArea = Math.max(maxArea, largestRectangleArea(heights));
        }
        return maxArea;
    }

    private int largestRectangleArea(int[] heights) {
        Stack<Integer> stack = new Stack<>();
        int maxArea = 0, i = 0;
        while (i < heights.length) {
            if (stack.isEmpty() || heights[i] >= heights[stack.peek()]) {
                stack.push(i++);
            } else {
                int h = heights[stack.pop()];
                int width = stack.isEmpty() ? i : i - stack.peek() - 1;
                maxArea = Math.max(maxArea, h * width);
            }
        }
        while (!stack.isEmpty()) {
            int h = heights[stack.pop()];
            int width = stack.isEmpty() ? i : i - stack.peek() - 1;
            maxArea = Math.max(maxArea, h * width);
        }
        return maxArea;
    }
}
```

- **Time Complexity**: O(m * n), where m is the number of rows and n is the number of columns.

These solutions should help clarify the problems and provide efficient approaches! Let me know if you need further explanation or adjustments.

Intermediate Level:
Celebrity Problem: In a party of N people, find the celebrity (a person who knows no one, but everyone knows them) using a stack.
Reverse a Queue using Stack: Given a queue, reverse its elements using a stack.
Trapping Rain Water Problem: Given heights of bars, calculate the amount of water that can be trapped after rain.
Simplify Directory Path: Given an absolute path for a file, simplify it using a stack (Unix-style path simplification).
Remove K Digits: Given a non-negative integer represented as a string, remove k digits to form the smallest possible number using a stack.
Design a Stack with Get Middle Operation: Implement a stack that allows finding and deleting the middle element in constant time.
Advanced Level:
Maximum Width Ramp: Given an array of integers, find the maximum width of a ramp. A ramp is a pair (i, j) where i < j and A[i] <= A[j].
Largest Rectangle in Histogram with Width K: Find the largest rectangle that can be formed with a

maximum width of k in a histogram.
Largest Area in Histogram with Multiple Constraints: Solve the largest area in the histogram problem with additional constraints, such as maximum height or width restrictions.
Maximum Rectangular Area in Matrix with 0s and 1s: Extend the max rectangle problem to a matrix containing both 0s and 1s and find the largest rectangular area.

Let's dive into these intermediate and advanced problems with explanations, examples, and optimized Java code.

Intermediate Level

1. Celebrity Problem

Problem: In a party of N people, find the celebrity (a person who knows no one, but everyone knows them) using a stack.
Example:

- Input:

  ```csharp
  matrix = [
    [0, 1, 0],
    [0, 0, 0],
    [0, 1, 0]
  ]
  ```

  - Here, person 1 knows everyone, and everyone knows person 1.

Java Code:

```java
import java.util.Stack;

class CelebrityProblem {
    public int findCelebrity(int[][] matrix) {
        Stack<Integer> stack = new Stack<>();
        int n = matrix.length;

        for (int i = 0; i < n; i++) {
            stack.push(i);
        }

        while (stack.size() > 1) {
            int a = stack.pop();
            int b = stack.pop();
            if (knows(matrix, a, b)) {
                stack.push(b);
            } else {
                stack.push(a);
            }
        }

        int potentialCelebrity = stack.pop();
        for (int i = 0; i < n; i++) {
```

```
            if (i != potentialCelebrity) {
                if (knows(matrix, potentialCelebrity, i) || !knows(matrix, i,
potentialCelebrity)) {
                    return -1;
                }
            }
        }
        return potentialCelebrity;
    }

    private boolean knows(int[][] matrix, int a, int b) {
        return matrix[a][b] == 1;
    }
}
```

- **Time Complexity**: O(N^2) for checking all relationships, but finding the celebrity is linear, O(N).

---

2. Reverse a Queue using Stack

Problem: Given a queue, reverse its elements using a stack.
Example:

- Input: Queue = `` `[1, 2, 3, 4, 5]` ``
- Output: Queue = `` `[5, 4, 3, 2, 1]` ``

Java Code:

```java
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

class ReverseQueueUsingStack {
    public Queue<Integer> reverseQueue(Queue<Integer> queue) {
        Stack<Integer> stack = new Stack<>();
        while (!queue.isEmpty()) {
            stack.push(queue.poll());
        }
        while (!stack.isEmpty()) {
            queue.add(stack.pop());
        }
        return queue;
    }
}
```

- **Time Complexity**: O(n), where n is the number of elements in the queue.

---

3. Trapping Rain Water Problem

Problem: Given heights of bars, calculate the amount of water that can be trapped after rain.
Example:

- Input: `` `[4, 2, 0, 3, 2, 5]` ``
- Output: `` `9` `` (amount of trapped water)

Java Code:

