Tower-of-Hanoi Problem

Problem | Submissions | Leaderboard | Discussions

Submitted 9 hours ago • Score: 10.00 Status: Accepted

✔ Test Case #0 ✔ Test Case #1

Submitted Code

Language: Java 8 Open in editor

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

class Solution{

    static void toh(int n,char s,char inter, char d){
        if(n==1)
            System.out.println("Disk "+n+" moved from " +s+ " to " +d);
        else
        {
            toh(n-1, s, d, inter);
            System.out.println("Disk "+n+" moved from " +s+ " to " +d);
            toh(n-1, inter,s, d);
        }
    }

    public static void main(String args[]){

        Scanner sc =new Scanner(System.in);
        int n = sc.nextInt();
        toh(n,'A','B','C');
        sc.close();

    }
}
```

Java String Reverse

Problem | Submissions | Leaderboard | Discussions

A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward or forward.

Given a string A, print `Yes` if it is a palindrome, print `No` otherwise.

Constraints

- A will consist at most 50 lower case english letters.

Sample Input

```
madam
```

Sample Output

```
Yes
```


Java 8

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

public class Solution {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);
        String A=sc.next();
        String res=new StringBuilder(A).reverse().toString();
        String rev=(A.equals(res))?"Yes":"No";
        System.out.print(rev);

    }
```

```
15 }
16
17
18
19
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Java Substring Comparisons

Problem | Submissions | Leaderboard | Discussions

We define the following terms:

- Lexicographical Order, also known as *alphabetic* or *dictionary* order, orders characters as follows:

  $$\mathtt{A} < \mathtt{B} < \ldots < \mathtt{Y} < \mathtt{Z} < \mathtt{a} < \mathtt{b} < \ldots < \mathtt{y} < \mathtt{z}$$

  For example, `ball < cat`, `dog < dorm`, `Happy < happy`, `Zoo < ball`.

- A substring of a string is a contiguous block of characters in the string. For example, the substrings of `abc` are `a`, `b`, `c`, `ab`, `bc`, and `abc`.

Given a string, s, and an integer, k, complete the function so that it finds the lexicographically *smallest* and *largest* substrings of length k.

Function Description

Complete the *getSmallestAndLargest* function in the editor below.

getSmallestAndLargest has the following parameters:

- *string s:* a string
- *int k:* the length of the substrings to find

Returns

- *string:* the string ' + "\n" + ' where and are the two substrings

Input Format

The first line contains a string denoting s.
The second line contains an integer denoting k.

Constraints

- $1 \leq |s| \leq 1000$
- s consists of English alphabetic letters only (i.e., `[a-zA-Z]`).

Sample Input 0

```
welcometojava
3
```

Sample Output 0

```
ava
wel
```

Explanation 0

String $s =$ **"welcometojava"** has the following lexicographically-ordered substrings of length $k = 3$:

["ava", "com", "elc", "eto", "jav", "lco", "met", "oja", "ome", "toj", "wel"]

We then return the first (lexicographically smallest) substring and the last (lexicographically largest) substring as two newline-separated values (i.e., `ava\nwel`).

The stub code in the editor then prints `ava` as our first line of output and `wel` as our second line of output.

Contest ends in 11 days

Submissions: 16
Max Score: 10
Difficulty: Easy

Rate This Challenge:

More

Java 8

```
import ⇔;

public class Solution {
    public static String getSmallestAndLargest(String s, int k) {
        String smallest = s.substring(0,k);
        String largest = s.substring(0,k);

        for(int i=0;i<= (s.length()-k) ;i++){
            smallest = s.substring(i,i+k).compareTo(smallest)<0?s.substring(i,i+k):smallest;

            largest = s.substring(i,i+k).compareTo(largest)>0?s.substring(i,i+k):largest;

        }
        return smallest + "\n" + largest;
    }


    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        String s = scan.next();
        int k = scan.nextInt();
        scan.close();

        System.out.println(getSmallestAndLargest(s, k));
    }
}
```

Line: 14 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Java Anagrams

Problem | Submissions | Leaderboard | Discussions

Two strings, a and b, are called anagrams if they contain all the same characters in the same frequencies. For this challenge, the test is not case-sensitive. For example, the anagrams of `CAT` are `CAT`, `ACT`, `tac`, `TCA`, `aTC`, and `CtA`.

Function Description

Complete the *isAnagram* function in the editor.

isAnagram has the following parameters:

- *string a:* the first string
- *string b:* the second string

Returns

- *boolean:* If a and b are case-insensitive anagrams, return true. Otherwise, return false.

Input Format

The first line contains a string a.
The second line contains a string b.

Constraints

- $1 \leq length(a), length(b) \leq 50$
- Strings a and b consist of English alphabetic characters.
- The comparison should NOT be case sensitive.

Sample Input 0

```
anagram
margana
```

Sample Output 0

```
Anagrams
```

Explanation 0

Character	Frequency: `anagram`	Frequency: `margana`
`A` or `a`	3	3
`G` or `g`	1	1

Character	Frequency: anagram	Frequency: margana
N or n	1	1
M or m	1	1
R or r	1	1

The two strings contain all the same letters in the same frequencies, so we print "Anagrams".

Sample Input 1

```
anagramm
marganaa
```

Sample Output 1

```
Not Anagrams
```

Explanation 1

Character	Frequency: anagramm	Frequency: marganaa
A or a	3	4
G or g	1	1
N or n	1	1
M or m	2	1
R or r	1	1

The two strings don't contain the same number of `a`'s and `m`'s, so we print "Not Anagrams".

Sample Input 2

```
Hello
hello
```

Sample Output 2

```
Anagrams
```

Explanation 2

Character	Frequency: Hello	Frequency: hello
E or e	1	1
H or h	1	1
L or l	2	2
O or o	1	1

The two strings contain all the same letters in the same frequencies, so we print "Anagrams".

Rate This Challenge:

More

Java 8

```
1  import ⇔;
2
3  public class Solution {
4      static boolean isAnagram(String a, String b) {
5          a=a.toUpperCase();
6          b=b.toUpperCase();
7
8          char[] A=a.toCharArray();
9          char[] B=b.toCharArray();
10
11         java.util.Arrays.sort(A);
12         java.util.Arrays.sort(B);
13
14         if(java.util.Arrays.equals(A, B)){
15             return true;
16         }
17         else{
18             return false;
19         }
20     }
21   public static void main(String[] args) {⇔}
30 }
31
```

Line: 19 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Java Strings Introduction

> "A string is traditionally a sequence of characters, either as a literal constant or as some kind of variable." — Wikipedia: String (computer science)

This exercise is to test your understanding of Java Strings. A sample *String* declaration:

```
String myString = "Hello World!"
```

The elements of a *String* are called *characters*. The number of *characters* in a *String* is called the *length*, and it can be retrieved with the *String.length()* method.

Given two strings of lowercase English letters, A and B, perform the following operations:

1. Sum the lengths of A and B.
2. Determine if A is lexicographically larger than B (i.e.: does B come before A in the dictionary?).
3. Capitalize the first letter in A and B and print them on a single line, separated by a space.

Input Format

The first line contains a string A. The second line contains another string B. The strings are comprised of only lowercase English letters.

Output Format

There are three lines of output:
For the first line, sum the lengths of A and B.
For the second line, write `Yes` if A is lexicographically greater than B otherwise print `No` instead.
For the third line, capitalize the first letter in both A and B and print them on a single line, separated by a space.

Sample Input 0

```
hello
java
```

Sample Output 0

```
9
No
Hello Java
```

Explanation 0

String A is "hello" and B is "java".

A has a *length* of 5, and B has a *length* of 4; the sum of their lengths is 9.
When sorted alphabetically/lexicographically, "hello" precedes "java"; therefore, A is not greater than B and the answer is `No`.

When you capitalize the first letter of both A and B and then print them separated by a space, you get "Hello Java".

Contest ends in 11 days

Submissions: 55
Max Score: 5
Difficulty: Easy

Rate This Challenge:

More

Java 8

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

public class Solution {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);
        String A=sc.next();
        String B=sc.next();

        int sum = A.length()+B.length();
        System.out.println(sum);
        if(A.compareTo(B)>B.compareTo(A)){
            System.out.println("Yes");
        }else{
            System.out.println("No");
        }
        A = A.substring(0,1).toUpperCase()+A.substring(1);
        B = B.substring(0,1).toUpperCase()+B.substring(1);

        System.out.println(A + " " + B);

    }
}
```

Line: 22 Col: 41

Upload Code as File Test against custom input Run Code Submit Code

R1: Sum of the series

Problem | Submissions | Leaderboard | Discussions

Recursive program to find the Sum of the series **1 – 1/2 + 1/3 – 1/4 ... 1/N** Given a positive integer N, the task is to find the sum of the series *1 – (1/2) + (1/3) – (1/4) +.... (1/N) using recursion.*

Input Format

Input: N = 3

Constraints

Nis a natural number.

Output Format

Output: 0.8333333 Explanation: 1 – (1/2) + (1/3) = 0.8333333

Sample Input 0

```
3
```

Sample Output 0

```
0.8333333730697632
```


Java 8

```
1 import java.io.*;
2 import java.util.*;
3
4 public class Solution {
5
6     static float series(float num){
7         if(num<=1){
```

```
            return 1;
        }else{
            if(num%2==0){
                return series(num-1)-1/num;
            }else{
                return series(num-1)+1/num;
            }
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        float n =sc.nextFloat();
        System.out.println(String.format("%.16f",series(n)));
        sc.close();
    }
}
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

R2:Negative numbers

Problem | Submissions | Leaderboard | Discussions

**Move all negative numbers to beginning and positive to end with constant extra space ** An array contains both positive and negative numbers in random order. Rearrange the array elements so that all negative numbers appear before all positive numbers.

Input Format

Input: -12, 11, -13, -5, 6, -7, 5, -3, -6

Constraints

All numbers are intergers

Output Format

Output: -12 -13 -5 -7 -3 -6 5 6 11

Sample Input 0

```
-12, 11, -13, -5, 6, -7, 5, -3, -6
```

Sample Output 0

```
-12 -13 -5 -7 -3 -6 11 6 5
```

Contest ends in 9 days

Submissions: 45
Max Score: 10
Difficulty: Medium

Rate This Challenge:

More

Java 8

```java
import java.util.ArrayList;

public class Solution {

    public static void main(String[] args) {
        int[] arr = {-12, 11, -13, -5, 6, -7, 5, -3, -6};

```

```
        ArrayList<Integer> negArrayList = new ArrayList<>();
        ArrayList<Integer> posArrayList = new ArrayList();

        for (int n : arr) {
            if (n < 0) {
                negArrayList.add(n);
            } else {
                posArrayList.add(n);
            }
        }

        negArrayList.addAll(posArrayList);
        for (int n : negArrayList) {
            System.out.print(n + " ");
        }

