**Data Structures and Algorithms**

Logbook Question Paper

**Note: Attempt all the Sections and add both answers and questions on the answer Book.**

**Section A**

**Note:** Attempt all three of the following questions and add proper comments and output.

1. Implement a stack using a singly linked list with the following operations:

**push(int val)**: Adds an element to the top of the stack.

**pop()**: Removes and returns the top element of the stack.

**peek():** Returns the top element without removing it.

**isEmpty():** Returns true if the stack is empty, otherwise false.

2. Given an array arr[] of non-negative integers and a value sum, the task is to check if there is a subset of the given array whose sum is equal to the given sum. Write code using recursion with explanation of code

Examples:

Input: arr[] = [3, 34, 4, 12, 5, 2], sum = 9

Output: True

Explanation: There is a subset (4, 5) with sum 9.

Input: arr[] = [3, 34, 4, 12, 5, 2], sum = 30

Output: False

Explanation: There is no subset that adds up to 30.

Constraints:

1 <= arr.size() <= 200

1<= arr[i] <= 200

1<= sum <= 104

3. Implement the code for tower of hanoi and explain with the help of call stack

# 

# 

# 

# **Section B**

**Note:** Attempt all three of the following questions and add proper comments and output.

1.**Final Prices With a Special Discount in a Shop**

You are given an integer array prices where prices[i] is the price of the ith item in a shop.

There is a special discount for items in the shop. If you buy the ith item, then you will receive a discount equivalent to prices[j] where j is the minimum index such that j > i and prices[j] <= prices[i]. Otherwise, you will not receive any discount at all.

Return an integer array answer where answer[i] is the final price you will pay for the ith item of the shop, considering the special discount.

**Example 1:**

Input: prices = [8,4,6,2,3]

Output: [4,2,4,2,3]

Explanation:

For item 0 with price[0]=8 you will receive a discount equivalent to prices[1]=4, therefore, the final price you will pay is 8 - 4 = 4.

For item 1 with price[1]=4 you will receive a discount equivalent to prices[3]=2, therefore, the final price you will pay is 4 - 2 = 2.

For item 2 with price[2]=6 you will receive a discount equivalent to prices[3]=2, therefore, the final price you will pay is 6 - 2 = 4.

For items 3 and 4 you will not receive any discount at all.

**Example 2:**

Input: prices = [1,2,3,4,5]

Output: [1,2,3,4,5]

Explanation: In this case, for all items, you will not receive any discount at all.

**Example 3:**

Input: prices = [10,1,1,6]

Output: [9,0,1,6]

**Constraints:**

1 <= prices.length <= 500

1 <= prices[i] <= 1000

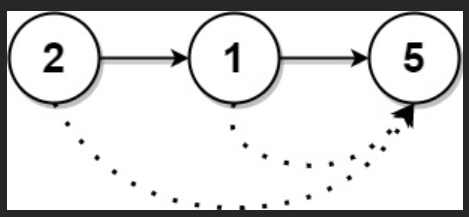
2. **Next Greater Node In Linked List**

You are given the head of a linked list with n nodes.

For each node in the list, find the value of the next greater node. That is, for each node, find the value of the first node that is next to it and has a strictly larger value than it.

Return an integer array answer where answer[i] is the value of the next greater node of the ith node (1-indexed). If the ith node does not have a next greater node, set answer[i] = 0.

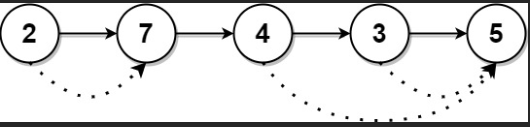
**Example 1**



Input: head = [2,1,5]

Output: [5,5,0]

**Example 2**



Input: head = [2,7,4,3,5]

Output: [7,0,5,5,0]

**Constraints:**

The number of nodes in the list is n.

1 <= n <= 104

1 <= Node.val <= 109

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

**class Solution {**

**public int[] nextLargerNodes(ListNode head) {**

//write the code here

**}**

**}**

3. **Predict the Winner using Recursion**

You are given an integer array nums. Two players are playing a game with this array: player 1 and player 2.

Player 1 and player 2 take turns, with player 1 starting first. Both players start the game with a score of 0. At each turn, the player takes one of the numbers from either end of the array (i.e., nums[0] or nums[nums.length - 1]) which reduces the size of the array by 1. The player adds the chosen number to their score. The game ends when there are no more elements in the array.

Return true if Player 1 can win the game. If the scores of both players are equal, then player 1 is still the winner, and you should also return true. You may assume that both players are playing optimally.

**Example 1:**

Input: nums = [1,5,2]

Output: false

Explanation: Initially, player 1 can choose between 1 and 2.

If he chooses 2 (or 1), then player 2 can choose from 1 (or 2) and 5. If player 2 chooses 5, then player 1 will be left with 1 (or 2).