```java
class TrappingRainWater {
    public int trap(int[] height) {
        int n = height.length;
        if (n == 0) return 0;

        int left = 0, right = n - 1;
        int leftMax = 0, rightMax = 0;
        int waterTrapped = 0;

        while (left <= right) {
            if (height[left] <= height[right]) {
                if (height[left] >= leftMax) {
                    leftMax = height[left];
                } else {
                    waterTrapped += leftMax - height[left];
                }
                left++;
            } else {
                if (height[right] >= rightMax) {
                    rightMax = height[right];
                } else {
                    waterTrapped += rightMax - height[right];
                }
                right--;
            }
        }
        return waterTrapped;
    }
}
```

- **Time Complexity**: O(n), where n is the number of elements in the height array.

---

4. Simplify Directory Path

Problem: Given an absolute path for a file, simplify it using a stack (Unix-style path simplification).
Example:

- Input: `` `"/a/./b/../../c/"` ``
- Output: `` `"/c"` ``

Java Code:

```java
import java.util.Stack;

class SimplifyDirectoryPath {
    public String simplifyPath(String path) {
        Stack<String> stack = new Stack<>();
        String[] parts = path.split("/");

        for (String part : parts) {
            if (part.equals("..")) {
                if (!stack.isEmpty()) {
                    stack.pop();
                }
            } else if (!part.isEmpty() && !part.equals(".")) {
                stack.push(part);
            }
        }
        StringBuilder sb = new StringBuilder();
        for (String s : stack) {
            sb.append("/").append(s);
        }
        return sb.length() == 0 ? "/" : sb.toString();
    }
}
```

- **Time Complexity**: O(n), where n is the length of the path string.

---

5. Remove K Digits

Problem: Given a non-negative integer represented as a string, remove k digits to form the smallest possible number using a stack.
Example:

- Input: `num = "1432219", k = 3`
- Output: `"1219"`

Java Code:

```java
import java.util.Stack;

class RemoveKDigits {
    public String removeKdigits(String num, int k) {
        Stack<Character> stack = new Stack<>();
        for (char c : num.toCharArray()) {
            while (k > 0 && !stack.isEmpty() && stack.peek() > c) {
                stack.pop();
                k--;
            }
            stack.push(c);
        }
        while (k > 0) {
            stack.pop();
            k--;
        }

        StringBuilder sb = new StringBuilder();
        boolean leadingZero = true;
        for (char c : stack) {
            if (leadingZero && c == '0') continue;
            leadingZero = false;
            sb.append(c);
        }
        return sb.length() == 0 ? "0" : sb.toString();
    }
}
```

- **Time Complexity**: O(n), where n is the length of the num string.

---

6. Design a Stack with Get Middle Operation

Problem: Implement a stack that allows finding and deleting the middle element in constant time.
Example:

- Input:
    - `push(1)`, `push(2)`, `push(3)`, `getMiddle()`, `pop()`, `getMiddle()`
- Output:
    - `getMiddle()` -> 2, `getMiddle()` -> 1

Java Code:

```java
import java.util.Stack;

class StackWithMiddle {
    private Stack<Integer> stack = new Stack<>();
    private Stack<Integer> middleStack = new Stack<>();

    public void push(int x) {
        stack.push(x);
        if (middleStack.isEmpty()) {
            middleStack.push(x);
        } else {
            if (stack.size() % 2 == 1) {
                middleStack.push(x);
            }
        }
    }

    public int pop() {
        if (stack.isEmpty()) return -1;
        int removed = stack.pop();
        if (stack.size() % 2 == 0) {
            middleStack.pop();
        }
        return removed;
    }

    public int getMiddle() {
        return middleStack.isEmpty() ? -1 : middleStack.peek();
    }
}
```

- **Time Complexity**: O(1) for `push()`, `pop()`, and `getMiddle()` operations.

Advanced Level

1. Maximum Width Ramp

Problem: Given an array of integers, find the maximum width of a ramp. A ramp is a pair `(i, j)` where `i < j` and `A[i] <= A[j]`.
Example:

- Input: `[6, 0, 8, 2, 1, 5]`
- Output: `4` (ramp between indices 1 and 5)

Java Code:

```java
class MaximumWidthRamp {
    public int maxWidthRamp(int[] A) {
        int n = A.length;
        int[] minIndices = new int[n];
        minIndices[0] = 0;
        for (int i = 1; i < n; i++) {
            if (A[i] < A[minIndices[i - 1]]) {
                minIndices[i] = i;
            } else {
                minIndices[i] = minIndices[i - 1];
            }
        }

        int maxWidth = 0;
        for (int j = n - 1; j >= 0; j--) {
            while (minIndices[j] <= j) {
                maxWidth = Math.max(maxWidth, j - minIndices[j]);
                break;
            }
        }
        return maxWidth;
    }
}
```

- **Time Complexity**: O(n), where n is the length of the array.

---

2. Largest Rectangle in Histogram with Width K

Problem: Find the largest rectangle that can be formed with a maximum width of `k` in a histogram.
Example:

- Input: `[2, 1, 5, 6, 2, 3]`, `k = 2`
- Output: `10` (rectangle with height 5 and width 2)

Java Code:

```java
import java.util.Stack;

class LargestRectangleInHistogramWithWidthK {
    public int largestRectangleArea(int[] heights, int k) {
        int n = heights.length;
        Stack<Integer> stack = new Stack<>();
        int maxArea = 0;

        for (int i = 0; i < n; i++) {
            while (!stack.isEmpty() && heights[i] < heights[stack.peek()]) {
                int height = heights[stack.pop()];
                int width = stack.isEmpty() ? i : i - stack.peek() - 1;
                width = Math.min(width, k);
                maxArea = Math.max(maxArea, height * width);
            }
            stack.push(i);
        }

        while (!stack.isEmpty()) {
            int height = heights[stack.pop()];
            int width = stack.isEmpty() ? n : n - stack.peek() - 1;
            width = Math.min(width, k);
            maxArea = Math.max(maxArea, height * width);
        }

        return maxArea;
    }
}
```

- **Time Complexity**: O(n), where n is the number of bars in the histogram.

---

3. Largest Area in Histogram with Multiple Constraints

Problem: Solve the largest area in the histogram problem with additional constraints, such as maximum height or width restrictions.
Example:

- Input: `[2, 1, 5, 6, 2, 3]`, `maxHeight = 4`
- Output: `10` (rectangle with height 5 and width 2, but restricted to maxHeight 4)

Java Code:

```java
import java.util.Stack;

class LargestAreaInHistogramWithConstraints {
    public int largestRectangleArea(int[] heights, int maxHeight) {
        int n = heights.length;
        Stack<Integer> stack = new Stack<>();
        int maxArea = 0;

        for (int i = 0; i < n; i++) {
            while (!stack.isEmpty() && heights[i] < heights[stack.peek()]) {
                int height = Math.min(heights[stack.pop()], maxHeight);
                int width = stack.isEmpty() ? i : i - stack.peek() - 1;
                maxArea = Math.max(maxArea, height * width);
            }
            stack.push(i);
        }

        while (!stack.isEmpty()) {
            int height = Math.min(heights[stack.pop()], maxHeight);
            int width = stack.isEmpty() ? n : n - stack.peek() - 1;
            maxArea = Math.max(maxArea, height * width);
        }

        return maxArea;
    }
}
```

- **Time Complexity**: O(n), where n is the number of bars in the histogram.

---

4. Maximum Rectangular Area in Matrix with 0s and 1s

Problem: Extend the max rectangle problem to a matrix containing both 0s and 1s and find the largest rectangular area.
Example:

- Input:

```csharp
[
  [1, 0, 1, 0, 0],
  [1, 0, 1, 1, 1],
  [1, 1, 1, 1, 1],
  [1, 0, 0, 1, 0]
]
```

- Output: `6` (largest rectangle of 1's)

Java Code:

```java
class MaximumRectangularAreaInMatrix {
    public int maximalRectangle(char[][] matrix) {
        if (matrix.length == 0 || matrix[0].length == 0) return 0;
        int m = matrix.length;
        int n = matrix[0].length;
        int[] heights = new int[n];
        int maxArea = 0;

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                heights[j] = matrix[i][j] == '1' ? heights[j] + 1 : 0;
            }
            maxArea = Math.max(maxArea, largestRectangleArea(heights));
        }
        return maxArea;
    }

    private int largestRectangleArea(int[] heights) {
        Stack<Integer> stack = new Stack<>();
        int maxArea = 0;
        int i = 0;

        while (i < heights.length) {
            if (stack.isEmpty() || heights[i] >= heights[stack.peek()]) {
                stack.push(i++);
            } else {
                int height = heights[stack.pop()];
                int width = stack.isEmpty() ? i : i - stack.peek() - 1;
                maxArea = Math.max(maxArea, height * width);
            }
        }

        while (!stack.isEmpty()) {
            int height = heights[stack.pop()];
            int width = stack.isEmpty() ? i : i - stack.peek() - 1;
            maxArea = Math.max(maxArea, height * width);
        }

        return maxArea;
    }
}
```

- **Time Complexity**: O(m * n), where m is the number of rows and n is the number of columns.

These solutions should provide you with a good understanding of each problem and how to tackle them efficiently. Let me know if you have any questions or need further clarifications!

ChatGPT can make mistakes. Check important info.