    }
}
```

Line: 1 Col: 1

Upload Code as File | Test against custom input | Run Code | Submit Code

R3: Max element

Problem | Submissions | Leaderboard | Discussions

Given a matrix, the task is to find the maximum element of each row.

Sample Input 0

```
{1, 2, 3}
{1, 4, 9}
{76, 34, 21}
```

Sample Output 0

```
3
9
76
```


Java 8

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

public class Solution {

    static int max( int[] arr , int max , int i ){
        if(arr[i] > max )
            max = arr[ i ];
        if( i < arr.length-1 )
            return max( arr , max, ++i);
        return max;
    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        for( int j = 0 ;j < 10 ; j++){
            String line;
            try{
                line = sc.nextLine();
            }
            catch( NoSuchElementException e ){
                return;
            }
            line = line.replace("{", " ");
            line = line.replace("}"," ");

            String[] s = line.split(",");
            int[] arr = new int[ s.length ];

```

```
31              //int max = 0;
32              for( int i = 0 ; i < s.length; i++){
33                      arr[i] = Integer.parseInt( s[i].trim() );
34                      //if( n > max )
35                          //max = n;
36
37              }
38              System.out.println(max(arr, 0 , 0));
39          }
40      }
41  }
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

ADS6:Linked List

Problem | Submissions | Leaderboard | Discussions

Linked List Implementation in Java

Input Format

Input:

Constraints

.

Output Format

1 -> 2 -> 3 -> 4 -> null

Sample Output 0

```
1 -> 2 -> 3 -> 4 -> null
```


Java 8

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

public class Solution {
    Node head;
    static class Node{
        Node next;
        int data;

        Node(int n){
            data=n;
            next=null;
        }
    }
    void display(){
            Node n = head;
            while(n!=null){
            System.out.print(n.data + " -> ");
            n=n.next;
            }
            System.out.print("null");
        }

    public static void main(String[] args){
        Solution s1 = new Solution();
        s1.head = new Node(1);
        Node N2 = new Node(2);
```

```
        Node N3 = new Node(3);
        Node N4 = new Node(4);

        s1.head.next=N2;
        N2.next=N3;
        N3.next=N4;

        s1.display();
    }
}
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Print the Elements of a Linked List

This challenge is part of a MyCodeSchool tutorial track and is accompanied by a video lesson.

This is an to practice traversing a *linked list*. Given a pointer to the *head* node of a linked list, print each node's $data$ element, one per line. If the head pointer is null (indicating the list is empty), there is nothing to print.

Function Description

Complete the *printLinkedList* function in the editor below.

printLinkedList has the following parameter(s):

- *SinglyLinkedListNode head:* a reference to the head of the list

Print

- For each node, print its $data$ value on a new line (console.log in Javascript).

Input Format

The first line of input contains n, the number of elements in the linked list.
The next n lines contain one element each, the $data$ values for each node.

Note: Do not read any input from stdin/console. Complete the printLinkedList function in the editor below.

Constraints

- $1 \leq n \leq 1000$
- $1 \leq list[i] \leq 1000$, where $list[i]$ is the i^{th} element of the linked list.

Sample Input

```
2
16
13
```

Sample Output

```
16
13
```

Explanation

There are two elements in the linked list. They are represented as 16 -> 13 -> NULL. So, the printLinkedList function should print 16 and 13 each on a new line.

Contest ends in 9 days

Submissions: 35
Max Score: 10
Difficulty: Easy

Rate This Challenge:

More

Java 8

```
import ⇔;

public class Solution {

    static class SinglyLinkedListNode {
        public int data;
        public SinglyLinkedListNode next;

        public SinglyLinkedListNode(int nodeData) {
            this.data = nodeData;
            this.next = null;
        }
    }

    static class SinglyLinkedList {
        public SinglyLinkedListNode head;
        public SinglyLinkedListNode tail;

        public SinglyLinkedList() {
            this.head = null;
            this.tail = null;
        }

        public void insertNode(int nodeData) {
            SinglyLinkedListNode node = new SinglyLinkedListNode(nodeData);

            if (this.head == null) {
                this.head = node;
            } else {
                this.tail.next = node;
            }

            this.tail = node;
        }
    }
```

```


    static void printLinkedList(SinglyLinkedListNode head) {
        while(head!=null){
            System.out.println(head.data);
            head = head.next;
        }

    }
```

```
    private static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        SinglyLinkedList llist = new SinglyLinkedList();

        int llistCount = scanner.nextInt();
        scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

        for (int i = 0; i < llistCount; i++) {
            int llistItem = scanner.nextInt();
            scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

            llist.insertNode(llistItem);
        }

        printLinkedList(llist.head);

        scanner.close();
    }
}

```

Line: 43 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Testcase 0 ✔ Testcase 1 ✔

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
2
16
13
```

Your Output (stdout)

```
16
13
```

Expected Output

```
16
13
```

Insert a node at the head of a linked list

This challenge is part of a tutorial track by MyCodeSchool and is accompanied by a video lesson.

Given a pointer to the head of a linked list, insert a new node before the head. The $next$ value in the new node should point to $head$ and the $data$ value should be replaced with a given value. Return a reference to the new head of the list. The head pointer given may be null meaning that the initial list is empty.

Function Description

Complete the function *insertNodeAtHead* in the editor below.

insertNodeAtHead has the following parameter(s):

- *SinglyLinkedListNode llist:* a reference to the head of a list
- *data:* the value to insert in the $data$ field of the new node

Input Format

The first line contains an integer n, the number of elements to be inserted at the head of the list.
The next n lines contain an integer each, the elements to be inserted, one per function call.

Constraints

- $1 \leq n \leq 1000$
- $1 \leq list[i] \leq 1000$

Sample Input

```
5
383
484
392
975
321
```

Sample Output

```
321
975
392
484
383
```

Explanation

Intially the list in NULL. After inserting 383, the list is 383 -> NULL.
After inserting 484, the list is 484 -> 383 -> NULL.
After inserting 392, the list is 392 -> 484 -> 383 -> NULL.
After inserting 975, the list is 975 -> 392 -> 484 -> 383 -> NULL.
After inserting 321, the list is 321 -> 975 -> 392 -> 484 -> 383 -> NULL.

Contest ends in 9 days

Submissions: 29
Max Score: 10
Difficulty: Easy

Rate This Challenge:

More

Java 8

```
import ⇔;

public class Solution {

    static class SinglyLinkedListNode {
        public int data;
        public SinglyLinkedListNode next;

        public SinglyLinkedListNode(int nodeData) {
            this.data = nodeData;
            this.next = null;
        }
    }

    static class SinglyLinkedList {
        public SinglyLinkedListNode head;
        public SinglyLinkedListNode tail;

        public SinglyLinkedList() {
            this.head = null;
            this.tail = null;
        }


    }

    public static void printSinglyLinkedList(SinglyLinkedListNode node, String sep,
BufferedWriter bufferedWriter) throws IOException {
        while (node != null) {
            bufferedWriter.write(String.valueOf(node.data));

            node = node.next;

            if (node != null) {
                bufferedWriter.write(sep);
            }
        }
    }

    // Complete the insertNodeAtHead function below.

    /*
     * For your reference:
     *
```

```
     * SinglyLinkedListNode {
     *     int data;
     *     SinglyLinkedListNode next;
     * }
     *
     */
    static SinglyLinkedListNode insertNodeAtHead(SinglyLinkedListNode llist, int data) {
            SinglyLinkedListNode newNode = new SinglyLinkedListNode(data);
            if(llist==null){
                llist=newNode;
                return llist;
            }
            newNode.next=llist;
            llist=newNode;
            return llist;
    }

    private static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) throws IOException {
        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));

        SinglyLinkedList llist = new SinglyLinkedList();

        int llistCount = scanner.nextInt();
        scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

        for (int i = 0; i < llistCount; i++) {
            int llistItem = scanner.nextInt();
            scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

          SinglyLinkedListNode llist_head = insertNodeAtHead(llist.head, llistItem);

          llist.head = llist_head;
        }



        printSinglyLinkedList(llist.head, "\n", bufferedWriter);
        bufferedWriter.newLine();

        bufferedWriter.close();

        scanner.close();
    }
}

```

Line: 23 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Tree: Inorder Traversal

In this challenge, you are required to implement inorder traversal of a tree.

Complete the *inOrder* function in your editor below, which has 1 parameter: a pointer to the root of a binary tree. It must print the values in the tree's inorder traversal as a single line of space-separated values.

Input Format

Our hidden tester code passes the root node of a binary tree to your $inOrder* function.

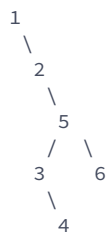
Constraints

$1 \leq Nodes\ in\ the\ tree \leq 500$

Output Format

Print the tree's inorder traversal as a single line of space-separated values.

Sample Input



Sample Output

```
1 2 3 4 5 6
```

Explanation

The tree's inorder traversal results in **1 2 3 4 5 6** as the required result.

Java 8

```java
import ⇔;

class Node {
    Node left;
    Node right;
    int data;

    Node(int data) {
        this.data = data;
        left = null;
        right = null;
    }
}

class Solution {
/* you only have to complete the function given below.
Node is defined as

class Node {
    int data;
    Node left;
    Node right;
}

*/

    public static void inOrder(Node root) {
        if(root == null)
            return;
        inOrder(root.left);
        System.out.print(root.data+" ");
        inOrder(root.right);
    }

    public static Node insert(Node root, int data) {⇔}

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int t = scan.nextInt();
        Node root = null;
        while(t-- > 0) {
            int data = scan.nextInt();
            root = insert(root, data);
        }
        scan.close();
        inOrder(root);
    }
}
```

Line: 34 Col: 6

Upload Code as File | Test against custom input | Run Code | Submit Code

Tree: Preorder Traversal

Complete the ***preOrder*** function in the editor below, which has **1** parameter: a pointer to the root of a binary tree. It must print the values in the tree's preorder traversal as a single line of space-separated values.

Input Format

Our test code passes the root node of a binary tree to the *preOrder* function.

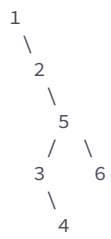
Constraints

$1 \leq$ Nodes in the tree ≤ 500

Output Format

Print the tree's preorder traversal as a single line of space-separated values.

Sample Input



Sample Output

```
1 2 5 3 4 6
```

Explanation

The preorder traversal of the binary tree is printed.

Java 8

```
import ⇔;

class Node {
    Node left;
    Node right;
    int data;

    Node(int data) {
        this.data = data;
        left = null;
        right = null;
    }
}

class Solution {
/* you only have to complete the function given below.
Node is defined as

class Node {
    int data;
    Node left;
    Node right;
}

*/

    public static void preOrder(Node root) {
        if(root == null)
            return;
        System.out.print(root.data+" ");
        preOrder(root.left);
        preOrder(root.right);
    }
    public static Node insert(Node root, int data) {⇔}

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int t = scan.nextInt();
        Node root = null;
        while(t-- > 0) {
            int data = scan.nextInt();
            root = insert(root, data);
        }
        scan.close();
        preOrder(root);
    }
}
```

Line: 20 Col: 1

Upload Code as File | Test against custom input | Run Code | Submit Code

Tree: Postorder Traversal

Complete the ***postOrder*** function in the editor below. It received **1** parameter: a pointer to the root of a binary tree. It must print the values in the tree's postorder traversal as a single line of space-separated values.