So, final score of player 1 is 1 + 2 = 3, and player 2 is 5.

Hence, player 1 will never be the winner and you need to return false.

**Example 2:**

Input: nums = [1,5,233,7]

Output: true

Explanation: Player 1 first chooses 1. Then player 2 has to choose between 5 and 7. No matter which number player 2 choose, player 1 can choose 233.

Finally, player 1 has more score (234) than player 2 (12), so you need to return True representing player1 can win.

**Constraints:**

1 <= nums.length <= 20

0 <= nums[i] <= 107

code:

**class Solution {**

**public boolean predictTheWinner(int[] nums) {**

//write the code here

**}**

**}**

# **Section C**

**Note:** Attempt All three of the following questions, add proper comments and output.

1. **Reorder List using stack**

You are given the head of a singly linked-list. The list can be represented as:

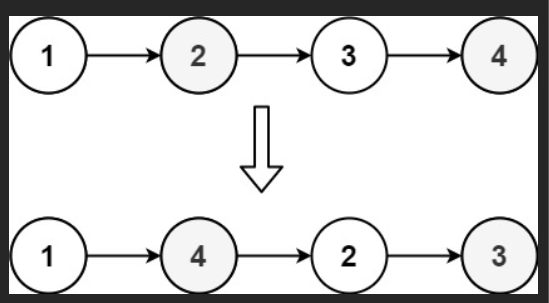
L0 → L1 → … → Ln - 1 → Ln

Reorder the list to be on the following form:

L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …

You may not modify the values in the list's nodes. Only nodes themselves may be changed.

**Example 1**



Input: head = [1,2,3,4]

Output: [1,4,2,3]

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

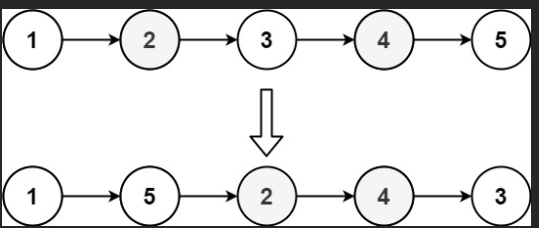
\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

.

**Example 2**

****

Input: head = [1,2,3,4,5]

Output: [1,5,2,4,3]

**Constraints:**

The number of nodes in the list is in the range [1, 5 \* 104].

1 <= Node.val <= 1000

code

**class Solution {**

**public void reorderList(ListNode head) {**

//write your code here

**}**

**}**

1. **Evaluate Reverse Polish Notation using stack**

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.

Note that:

The valid operators are '+', '-', '\*', and '/'.

Each operand may be an integer or another expression.

The division between two integers always truncates toward zero.

There will not be any division by zero.

The input represents a valid arithmetic expression in a reverse polish notation.

The answer and all the intermediate calculations can be represented in a 32-bit integer.

Example 1:

Input: tokens = ["2","1","+","3","\*"]

Output: 9

Explanation: ((2 + 1) \* 3) = 9

Example 2:

Input: tokens = ["4","13","5","/","+"]

Output: 6

Explanation: (4 + (13 / 5)) = 6

Example 3:

Input: tokens = ["10","6","9","3","+","-11","\*","/","\*","17","+","5","+"]

Output: 22

Explanation: ((10 \* (6 / ((9 + 3) \* -11))) + 17) + 5

= ((10 \* (6 / (12 \* -11))) + 17) + 5

= ((10 \* (6 / -132)) + 17) + 5

= ((10 \* 0) + 17) + 5

= (0 + 17) + 5

= 17 + 5

= 22

Constraints:

1 <= tokens.length <= 104

tokens[i] is either an operator: "+", "-", "\*", or "/", or an integer in the range [-200, 200].

Code

**class Solution {**

**public int evalRPN(String[] tokens) {**

//write your code here

**}**

**}**

3.**Special Binary String (using recursion)**

Special binary strings are binary strings with the following two properties:

The number of 0's is equal to the number of 1's.

Every prefix of the binary string has at least as many 1's as 0's.

You are given a special binary string s.

A move consists of choosing two consecutive, non-empty, special substrings of s, and swapping them. Two strings are consecutive if the last character of the first string is exactly one index before the first character of the second string.

Return the lexicographically largest resulting string possible after applying the mentioned operations on the string.

Example 1:

Input: s = "11011000"

Output: "11100100"

Explanation: The strings "10" [occuring at s[1]] and "1100" [at s[3]] are swapped.

This is the lexicographically largest string possible after some number of swaps.

Example 2:

Input: s = "10"

Output: "10"

Constraints:

1 <= s.length <= 50

s[i] is either '0' or '1'.

s is a special binary string.

**Code**

**class Solution {**

**public String makeLargestSpecial(String s) {**

//write your code here

**}**

**}**