

Stack Solutions

Solution 1:

Time Complexity : $O(n)$

Space Complexity: $O(n)$

```
import java.util.*;
```

```
class Solution {  
    public static void main(String args[]){  
        Node one = new Node(1);  
        Node two = new Node(2);  
        Node three = new Node(3);  
        Node four = new Node(4);  
        Node five = new Node(3);  
        Node six = new Node(2);  
        Node seven = new Node(1);  
        one.ptr = two;  
        two.ptr = three;  
        three.ptr = four;  
        four.ptr = five;  
        five.ptr = six;  
        six.ptr = seven;  
        boolean condition = isPalindrome(one);  
        System.out.println("Palindrome :" + condition);  
    }  
    static boolean isPalindrome(Node head){  
  
        Node slow = head;  
        boolean ispalin = true;  
        Stack<Integer> stack = new Stack<Integer>();  
  
        while (slow != null) {  
            stack.push(slow.data);  
            slow = slow.ptr;  
        }  
  
        while (head != null) {
```

```

        int i = stack.pop();
        if (head.data == i) {
            ispalin = true;
        }
        else {
            ispalin = false;
            break;
        }
        head = head.ptr;
    }
    return ispalin;
}
}

```

```

class Node {
    int data;
    Node ptr;
    Node(int d){
        ptr = null;
        data = d;
    }
}

```

APNA
COLLEGE

Solution 2 :

Time Complexity : $O(n)$

Space Complexity: $O(1)$

```

import java.io.*;
import java.util.*;

```

```

class Solution{
    public static void main(String []args){
        String str = new String("/a/./b/../../c/");
        String res = simplify(str);
        System.out.println(res);
    }
}

```

```
static String simplify(String A){
    Stack<String> st = new Stack<String>();
    String res = "";
    res += "/";
    int len_A = A.length();

    for (int i = 0; i < len_A; i++){
        String dir = "";
        while (i < len_A && A.charAt(i) == '/')
            i++;

        while (i < len_A && A.charAt(i) != '/'){
            dir += A.charAt(i);
            i++;
        }

        if (dir.equals("..") == true){
            if (!st.empty())
                st.pop();
        }

        else if (dir.equals(".") == true)
            continue;

        else if (dir.length() != 0)
            st.push(dir);
    }

    Stack<String> st1 = new Stack<String>();
    while (!st.empty()){

        st1.push(st.pop());
    }

    while (!st1.empty()){
        if (st1.size() != 1)
            res += (st1.pop() + "/");
        else
            res += st1.pop();
    }
}
```

```

        return res;
    }

}

```

Solution 3 :

Time Complexity : $O(n)$

Space Complexity: $O(n)$

```
import java.util.Stack;
```

```

class Solution{
    static String decode(String str){
        Stack<Integer> integerstack = new Stack<>();
        Stack<Character> stringstack = new Stack<>();
        String temp = "", result = "";
        for (int i = 0; i < str.length(); i++){
            int count = 0;
            if (Character.isDigit(str.charAt(i))){
                while (Character.isDigit(str.charAt(i))){
                    count = count * 10 + str.charAt(i) - '0';
                    i++;
                }

                i--;
                integerstack.push(count);
            }

            else if (str.charAt(i) == ' '){
                temp = "";
                count = 0;

                if (!integerstack.isEmpty()){
                    count = integerstack.peek();
                    integerstack.pop();
                }

                while (!stringstack.isEmpty() && stringstack.peek() != '['){

```

```
        temp = stringstack.peek() + temp;
        stringstack.pop();
    }
```

```
    if (!stringstack.empty() && stringstack.peek() == '[')
        stringstack.pop();
```

```
    for (int j = 0; j < count; j++)
        result = result + temp;
```

```
    for (int j = 0; j < result.length(); j++)
        stringstack.push(result.charAt(j));
```

```
    result = "";
```

```
}
```

```
else if (str.charAt(i) == '['){
    if (Character.isDigit(str.charAt(i-1)))
        stringstack.push(str.charAt(i));
```

```
    else{
        stringstack.push(str.charAt(i));
        integerstack.push(1);
    }
```

```
}
```

```
else
    stringstack.push(str.charAt(i));
```

```
}
```

```
while (!stringstack.isEmpty()){
    result = stringstack.peek() + result;
    stringstack.pop();
```

```
}
```

```
return result;
```

```
}
```

```
public static void main(String args[]){
    String str = "3[b2[ca]]";
    System.out.println(decode(str));
```

```
}  
}
```

Solution 4 :

Time Complexity : $O(n)$

Space Complexity: $O(n)$

```
import java.io.*;  
import java.util.*;
```

```
class Solution{  
  
    public static int maxWater(int[] height){  
        Stack<Integer> stack = new Stack<>();  
        int n = height.length;  
        int ans = 0;  
        for (int i = 0; i < n; i++) {  
            while ((!stack.isEmpty())  
                && (height[stack.peek()] < height[i])) {  
                int pop_height = height[stack.peek()];  
                stack.pop();  
                if (stack.isEmpty())  
                    break;  
                int distance = i - stack.peek() - 1;  
                int min_height  
                    = Math.min(height[stack.peek()],  
                                height[i])  
                    - pop_height;  
  
                ans += distance * min_height;  
            }  
            stack.push(i);  
        }  
  
        return ans;  
    }  
  
    public static void main(String[] args){  
        int arr[] = { 0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1 };  
    }  
}
```

```
System.out.print(maxWater(arr));
```

```
}
```

```
}
```