Input Format

Our test code passes the root node of a binary tree to the ***postOrder*** function.

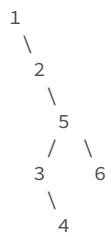
Constraints

$1 \leq$ Nodes in the tree ≤ 500

Output Format

Print the tree's postorder traversal as a single line of space-separated values.

Sample Input



Sample Output

```
4 3 6 5 2 1
```

Explanation

The postorder traversal is shown.

Java 8

```
import ⇔;

class Node {
    Node left;
    Node right;
    int data;

    Node(int data) {
        this.data = data;
        left = null;
        right = null;
    }
}

class Solution {
/* you only have to complete the function given below.
Node is defined as

class Node {
    int data;
    Node left;
    Node right;
}

*/

    public static void postOrder(Node root) {
        if(root == null)
            return;
        postOrder(root.left);
        postOrder(root.right);
        System.out.print(root.data+" ");
    }

    public static Node insert(Node root, int data) {⇔}

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int t = scan.nextInt();
        Node root = null;
        while(t-- > 0) {
            int data = scan.nextInt();
            root = insert(root, data);
        }
        scan.close();
        postOrder(root);
    }
}
```

Line: 20 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Insert a Node at the Tail of a Linked List

This challenge is part of a tutorial track by MyCodeSchool and is accompanied by a video lesson.

You are given the pointer to the head node of a linked list and an integer to add to the list. Create a new node with the given integer. Insert this node at the tail of the linked list and return the head node of the linked list formed after inserting this new node. The given head pointer may be null, meaning that the initial list is empty.

Function Description

Complete the *insertNodeAtTail* function in the editor below.

insertNodeAtTail has the following parameters:

- *SinglyLinkedListNode pointer head:* a reference to the head of a list
- *int data:* the data value for the node to insert

Returns

- *SinglyLinkedListNode pointer:* reference to the head of the modified linked list

Input Format

The first line contains an integer n, the number of elements in the linked list.
The next n lines contain an integer each, the value that needs to be inserted at tail.

Constraints

- $1 \leq n \leq 1000$
- $1 \leq list_i \leq 1000$

Sample Input

STDIN Function ----- -------- 5 size of linked list n = 5 141 linked list data values 141..474 302 164 530 474

Sample Output

```
141
302
164
530
474
```

Explanation

First the linked list is NULL. After inserting 141, the list is 141 -> NULL.
After inserting 302, the list is 141 -> 302 -> NULL.
After inserting 164, the list is 141 -> 302 -> 164 -> NULL.
After inserting 530, the list is 141 -> 302 -> 164 -> 530 -> NULL. After inserting 474, the list is 141 -> 302 -> 164 -> 530 -> 474 -> NULL, which is the final list.

Java 8

```
import ⇔;

public class Solution {

    static class SinglyLinkedListNode {
        public int data;
        public SinglyLinkedListNode next;

        public SinglyLinkedListNode(int nodeData) {
            this.data = nodeData;
            this.next = null;
        }
    }

    static class SinglyLinkedList {
        public SinglyLinkedListNode head;

        public SinglyLinkedList() {
            this.head = null;
        }


    }

    public static void printSinglyLinkedList(SinglyLinkedListNode node, String sep,
BufferedWriter bufferedWriter) throws IOException {
        while (node != null) {
            bufferedWriter.write(String.valueOf(node.data));

            node = node.next;

            if (node != null) {
                bufferedWriter.write(sep);
            }
        }
    }
```

```

    static SinglyLinkedListNode insertNodeAtTail(SinglyLinkedListNode head, int data) {
        SinglyLinkedListNode new_node = new SinglyLinkedListNode(data);
        if(head==null){
            head=new_node;
            return new_node;
        }
```

```
        new_node.next=null;
        SinglyLinkedListNode last = head;
        while(last.next !=null){
            last = last.next;
        }
        last.next = new_node;
        return head;
    }

    private static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) throws IOException {
        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));

        SinglyLinkedList llist = new SinglyLinkedList();

        int llistCount = scanner.nextInt();
        scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

        for (int i = 0; i < llistCount; i++) {
            int llistItem = scanner.nextInt();
            scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

          SinglyLinkedListNode llist_head = insertNodeAtTail(llist.head, llistItem);

          llist.head = llist_head;
        }



        printSinglyLinkedList(llist.head, "\n", bufferedWriter);
        bufferedWriter.newLine();

        bufferedWriter.close();

        scanner.close();
    }
}

```

Line: 42 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Reverse a List

You are given a list of N elements. Reverse the list without using the *reverse* function. The input and output portions will be handled automatically. You need to write a function with the recommended method signature.

Input Format

Each element, X, of the list is displayed on a separate line.

Output Format

The output is the reverse of the input list.

Sample Input

```
19
22
3
28
26
17
18
4
28
0
```

Sample Output

```
0
28
4
18
17
26
28
3
22
19
```

Method Signature

```
Number Of Parameters: 1
Parameters: [list]
Returns: List or Vector
```

Constraints

$1 <= N <= 100$
$0 <= X <= 100$.

For Hackers Using Clojure

This will be the outline of your function body (fill in the blank portion marked by underscores):

```
(fn[lst]_____________________________)
```

For Hackers Using Scala

This will be the outline of your function body (fill in the blank portion marked by underscores):

```
def f(arr:List[Int]):List[Int] = ___________________
```

For Hackers Using Haskell

This will be the outline of your function body (fill in the blank portion marked by underscores):

```
rev l = ___________________
```

For Hackers Using other Languages

You have to read input from standard input and write output to standard output. Please follow the input/output format mentioned above.

NOTE: You only need to submit the code above after filling in the blanks appropriately. The input and output section will be handled by us. The focus is on writing the correct function.

Scala

```
2 ▾def f(arr: List[Int]): List[Int] = arr match {
3     case Nil => Nil   // Base case: an empty list is reversed to an empty list
4     case head :: tail => f(tail) :+ head
5 }
```

Line: 1 Col: 1

Binary Search Tree : Insertion

You are given a pointer to the root of a binary search tree and values to be inserted into the tree. Insert the values into their appropriate position in the binary search tree and return the root of the updated binary tree. You just have to complete the function.

Input Format

You are given a function,

```
Node * insert (Node * root ,int data) {

}
```

Constraints

- No. of nodes in the tree ≤ 500

Output Format

Return the root of the binary search tree after inserting the value into the tree.

Sample Input

The value to be inserted is 6.

Sample Output

Rate This Challenge:

More

Java 8

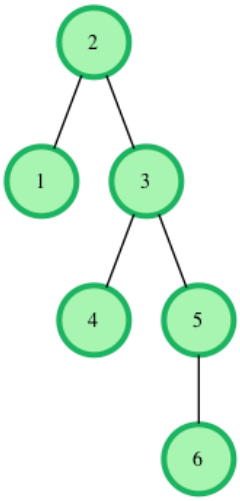
```
import ⇔;

class Node {
    Node left;
    Node right;
    int data;

    Node(int data) {
        this.data = data;
        left = null;
        right = null;
    }
}

class Solution {

    public static void preOrder( Node root ) {

        if( root == null)
            return;

        System.out.print(root.data + " ");
        preOrder(root.left);
        preOrder(root.right);

    }
public static Node insert(Node root,int data) {
        if(root == null){
            root = new Node(data);
            return root;
        }
        if(root.data > data){
            //left subtree
            root.left = insert(root.left, data);
        }
        else{
            //right subtree
            root.right = insert(root.right, data);
        }
        return root;

    }
    public static void main(String[] args) {⇔}
}
```

Line: 18 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Binary Search Tree : Lowest Common Ancestor

You are given pointer to the root of the binary search tree and two values $v1$ and $v2$. You need to return the lowest common ancestor (LCA) of $v1$ and $v2$ in the binary search tree.



In the diagram above, the lowest common ancestor of the nodes 4 and 6 is the node 3. Node 3 is the lowest node which has nodes 4 and 6 as descendants.

Function Description

Complete the function *lca* in the editor below. It should return a pointer to the lowest common ancestor node of the two values given.

lca has the following parameters:
- root: a pointer to the root node of a binary search tree
- v1: a node.data value
- v2: a node.data value

Input Format

The first line contains an integer, n, the number of nodes in the tree.
The second line contains n space-separated integers representing $node.data$ values.
The third line contains two space-separated integers, $v1$ and $v2$.

To use the test data, you will have to create the binary search tree yourself. Here on the platform, the tree will be created for you.

Constraints

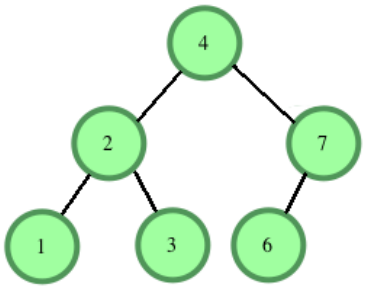
$1 \leq n, node.data \leq 25$
$1 \leq v1, v2 \leq 25$
$v1 \neq v2$
The tree will contain nodes with *data* equal to $v1$ and $v2$.

Output Format

Return the a pointer to the node that is the lowest common ancestor of $v1$ and $v2$.

Sample Input

```
6
4 2 3 1 7 6
1 7
```



$v1 = 1$ and $v2 = 7$.

Sample Output

[reference to node 4]

Explanation

LCA of 1 and 7 is 4, the root in this case.
Return a pointer to the node.

Contest ends in 6 days

Submissions: 10
Max Score: 30
Difficulty: Easy

Rate This Challenge:

More

Java 8

```
1  import ⇔;
3
4  class Node {
5      Node left;
6      Node right;
7      int data;
8
9      Node(int data) {
10         this.data = data;
11         left = null;
12         right = null;
13     }
```

```
}

class Solution {

    public static Node lca(Node root, int v1, int v2) {
        if(v1<root.data && v2<root.data){
            return lca(root.left, v1 , v2);
        }else if (v1 > root.data && v2 > root.data){
            return lca(root.right, v1 , v2);
        }else{
          return root;
        }
    }

    public static Node insert(Node root, int data) {}

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int t = scan.nextInt();
        Node root = null;
        while(t-- > 0) {
            int data = scan.nextInt();
            root = insert(root, data);
        }
        int v1 = scan.nextInt();
        int v2 = scan.nextInt();
        scan.close();
        Node ans = lca(root,v1,v2);
        System.out.println(ans.data);
    }
}
```

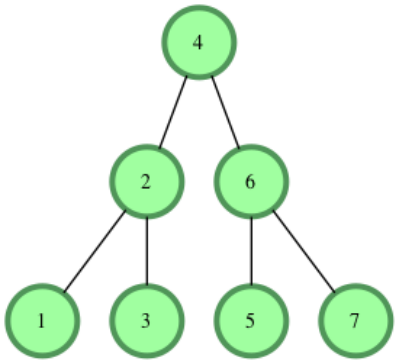
Line: 12 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Tree: Height of a Binary Tree

Problem | Submissions | Leaderboard | Discussions

The height of a binary tree is the number of edges between the tree's root and its furthest leaf. For example, the following binary tree is of height 2:



Function Description

Complete the *getHeight* or *height* function in the editor. It must return the height of a binary tree as an integer.

getHeight or height has the following parameter(s):

- *root*: a reference to the root of a binary tree.

Note -The Height of binary tree with single node is taken as zero.

Input Format

The first line contains an integer n, the number of nodes in the tree.
Next line contains n space separated integer where ith integer denotes node[i].data.

Note: Node values are inserted into a binary search tree before a reference to the tree's root node is passed to your function. In a binary search tree, all nodes on the left branch of a node are less than the node value. All values on the right branch are greater than the node value.

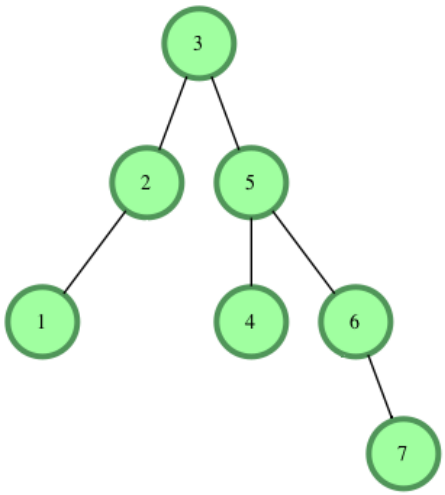
Constraints

$1 \leq node.data[i] \leq 20$
$1 \leq n \leq 20$

Output Format

Your function should return a single integer denoting the height of the binary tree.

Sample Input

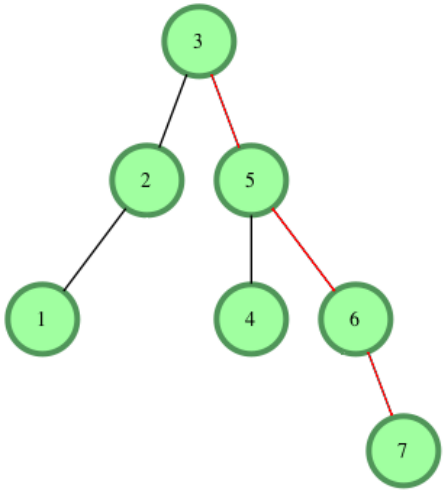


Sample Output

```
3
```

Explanation

The longest root-to-leaf path is shown below:



There are $\mathbf{4}$ nodes in this path that are connected by $\mathbf{3}$ edges, meaning our binary tree's $\boldsymbol{height} = \mathbf{3}$.

Contest ends in 6 days

Submissions: 21
Max Score: 10
Difficulty: Easy

Rate This Challenge:

More

Java 7

```
1  import ⇔;
3
4  class Node {
5      Node left;
```

```
    Node right;
    int data;

    Node(int data) {
        this.data = data;
        left = null;
        right = null;
    }
}

class Solution {

    public static int height(Node root) {
        if (root == null) {
            return -1;
        } else {
            int leftHeight = height(root.left);
            int rightHeight = height(root.right);
            return Math.max(leftHeight, rightHeight) + 1;
        }
    }

    public static Node insert(Node root, int data) {}

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int t = scan.nextInt();
        Node root = null;
        while(t-- > 0) {
            int data = scan.nextInt();
            root = insert(root, data);
        }
        scan.close();
        int height = height(root);
        System.out.println(height);
    }
}
```

Line: 12 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Merge two sorted linked lists

This challenge is part of a tutorial track by MyCodeSchool

Given pointers to the heads of two sorted linked lists, merge them into a single, sorted linked list. Either head pointer may be null meaning that the corresponding list is empty.

Example

$headA$ refers to $1 \rightarrow 3 \rightarrow 7 \rightarrow NULL$
$headB$ refers to $1 \rightarrow 2 \rightarrow NULL$

The new list is $1 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 7 \rightarrow NULL$

Function Description

Complete the *mergeLists* function in the editor below.

mergeLists has the following parameters:

- *SinglyLinkedListNode pointer headA:* a reference to the head of a list
- *SinglyLinkedListNode pointer headB:* a reference to the head of a list

Returns

- *SinglyLinkedListNode pointer:* a reference to the head of the merged list

Input Format

The first line contains an integer t, the number of test cases.

The format for each test case is as follows:

The first line contains an integer n, the length of the first linked list.
The next n lines contain an integer each, the elements of the linked list.
The next line contains an integer m, the length of the second linked list.
The next m lines contain an integer each, the elements of the second linked list.

Constraints

- $1 \leq t \leq 10$
- $1 \leq n, m \leq 1000$
- $1 \leq list[i] \leq 1000$, where $list[i]$ is the i^{th} element of the list.

Sample Input

```
1
3
```

```
1
2
3
2
3
4
```

Sample Output

```
1 2 3 3 4
```

Explanation

The first linked list is: $1 \rightarrow 3 \rightarrow 7 \rightarrow NULL$

The second linked list is: $3 \rightarrow 4 \rightarrow NULL$

Hence, the merged linked list is: $1 \rightarrow 2 \rightarrow 3 \rightarrow 3 \rightarrow 4 \rightarrow NULL$

Contest ends in 6 days

Submissions: 22
Max Score: 10
Difficulty: Easy

Rate This Challenge:

More

Java 8

```
1  import ⇔;
8
9  public class Solution {
10
11     static class SinglyLinkedListNode {
12         public int data;
13         public SinglyLinkedListNode next;
14
15         public SinglyLinkedListNode(int nodeData) {
16             this.data = nodeData;
17             this.next = null;
18         }
19     }
20
21     static class SinglyLinkedList {
22         public SinglyLinkedListNode head;
23         public SinglyLinkedListNode tail;
24
25         public SinglyLinkedList() {
26             this.head = null;
27             this.tail = null;
28         }
29
30         public void insertNode(int nodeData) {
31             SinglyLinkedListNode node = new SinglyLinkedListNode(nodeData);
32
33             if (this.head == null) {
34                 this.head = node;
35             } else {
36                 this.tail.next = node;
37             }
```

```
            this.tail = node;
        }
    }

    public static void printSinglyLinkedList(SinglyLinkedListNode node, String sep,
BufferedWriter bufferedWriter) throws IOException {
        while (node != null) {
            bufferedWriter.write(String.valueOf(node.data));

            node = node.next;

            if (node != null) {
                bufferedWriter.write(sep);
            }
        }
    }
```

```
    static SinglyLinkedListNode mergeLists(SinglyLinkedListNode head1, SinglyLinkedListNode
head2) {
        if (head1==null && head2==null){
            return null;
        }
        if (head1 !=null && head2==null){
            return head1;
        }
        if (head1==null & head2!=null){
            return head2;
        }

        SinglyLinkedListNode head;
        if(head1.data <= head2.data){
            head= new SinglyLinkedListNode(head1.data);
            head.next =mergeLists(head1.next,head2);
            }
            else{
            head= new SinglyLinkedListNode(head2.data);
            head.next =mergeLists(head1,head2.next);
            }
        return head;
    }
```

```
    private static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) throws IOException {
        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));

        int tests = scanner.nextInt();
        scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

        for (int testsItr = 0; testsItr < tests; testsItr++) {
            SinglyLinkedList llist1 = new SinglyLinkedList();

            int llist1Count = scanner.nextInt();
            scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

            for (int i = 0; i < llist1Count; i++) {
                int llist1Item = scanner.nextInt();
                scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

                llist1.insertNode(llist1Item);
            }

            SinglyLinkedList llist2 = new SinglyLinkedList();

```

```
            int llist2Count = scanner.nextInt();
            scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

            for (int i = 0; i < llist2Count; i++) {
                int llist2Item = scanner.nextInt();
                scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

                llist2.insertNode(llist2Item);
            }

            SinglyLinkedListNode llist3 = mergeLists(llist1.head, llist2.head);

            printSinglyLinkedList(llist3, " ", bufferedWriter);
            bufferedWriter.newLine();
        }

        bufferedWriter.close();

        scanner.close();
    }
}
```

Line: 25 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Your ADS7: Delete in Linked List submission got 10.00 points. Share Tweet

Try the Next Challenge | Contest Leaderboard

ADS7: Delete in Linked List

Problem Submissions Leaderboard Discussions

Delete the node at a given position in a linked list and return a reference to the head node. The head is at position 0. The list may be empty after you delete the node. In that case, return a null value.

Example

After removing the node at position , .

Function Description

Complete the deleteNode function in the editor below.

deleteNode has the following parameters: - SinglyLinkedListNode pointer llist: a reference to the head node in the list - int position: the position of the node to remove

Returns - SinglyLinkedListNode pointer: a reference to the head of the modified list

Input Format

The first line of input contains an integer n, the number of elements in the linked list. Each of the next n lines contains an integer, the node data values in order. The last line contains an integer,position, the position of the node to delete.

Constraints

1<=n<=1000

Output Format

Linked list deleted

Sample Input 0

```
8
20
6
2
19
7
4
15
9
3
```

Sample Output 0

Linked list deleted

Contest ends in 6 days

Submissions: 80
Max Score: 10
Difficulty: Medium

Rate This Challenge:

More

Java 8

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

class Node{
    int val;
    Node next;
    Node(int val){
        this.val=val;
        this.next=null;
    }
}
class LinkedListClass{
    Node head=null;
    public void addNode(int val){
        Node dummy = new Node(val);
        if(head==null){
            head = dummy;
        }else if(head!=null){
            Node curr = head;
            while(curr.next!=null){
                curr = curr.next;
            }
            curr.next = dummy;
        }
    }

    public void printList(){
        Node curr = head;
        while(curr!=null){
            System.out.print(curr.val + " -> ");
            curr = curr.next;
        }
        System.out.print("null");
    }
    public void deleteList(){
        head=null;
        if(head==null){
            System.out.print("Linked list deleted");
        }
    }
}
public class Solution {
    public static void main(String[] args) {
        LinkedListClass list = new LinkedListClass();
        list.addNode(8);
        list.addNode(20);
        list.addNode(6);
```

```
48          list.addNode(2);
49          list.addNode(19);
50          list.addNode(7);
51          list.addNode(4);
52          list.addNode(15);
53          list.addNode(9);
54          list.addNode(3);
55          list.deleteList();
56
57      }
58  }
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

ADS9: Insert a node at a specific position in a linked list

Given the pointer to the head node of a linked list and an integer to insert at a certain position, create a new node with the given integer as its data attribute, insert this node at the desired position and return the head node.

A position of 0 indicates head, a position of 1 indicates one node away from the head and so on. The head pointer given may be null meaning that the initial list is empty.

Example: head refers to the first node in the list 1->2->3 data = 4 position=2

Insert a node at position 2 with data =4. The new list is 1->2->4->3

Input Format

The first line contains an integer n, the number of elements in the linked list. Each of the next n lines contains an integer SinglyLinkedListNode[i].data. The next line contains an integer data, the data of the node that is to be inserted. The last line contains an integer position.

Insert element 1 at position 2 in given list of elements

Constraints

1<=n<=1000 0<=position<=n

Output Format

list

Sample Input 0

```
16
13
7
```

Sample Output 0

```
16 13 1 7
```

Explanation 0

Insert element 1 at position 2.

Submissions: 74
Max Score: 10
Difficulty: Medium

Rate This Challenge:

More

Java 8

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

public class Solution {


    Node head;

    static class Node {
        int data;
        Node next;

        Node(int d) {
            data = d;
            next = null;
        }
    }


    void InsertAtPos(int position, int data) {
        Node new_node = new Node(data);

        if (position < 1) {

            return;
        }

        if (position == 1) {
            new_node.next = head;
            head = new_node;
        } else {
            Node previous = head;
            for (int i = 0; i < position - 1; i++) {
                if (previous == null) {

                    return;
                }
                previous = previous.next;
            }
            if (previous == null) {

                return;
            }

            new_node.next = previous.next;
            previous.next = new_node;
        }
    }

    void Display() {
        Node current = head;
        while (current != null) {
            System.out.print(current.data+" ");
            current = current.next;
```

```
        }

    }

    public static void main(String[] args) {
        Solution L1 = new Solution();

        L1.head = new Node(16);
        Node second = new Node(13);
        Node third = new Node(7);

        L1.head.next = second;
        second.next = third;

        L1.InsertAtPos(2, 1);
        L1.Display();
    }


}
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Equal Stacks

You have three stacks of cylinders where each cylinder has the same diameter, but they may vary in height. You can change the height of a stack by removing and discarding its topmost cylinder any number of times.

Find the maximum possible height of the stacks such that all of the stacks are exactly the same height. This means you must remove zero or more cylinders from the top of zero or more of the three stacks until they are all the same height, then return the height.

Example

$h1 = [1, 2, 1, 1]$
$h2 = [1, 1, 2]$
$h3 = [1, 1]$

There are $\mathbf{4, 3}$ and $\mathbf{2}$ cylinders in the three stacks, with their heights in the three arrays. Remove the top 2 cylinders from $\boldsymbol{h1}$ (heights = [1, 2]) and from $\boldsymbol{h2}$ (heights = [1, 1]) so that the three stacks all are 2 units tall. Return $\mathbf{2}$ as the answer.

Note: An empty stack is still a stack.

Function Description

Complete the *equalStacks* function in the editor below.

equalStacks has the following parameters:

- *int h1[n1]:* the first array of heights
- *int h2[n2]:* the second array of heights
- *int h3[n3]:* the third array of heights

Returns

- *int:* the height of the stacks when they are equalized

Input Format

The first line contains three space-separated integers, $\boldsymbol{n1}$, $\boldsymbol{n2}$, and $\boldsymbol{n3}$, the numbers of cylinders in stacks $\mathbf{1}$, $\mathbf{2}$, and $\mathbf{3}$. The subsequent lines describe the respective heights of each cylinder in a stack *from top to bottom*:

- The second line contains $\boldsymbol{n1}$ space-separated integers, the cylinder heights in stack $\mathbf{1}$. The first element is the top cylinder of the stack.
- The third line contains $\boldsymbol{n2}$ space-separated integers, the cylinder heights in stack $\mathbf{2}$. The first element is the top cylinder of the stack.
- The fourth line contains $\boldsymbol{n3}$ space-separated integers, the cylinder heights in stack $\mathbf{3}$. The first element is the top cylinder of the stack.

Constraints

- $0 < n1, n2, n3 \leq 10^5$
- $0 < \ height\ of\ any\ cylinder\ \leq 100$

Sample Input

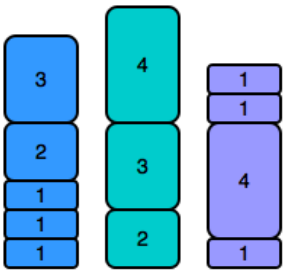
```
STDIN       Function
-----       --------
5 3 4       h1[] size n1 = 5, h2[] size n2 = 3, h3[] size n3 = 4
3 2 1 1 1   h1 = [3, 2, 1, 1, 1]
4 3 2       h2 = [4, 3, 2]
1 1 4 1     h3 = [1, 1, 4, 1]
```

Sample Output

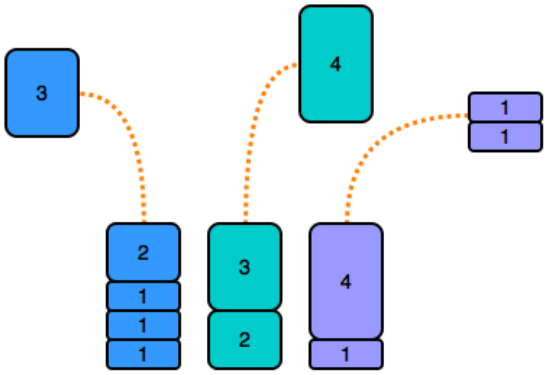
```
5
```

Explanation

Initially, the stacks look like this:



To equalize thier heights, remove the first cylinder from stacks $\mathbf{1}$ and $\mathbf{2}$, and then remove the top two cylinders from stack $\mathbf{3}$ (shown below).



The stack heights are reduced as follows:

1. $8 - 3 = 5$
2. $9 - 4 = 5$
3. $7 - 1 - 1 = 5$

All three stacks now have $\boldsymbol{height} = \mathbf{5}$, the value to return.

Contest ends in 5 days

Submissions: 7
Max Score: 25
Difficulty: Easy

Rate This Challenge:

More

Java 8

```
import java.io.*;
import java.math.*;
import java.security.*;
import java.text.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.function.*;
import java.util.regex.*;
import java.util.stream.*;
import static java.util.stream.Collectors.joining;
import static java.util.stream.Collectors.toList;

class Result {

    public static int equalStacks(List<Integer> h1, List<Integer> h2, List<Integer> h3) {
        int h1Sum=0;
        int h2Sum=0;
        int h3Sum=0;
        Stack<Integer> h1st =new Stack<>();
        Stack<Integer> h2st =new Stack<>();
        Stack<Integer> h3st =new Stack<>();
    for(int i=h1.size()-1;i>=0;i--){
        h1Sum+=h1.get(i);
        h1st.push(h1.get(i));
    }
    for(int i=h2.size()-1;i>=0;i--){
        h2Sum+=h2.get(i);
        h2st.push(h2.get(i));
    }
    for(int i=h3.size()-1;i>=0;i--){
        h3Sum+=h3.get(i);
        h3st.push(h3.get(i));
    }
    while(!(h1st.isEmpty()) || !(h2st.isEmpty()) || !(h3st.isEmpty())){
    if(h1Sum>h2Sum || h1Sum>h3Sum){
        int store = h1st.pop();
        h1Sum-=store;
    }
    else  if(h2Sum>h1Sum || h2Sum>h3Sum){
        int store = h2st.pop();
        h2Sum-=store;
    }
    else  if(h3Sum>h1Sum || h3Sum>h2Sum){
        int store = h3st.pop();
        h3Sum-=store;
    }
    else
      {
          return h1Sum;
      }

    }
    return 0;
  }
}

public class Solution {
    public static void main(String[] args) throws IOException {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
```

```
        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));

        String[] firstMultipleInput = bufferedReader.readLine().replaceAll("\\s+$", "").split("
");

        int n1 = Integer.parseInt(firstMultipleInput[0]);

        int n2 = Integer.parseInt(firstMultipleInput[1]);

        int n3 = Integer.parseInt(firstMultipleInput[2]);

        List<Integer> h1 = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split("
"))
            .map(Integer::parseInt)
            .collect(toList());

        List<Integer> h2 = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split("
"))
            .map(Integer::parseInt)
            .collect(toList());

        List<Integer> h3 = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split("
"))
            .map(Integer::parseInt)
            .collect(toList());

        int result = Result.equalStacks(h1, h2, h3);

        bufferedWriter.write(String.valueOf(result));
        bufferedWriter.newLine();

        bufferedReader.close();
        bufferedWriter.close();
    }
}

```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Find minimum element from the Stack

Given N elements and your task is to Implement a Stack in which you can get a minimum element in O(1) time.

Describtion:

You are required to complete the three methods push() which takes one argument an integer 'x' to be pushed into the stack, pop() which returns an integer popped out from the stack, and getMin() which returns the min element from the stack. (-1 will be returned if for pop() and getMin() the stack is empty.)

Expected Time Complexity: O(1) for all the 3 methods. Expected Auxiliary Space: O(1) for all the 3 methods.

Input Format

Push : operation to insert an element Pop : operation to delete an element

Constraints

1 <= Number of queries <= 100 1 <= values of the stack <= 100

Output Format

Minimum element from the stack

Sample Input 0

```
push(2)
push(3)
pop()
getMin()
push(1)
getMin()
```

Sample Output 0

```
2
1
```

Explanation 0

Explanation: In the first test case for query push(2) Insert 2 into the stack. The stack will be {2} push(3) Insert 3 into the stack. The stack will be {2 3} pop() Remove top element from stack Poped element will be 3 the stack will be {2} getMin() Return the minimum element min element will be 2 push(1) Insert 1 into the stack. The stack will be {2 1} getMin() Return the minimum element min element will be 1

Contest ends in 5 days

Submissions: 14
Max Score: 10
Difficulty: Medium

Rate This Challenge:

More

Java 8

```java
import java.util.*;
class StackDemo{
    static final int MAX = 100;
    int top;
    int a[] = new int[MAX];

    StackDemo(){
        top = -1;
    }

    int push(int x){
        if(top >= (MAX-1)){
            return 0;
        }
        else{
            a[++top] = x;
            return x;
        }
    }

    int pop(){
        if(top < 0){
            return 0;
        }
        else{
            int y = a[--top];
            return y;
        }
    }
    void getMin(){
        for(int i = 0; i<=top; i++){
            for(int j = i+1; j<=top ; j++){
                if(a[i]>a[j]){
                    int temp = a[i];
                    a[i]=a[j];
                    a[j]=temp;
                }
            }
        }
        System.out.println(a[0]);
    }
}
class Solution{
    public static void main(String args[]){
        StackDemo s1 = new StackDemo();
        s1.push(2);
        s1.push(3);
        s1.pop();
        s1.getMin();
        s1.push(1);
        s1.getMin();
    }
```

```
53 }
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Inserting a Node Into a Sorted Doubly Linked List

Given a reference to the head of a doubly-linked list and an integer, $data$, create a new *DoublyLinkedListNode* object having data value $data$ and insert it at the proper location to maintain the sort.

Example

$head$ refers to the list $1 \leftrightarrow 2 \leftrightarrow 4 \rightarrow NULL$
$data = 3$

Return a reference to the new list: $1 \leftrightarrow 2 \leftrightarrow 3 \leftrightarrow 4 \rightarrow NULL$.

Function Description

Complete the *sortedInsert* function in the editor below.

sortedInsert has two parameters:

- *DoublyLinkedListNode pointer head*: a reference to the head of a doubly-linked list
- *int data*: An integer denoting the value of the $data$ field for the *DoublyLinkedListNode* you must insert into the list.

Returns

- *DoublyLinkedListNode pointer:* a reference to the head of the list

Note: Recall that an empty list (i.e., where $head = \texttt{NULL}$) and a list with one element *are* sorted lists.

Input Format

The first line contains an integer t, the number of test cases.

Each of the test case is in the following format:

- The first line contains an integer n, the number of elements in the linked list.
- Each of the next n lines contains an integer, the *data* for each node of the linked list.
- The last line contains an integer, $data$, which needs to be inserted into the sorted doubly-linked list.

Constraints

- $1 \leq t \leq 10$
- $1 \leq n \leq 1000$
- $1 \leq DoublyLinkedListNode.data \leq 1000$

Sample Input

```
STDIN   Function
-----   --------
1       t = 1
4       n = 4
1       node data values = 1, 3, 4, 10
3
4
10
5       data = 5
```

Sample Output

```
1 3 4 5 10
```

Explanation

The initial doubly linked list is: $1 \leftrightarrow 3 \leftrightarrow 4 \leftrightarrow 10 \rightarrow NULL$.

The doubly linked list after insertion is: $1 \leftrightarrow 3 \leftrightarrow 4 \leftrightarrow 5 \leftrightarrow 10 \rightarrow NULL$

Contest ends in 5 days

Submissions: 24
Max Score: 10
Difficulty: Easy

Rate This Challenge:

More

Java 8

```
import ⇔;

public class Solution {

    static class DoublyLinkedListNode {
        public int data;
        public DoublyLinkedListNode next;
        public DoublyLinkedListNode prev;

        public DoublyLinkedListNode(int nodeData) {
            this.data = nodeData;
            this.next = null;
            this.prev = null;
        }
    }

    static class DoublyLinkedList {
        public DoublyLinkedListNode head;
        public DoublyLinkedListNode tail;

        public DoublyLinkedList() {
            this.head = null;
            this.tail = null;
        }

        public void insertNode(int nodeData) {
            DoublyLinkedListNode node = new DoublyLinkedListNode(nodeData);

            if (this.head == null) {
```

```
                this.head = node;
            } else {
                this.tail.next = node;
                node.prev = this.tail;
            }

            this.tail = node;
        }
    }

    public static void printDoublyLinkedList(DoublyLinkedListNode node, String sep,
BufferedWriter bufferedWriter) throws IOException {
        while (node != null) {
            bufferedWriter.write(String.valueOf(node.data));

            node = node.next;

            if (node != null) {
                bufferedWriter.write(sep);
            }
        }
    }
```

```
public static DoublyLinkedListNode sortedInsert(DoublyLinkedListNode llist, int data) {

    DoublyLinkedListNode newNode = new DoublyLinkedListNode(data);
    DoublyLinkedListNode cur = llist;

    while(cur != null){
        if(cur.data>data){
            if(cur.prev == null){
                cur.prev = newNode;
                newNode.next = cur;
                llist = newNode;
            }
            else{
                cur.prev.next = newNode;
                newNode.prev = cur.prev;
                newNode.next = cur;
                cur.prev = newNode;
            }
        return llist;
        }
        if(cur.next == null){
            cur.next  = newNode;
            newNode.prev = cur;
            newNode.next = null;
        }
        cur = cur.next;
    }
    return llist;
    }
```

```
    private static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) throws IOException {
        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));

        int t = scanner.nextInt();
        scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

        for (int tItr = 0; tItr < t; tItr++) {
            DoublyLinkedList llist = new DoublyLinkedList();

            int llistCount = scanner.nextInt();
            scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");
```

```
99
100             for (int i = 0; i < llistCount; i++) {
101                 int llistItem = scanner.nextInt();
102                 scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");
103
104                 llist.insertNode(llistItem);
105             }
106
107             int data = scanner.nextInt();
108             scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");
109
110             DoublyLinkedListNode llist1 = sortedInsert(llist.head, data);
111
112             printDoublyLinkedList(llist1, " ", bufferedWriter);
113             bufferedWriter.newLine();
114         }
115
116         bufferedWriter.close();
117
118         scanner.close();
119     }
120 }
121
```

Line: 85 Col: 6

Upload Code as File Test against custom input Run Code Submit Code

Delete a Node 3

You're given the pointer to the head node of a linked list and the position of a node to delete. Delete the node at the given position and return the head node. A position of 0 indicates head, a position of 1 indicates one node away from the head and so on. The list may become empty after you delete the node.

Input Format

You have to complete the Node* Delete(Node* head, int position) method which takes two arguments - the head of the linked list and the position of the node to delete. You should NOT read any input from stdin/console. position will always be at least 0 and less than the number of the elements in the list

Constraints

- $1 \leq n \leq 1000$
- $1 \leq list_i \leq 1000$, where $list_i$ is the i^{th} element of the linked list.

Output Format

Delete the node at the given position and return the head of the updated linked list. Do NOT print anything to stdout/console.

The code in the editor will print the updated linked list in a single line separated by spaces.

Sample Input 0

```
8
20
6
2
19
7
4
15
9
3
```

Sample Output 0

```
20 6 2 7 4 15 9
```

Explanation 0

The given linked list is 20->6->2->19->7->4->15->9. We have to delete the node at position 3, which is 19. After deleting that node, the updated linked list is: 20->6->2->7->4->15->9

Submissions: 69
Max Score: 10
Difficulty: Medium

Rate This Challenge:

More

Java 8

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

public class Solution {

    public static class LinkedList{

        Node head;

        LinkedList(){

            head = null;

        }

        public static class Node {

            int data;
            Node next;

            Node(int new_data){
                data = new_data;
                next = null;
            }
        }

        public void displayList (){

            Node current = head;
            do {
                System.out.print(current.data + " ");
                current = current.next;
            } while (current != null);
        }

        public void insert(int new_data){
            Node new_node = new Node (new_data);
            if (head == null){
                head = new_node;
            } else {
                new_node.next = head;
                head = new_node;
            }
        }

        public void append(int new_data){
            Node new_node = new Node (new_data);
            if (head == null){
                head = new_node;
            } else {
                Node current = head;
                while (current.next != null){
                    current = current.next;
                }
```

```
                current.next = new_node;
            }
        }

        public void insertAfter(Node prev, int new_data){
            if (head == null){
                System.out.println("List is empty");
                return;
            }

            if (prev == null){
                System.out.println("Node doesn't exist");
                return;
            }

            if (prev.next==null){
                append(new_data);
            } else {
                Node new_node = new Node(new_data);
                new_node = prev.next;
                prev.next = new_node;
            }
        }

        public Node deleteNode(Node head, int position){

            if (head == null){
                System.out.println("List is empty");
            } else {
                Node current = head, previous = null;
                for (int i = 0; i < position; i++){
                    previous = current;
                    current = current.next;
                }
                previous.next = current.next;
            }
            return head;
        }



    }

    public static void main(String[] args) {
        /* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution. */
        LinkedList linkdLi1 = new LinkedList();

        Scanner sc = new Scanner(System.in);
        //System.out.print("Enter Number of nodes to create");
        int n = sc.nextInt();
        for (int i = 0; i<n; i++){
            //System.out.print("Enter data for each node");
            int data = sc.nextInt();
            linkdLi1.append(data);
        }

        int position = sc.nextInt();

        //linkdLi1.displayList();

        linkdLi1.deleteNode(linkdLi1.head, position);

        //System.out.println("After deleting node");
        linkdLi1.displayList();

```

```
120         }
121 }
122
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Delete duplicate-value nodes from a sorted linked list

This challenge is part of a tutorial track by MyCodeSchool

You are given the pointer to the head node of a sorted linked list, where the data in the nodes is in ascending order. Delete nodes and return a sorted list with each distinct value in the original list. The given head pointer may be null indicating that the list is empty.

Example

$head$ refers to the first node in the list $1 \rightarrow 2 \rightarrow 2 \rightarrow 3 \rightarrow 3 \rightarrow 3 \rightarrow 3 \rightarrow NULL$.

Remove 1 of the 2 data values and return $head$ pointing to the revised list $1 \rightarrow 2 \rightarrow 3 \rightarrow NULL$.

Function Description

Complete the *removeDuplicates* function in the editor below.

removeDuplicates has the following parameter:

- *SinglyLinkedListNode pointer head:* a reference to the head of the list

Returns

- *SinglyLinkedListNode pointer:* a reference to the head of the revised list

Input Format

The first line contains an integer t, the number of test cases.

The format for each test case is as follows:

The first line contains an integer n, the number of elements in the linked list.
Each of the next n lines contains an integer, the $data$ value for each of the elements of the linked list.

Constraints

- $1 \leq t \leq 10$
- $1 \leq n \leq 1000$
- $1 \leq list[i] \leq 1000$

Sample Input

```
STDIN   Function
-----   --------
1       t = 1
```

```
5
1
2
2
3
4
```

n = 5
data values = 1, 2, 2, 3, 4

Sample Output

```
1 2 3 4
```

Explanation

The initial linked list is: $1 \rightarrow 2 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow NULL$.

The final linked list is: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow NULL$.


```java
import ⇔;

public class Solution {

    static class SinglyLinkedListNode {
        public int data;
        public SinglyLinkedListNode next;

        public SinglyLinkedListNode(int nodeData) {
            this.data = nodeData;
            this.next = null;
        }
    }

    static class SinglyLinkedList {
        public SinglyLinkedListNode head;
        public SinglyLinkedListNode tail;

        public SinglyLinkedList() {
            this.head = null;
            this.tail = null;
        }

        public void insertNode(int nodeData) {
            SinglyLinkedListNode node = new SinglyLinkedListNode(nodeData);

            if (this.head == null) {
                this.head = node;
            } else {
                this.tail.next = node;
            }

            this.tail = node;
```

```
        }
    }

    public static void printSinglyLinkedList(SinglyLinkedListNode node, String sep,
BufferedWriter bufferedWriter) throws IOException {
        while (node != null) {
            bufferedWriter.write(String.valueOf(node.data));

            node = node.next;

            if (node != null) {
                bufferedWriter.write(sep);
            }
        }
    }
```

```
public static SinglyLinkedListNode removeDuplicates(SinglyLinkedListNode llist) {
    SinglyLinkedListNode temp = llist;

        if (temp.next == null){
            return temp;
        }

        while (temp.next != null){
            if (temp.data == temp.next.data){
                temp.next = temp.next.next;
            } else {
                temp = temp.next;
            }
        }
        return llist;

    }
```

```
    private static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) throws IOException {
        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));

        int t = scanner.nextInt();
        scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

        for (int tItr = 0; tItr < t; tItr++) {
            SinglyLinkedList llist = new SinglyLinkedList();

            int llistCount = scanner.nextInt();
            scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

            for (int i = 0; i < llistCount; i++) {
                int llistItem = scanner.nextInt();
                scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

                llist.insertNode(llistItem);
            }

            SinglyLinkedListNode llist1 = removeDuplicates(llist.head);

            printSinglyLinkedList(llist1, " ", bufferedWriter);
            bufferedWriter.newLine();
        }

        bufferedWriter.close();

        scanner.close();
    }
}
```

103

Line: 19 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Compare two linked lists

This challenge is part of a tutorial track by MyCodeSchool

You're given the pointer to the head nodes of two linked lists. Compare the data in the nodes of the linked lists to check if they are equal. If all data attributes are equal and the lists are the same length, return 1. Otherwise, return 0.

Example

$llist1 = 1 \rightarrow 2 \rightarrow 3 \rightarrow NULL$
$llist2 = 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow NULL$

The two lists have equal data attributes for the first 3 nodes. $llist2$ is longer, though, so the lists are not equal. Return 0.

Function Description

Complete the *compare_lists* function in the editor below.

compare_lists has the following parameters:

- *SinglyLinkedListNode llist1:* a reference to the head of a list
- *SinglyLinkedListNode llist2:* a reference to the head of a list

Returns

- *int:* return 1 if the lists are equal, or 0 otherwise

Input Format

The first line contains an integer t, the number of test cases.

Each of the test cases has the following format:
The first line contains an integer n, the number of nodes in the first linked list.
Each of the next n lines contains an integer, each a value for a data attribute.
The next line contains an integer m, the number of nodes in the second linked list.
Each of the next m lines contains an integer, each a value for a data attribute.

Constraints

- $1 \leq t \leq 10$
- $1 \leq n, m \leq 1000$
- $1 \leq llist1[i], llist2[i] \leq 1000$

Output Format

Compare the two linked lists and `return 1` if the lists are equal. Otherwise, `return 0`. Do NOT print anything to stdout/console.

The output is handled by the code in the editor and it is as follows:

For each test case, in a new line, print $\mathbf{1}$ if the two lists are equal, else print $\mathbf{0}$.

Sample Input

```
2
2
1
2
1
1
2
1
2
2
1
2
```

Sample Output

```
0
1
```

Explanation

There are $t = 2$ test cases, each with a pair of linked lists.

- In the first case, linked lists are: 1 -> 2 -> NULL and 1 -> NULL
- In the second case, linked lists are: 1 -> 2 -> NULL and 1 -> 2 -> NULL


```
import ⇔;

public class Solution {

    static class SinglyLinkedListNode {
        public int data;
        public SinglyLinkedListNode next;

        public SinglyLinkedListNode(int nodeData) {
            this.data = nodeData;
            this.next = null;
        }
    }

    static class SinglyLinkedList {
        public SinglyLinkedListNode head;
        public SinglyLinkedListNode tail;

        public SinglyLinkedList() {
            this.head = null;
```

```
            this.tail = null;
        }

        public void insertNode(int nodeData) {
            SinglyLinkedListNode node = new SinglyLinkedListNode(nodeData);

            if (this.head == null) {
                this.head = node;
            } else {
                this.tail.next = node;
            }

            this.tail = node;
        }
    }

    public static void printSinglyLinkedList(SinglyLinkedListNode node, String sep,
BufferedWriter bufferedWriter) throws IOException {
        while (node != null) {
            bufferedWriter.write(String.valueOf(node.data));

            node = node.next;

            if (node != null) {
                bufferedWriter.write(sep);
            }
        }
    }
```

```
static boolean compareLists(SinglyLinkedListNode head1, SinglyLinkedListNode head2) {

        if (head1 == null && head2 == null) {
            return true;
        } else if (head1 == null || head2 == null) {
            return false;
        } else if (head1.data != head2.data) {
            return false;
        }
        return compareLists(head1.next,head2.next);

    }
```

```
    private static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) throws IOException {
        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));

        int tests = scanner.nextInt();
        scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

        for (int testsItr = 0; testsItr < tests; testsItr++) {
            SinglyLinkedList llist1 = new SinglyLinkedList();

            int llist1Count = scanner.nextInt();
            scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

            for (int i = 0; i < llist1Count; i++) {
                int llist1Item = scanner.nextInt();
                scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

                llist1.insertNode(llist1Item);
            }

            SinglyLinkedList llist2 = new SinglyLinkedList();
```

```
            int llist2Count = scanner.nextInt();
            scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

            for (int i = 0; i < llist2Count; i++) {
                int llist2Item = scanner.nextInt();
                scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

                llist2.insertNode(llist2Item);
            }

            boolean result = compareLists(llist1.head, llist2.head);

            bufferedWriter.write(String.valueOf(result ? 1 : 0));
            bufferedWriter.newLine();
        }

        bufferedWriter.close();

        scanner.close();
    }
}
```

Line: 14 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Append and Delete

You have two strings of lowercase English letters. You can perform two types of operations on the first string:

1. *Append* a lowercase English letter to the end of the string.
2. *Delete* the last character of the string. Performing this operation on an empty string results in an empty string.

Given an integer, k, and two strings, s and t, determine whether or not you can convert s to t by performing *exactly* k of the above operations on s. If it's possible, print `Yes`. Otherwise, print `No`.

Example. $s = [a, b, c]$
$t = [d, e, f]$
$k = 6$

To convert s to t, we first delete all of the characters in 3 moves. Next we add each of the characters of t in order. On the 6^{th} move, you will have the matching string. Return `Yes`.

If there were more moves available, they could have been eliminated by performing multiple deletions on an empty string. If there were fewer than 6 moves, we would not have succeeded in creating the new string.

Function Description

Complete the *appendAndDelete* function in the editor below. It should return a string, either `Yes` or `No`.

appendAndDelete has the following parameter(s):

- *string s*: the initial string
- *string t*: the desired string
- *int k*: the exact number of operations that must be performed

Returns

- *string:* either `Yes` or `No`

Input Format

The first line contains a string s, the initial string.
The second line contains a string t, the desired final string.
The third line contains an integer k, the number of operations.

Constraints

- $1 \leq |s| \leq 100$
- $1 \leq |t| \leq 100$
- $1 \leq k \leq 100$

- s and t consist of lowercase English letters, $\mathbf{ascii}[\mathbf{a\text{-}z}]$.

Sample Input 0

```
hackerhappy
hackerrank
9
```

Sample Output 0

```
Yes
```

Explanation 0

We perform 5 delete operations to reduce string s to `hacker`. Next, we perform 4 append operations (i.e., `r`, `a`, `n`, and `k`), to get `hackerrank`. Because we were able to convert s to t by performing exactly $k = 9$ operations, we return `Yes`.

Sample Input 1

```
aba
aba
7
```

Sample Output 1

```
Yes
```

Explanation 1

We perform 4 delete operations to reduce string s to the empty string. Recall that though the string will be empty after 3 deletions, we can still perform a delete operation on an empty string to get the empty string. Next, we perform 3 append operations (i.e., `a`, `b`, and `a`). Because we were able to convert s to t by performing exactly $k = 7$ operations, we return `Yes`.

Sample Input 2

```
ashley
ash
2
```

Sample Output 2

```
No
```

Explanation 2

To convert `ashley` to `ash` a minimum of 3 steps are needed. Hence we print `No` as answer.

Java 8

```java
import java.io.*;
import java.math.*;
import java.security.*;
import java.text.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.function.*;
import java.util.regex.*;
import java.util.stream.*;
import static java.util.stream.Collectors.joining;
import static java.util.stream.Collectors.toList;

class Result {

        public static String appendAndDelete(String s, String t, int k) {
           int sLen=s.length(),tLen=t.length();
           int remainLen;
           int n=Math.min(sLen,tLen);

           if( (sLen+tLen) <=k) return "Yes";

           for(int i=0;i<n;i++){
             if(s.charAt(i)==t.charAt(i)){
               sLen--;
               tLen--;
               }
             else
               break;
             }

             remainLen=sLen+tLen;

            if(  remainLen <=k &&
            (( remainLen%2==0 && k%2==0)||( remainLen%2!=0 && k%2!=0)) )
              return "Yes";
             return "No";
   }

}


public class Solution {
    public static void main(String[] args) throws IOException {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));

        String s = bufferedReader.readLine();

        String t = bufferedReader.readLine();

        int k = Integer.parseInt(bufferedReader.readLine().trim());

        String result = Result.appendAndDelete(s, t, k);

        bufferedWriter.write(result);
        bufferedWriter.newLine();

        bufferedReader.close();
        bufferedWriter.close();
    }
}
```

62

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Reverse a doubly linked list

This challenge is part of a tutorial track by MyCodeSchool

Given the pointer to the head node of a doubly linked list, reverse the order of the nodes in place. That is, change the *next* and *prev* pointers of the nodes so that the direction of the list is reversed. Return a reference to the head node of the reversed list.

Note: The head node might be NULL to indicate that the list is empty.

Function Description

Complete the *reverse* function in the editor below.

reverse has the following parameter(s):

- *DoublyLinkedListNode head*: a reference to the head of a DoublyLinkedList

Returns
- *DoublyLinkedListNode*: a reference to the head of the reversed list

Input Format

The first line contains an integer t, the number of test cases.

Each test case is of the following format:

- The first line contains an integer n, the number of elements in the linked list.
- The next n lines contain an integer each denoting an element of the linked list.

Constraints

- $1 \leq t \leq 10$
- $0 \leq n \leq 1000$
- $0 \leq DoublyLinkedListNode.data \leq 1000$

Output Format

Return a reference to the head of your reversed list. The provided code will print the reverse array as a one line of space-separated integers for each test case.

Sample Input

```
1
4
1
2
```

```
3
4
```

Sample Output

```
4 3 2 1
```

Explanation

The initial doubly linked list is: $\mathbf{1} \leftrightarrow \mathbf{2} \leftrightarrow \mathbf{3} \leftrightarrow \mathbf{4} \rightarrow \boldsymbol{NULL}$

The reversed doubly linked list is: $\mathbf{4} \leftrightarrow \mathbf{3} \leftrightarrow \mathbf{2} \leftrightarrow \mathbf{1} \rightarrow \boldsymbol{NULL}$

Java 8

```java
import ⇔;

public class Solution {

    static class DoublyLinkedListNode {
        public int data;
        public DoublyLinkedListNode next;
        public DoublyLinkedListNode prev;

        public DoublyLinkedListNode(int nodeData) {
            this.data = nodeData;
            this.next = null;
            this.prev = null;
        }
    }

    static class DoublyLinkedList {
        public DoublyLinkedListNode head;
        public DoublyLinkedListNode tail;

        public DoublyLinkedList() {
            this.head = null;
            this.tail = null;
        }

        public void insertNode(int nodeData) {
            DoublyLinkedListNode node = new DoublyLinkedListNode(nodeData);

            if (this.head == null) {
                this.head = node;
            } else {
                this.tail.next = node;
                node.prev = this.tail;
            }

            this.tail = node;
```

```
        }
    }

    public static void printDoublyLinkedList(DoublyLinkedListNode node, String sep,
BufferedWriter bufferedWriter) throws IOException {
        while (node != null) {
            bufferedWriter.write(String.valueOf(node.data));

            node = node.next;

            if (node != null) {
                bufferedWriter.write(sep);
            }
        }
    }


    public static DoublyLinkedListNode reverse(DoublyLinkedListNode llist) {
    DoublyLinkedListNode current = llist;
    DoublyLinkedListNode prev = null;
    DoublyLinkedListNode next;

    while (current != null) {
        next = current.next;
        current.next = prev;
        current.prev = next;
        prev = current;
        current = next;
    }

    return prev;
}


    private static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) throws IOException {
        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));

        int t = scanner.nextInt();
        scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

        for (int tItr = 0; tItr < t; tItr++) {
            DoublyLinkedList llist = new DoublyLinkedList();

            int llistCount = scanner.nextInt();
            scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

            for (int i = 0; i < llistCount; i++) {
                int llistItem = scanner.nextInt();
                scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

                llist.insertNode(llistItem);
            }

            DoublyLinkedListNode llist1 = reverse(llist.head);

            printDoublyLinkedList(llist1, " ", bufferedWriter);
            bufferedWriter.newLine();
        }

        bufferedWriter.close();

        scanner.close();
    }
}
```

106

Line: 19 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

H3: Find the Number of Nodes in a Binary Search Tree

This section discusses the recursive algorithm which counts the size or total number of nodes in a Binary Search Tree.

Total No. of nodes=Total No. of nodes in left sub-tree + Total no. of node in right sub-tree + 1

Algorithm:

CountNodes(node x) set n=1 //global variable If x=NULL return 0 [End If] If(x->left!=NULL) n=n+1 CountNode(x->left) [End If] If(x->right!=NULL) n=n+1 CountNode(x->right) [End If] Return n

Input Format

3 4 2 5 1

Constraints

0

Output Format

Total No. of Nodes in the BST = 50

Sample Input 0

```
3
4
2
5
1
```

Sample Output 0

```
Total No. of Nodes in the BST = 5
```

Java 8

```java
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {
    public static void main(String args[] ) throws Exception {
         Node root = newNode(3);
        root.left = newNode(4);
        root.right = newNode(2);
        root.left.left = newNode(5);
        root.left.right = newNode(1);

System.out.print("Total No. of Nodes in the BST = ");
        System.out.println(countNodes(root));

    }
    static class Node
    {
        public int data;
        public Node left, right;
        public Node(){
            data = 0;
            left = right = null;
        }

    }
    static int countNodes(Node root)
    {
        if (root == null)
            return 0;

        int l = countNodes(root.left);
        int r = countNodes(root.right);

        return 1 + l + r;
    }
     static Node newNode(int data)
    {
        Node temp = new Node();
        temp.data = data;
        temp.left = temp.right = null;
        return temp;
    }
}

```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

H2: Reverse a single linked list

Problem statement: Given a linked list reverse it without using any additional space (In O(1) space complexity).

Solution:

Reversing a single linked list is about reversing the linking direction. We can solve the above problem by following steps:

1. Building the linked list

Firstly, we build the linked list by inserting each node at the beginning. For detailed analysis of how to build a linked list using insertion at beginning, kindly go through this full article: Single linked list insertion

2. Function to reverse the link list

As told previously, the basic idea to reverse a linked list is to reverse the direction of linking. This concept can be implemented without using any additional space. We need three pointers *prev, *cur, *next to implement the function. These variables are named accordingly to indicate their serving part in the function.

*prev - to store the previous node which will become ultimately the next node after reversal for current node

*cur - to store the current node

*next - to store the next node which will become current node in the next iteration.

Initialize *prev & *next to NULL, *cur to head

while(cur!=NULL) Set *nextto cur->next Set cur->nextto *prev Set *prev to *cur Set *cur to *next End While loop Set head to *prev

3. Print the reversed linked list

Example:

Let the linked list be 1->2->3->4->5->NULL (for simplicity in understanding representing pointer to node by node value) Head is 1 Initialize: cur =1, prev=NULL, next=NULL

in iteration 1: next=2 cur->next=NULL prev=1 cur=2 thus reversed part: 1->NULL

in iteration 2: next=3 cur->next=1 prev=2 cur=3 thus reversed part: 2->1->NULL

in iteration 3: next=4 cur->next=2 prev=3 cur=4 thus reversed part: 3->2->1->NULL

in iteration 4: next=5 cur->next=3 prev=4 cur=5 thus reversed part: 4->3->2->1->NULL

in iteration 5: next=NULL cur->next=4 prev=5 cur=NULL thus reversed part: 5->4->3->2->1->NULL iteration ends at cur is NULL linked list reversed, head at 5

Input Format

creating the linked list by inserting new nodes at the begining enter 0 to stop building the list, else enter any integer 6 7 8 9 4 3 3 1 0

Constraints

data>0

Output Format

traversing the list 1 3 3 4 9 8 7 6 total no of nodes : 8 reversing the list.....

traversing the list 6 7 8 9 4 3 3 1 total no of nodes : 8

Java 7

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

public class Solution {

    static Node head;
    int count = 0;       // To keep track of the count of elements

    static class Node {
        int data;
        Node next;

        Node(int d) {
            data = d;
            next = null;
        }
    }


    public static int countNodes()
    {
            int count = 0;
             Node current = head;

             while (current != null) {
                count++;
             current = current.next;
            }

        return count;
    }

    void reverse() {
        Node prev = null;
        Node current = head;
        Node next = null;

        while (current != null) {
            next = current.next;
            current.next = prev;
```

```
            prev = current;
            current = next;
        }

        head = prev;
    }


    void display() {
        Node current = head;

        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
    }

void insertEnd(int new_data)
        {
            Node new_node = new Node(new_data);       //new node is created

            if(head == null)                                                //check list is empty ?
            {
                head = new_node;
                return;
            }
            new_node.next = null;

            Node last = head;
            while( last.next != null)
            {
                last = last.next;
            }
            last.next = new_node;
            return ;

        }

    public static void main(String[] args)
    {
        Solution list = new Solution();
        Scanner scanner = new Scanner(System.in);

        while (true) {
        //    System.out.print("Enter a number to insert (or 0 to quit): ");
            int data = scanner.nextInt();
            if (data == 0) {
                break;
            }
             list.insertEnd(data);
        }
        scanner.close();
        System.out.print("traversing the list ");
        list.display();
         int nodeCount = countNodes();
        System.out.print("total no of nodes : " + nodeCount);
        System.out.println(" reversing the list.....");
        list.reverse();
        System.out.print("\ntraversing the list ");
        list.display();
        System.out.println("total no of nodes : " + nodeCount);

    }
}
```

Line: 1 Col: 1

Upload Code as File | Test against custom input | Run Code | Submit Code

Your Implement a Queue using Linked List submission got 10.00 points. Share Tweet

Try the Next Challenge | Contest Leaderboard

Implement a Queue using Linked List

Problem Submissions Leaderboard Discussions

Implement a Queue using Linked List with the following operations.

1.Insert

2.Delete (Display "Queue is empty" if queue is empty).

3.Display (Display "NULL" if queue is empty).

4.Exit

Input Format

First line contains the Menu followed Input for given menu.

Constraints

NA

Output Format

Print the out put based on the Menu

Sample Input 0

```
1
10
1
20
1
30
3
2
3
4
```

Sample Output 0

```
->10->20->30
->20->30
```

Sample Input 1

Contest ends in 14 hours

Submissions: 23
Max Score: 10
Difficulty: Medium

Rate This Challenge:

More

```
1
10
1
20
1
30
3
2
3
2
3
2
3
2
3
4
```

Sample Output 1

```
->10->20->30
->20->30
->30
NULL
Queue is empty
NULL
```

Java 8

```java
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {
   public static void main(String[] args)
        {
            Scanner scanner = new Scanner(System.in);
            QueueLinkedList queue = new QueueLinkedList();

        while (true)
        {
            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    int item = scanner.nextInt();
                    queue.insert(item);
                    break;
                case 2:
                    queue.delete();
                    break;
                case 3:
                    queue.display();
                    break;
                case 4:
                    scanner.close();
                    System.exit(0);
                    break;
            }
        }
    }

}

class QueueLinkedList {
```

```
    private LinkedList<Integer> queue = new LinkedList<>();

    public void insert(int item)
    {
        queue.addLast(item);
    }
    public void delete()
    {
        if (queue.isEmpty())
        {
            System.out.println("Queue is empty");
        } else {
            queue.removeFirst();
        }
    }

    public void display()
    {
        if (queue.isEmpty()) {
            System.out.println("NULL");
        } else {
            for (int item : queue) {
                System.out.print("->" + item);
            }
            System.out.println();
        }
    }
}

```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Queue using Two Stacks

A queue is an abstract data type that maintains the order in which elements were added to it, allowing the oldest elements to be removed from the front and new elements to be added to the rear. This is called a *First-In-First-Out* (FIFO) data structure because the first element added to the queue (i.e., the one that has been waiting the longest) is always the first one to be removed.

A basic queue has the following operations:

- *Enqueue*: add a new element to the end of the queue.
- *Dequeue*: remove the element from the front of the queue and return it.

In this challenge, you must first implement a queue using *two stacks*. Then process q queries, where each query is one of the following 3 types:

1. `1 x` : Enqueue element x into the end of the queue.
2. `2` : Dequeue the element at the front of the queue.
3. `3` : Print the element at the front of the queue.

Input Format

The first line contains a single integer, q, denoting the number of queries.
Each line i of the q subsequent lines contains a single query in the form described in the problem statement above. All three queries start with an integer denoting the query $type$, but only query 1 is followed by an additional space-separated value, x, denoting the value to be enqueued.

Constraints

- $1 \leq q \leq 10^5$
- $1 \leq type \leq 3$
- $1 \leq |x| \leq 10^9$
- It is guaranteed that a valid answer always exists for each query of type 3.

Output Format

For each query of type 3, print the value of the element at the front of the queue on a new line.

Sample Input

```
STDIN   Function
-----   --------
10      q = 10 (number of queries)
1 42    1st query, enqueue 42
2       dequeue front element
1 14    enqueue 42
3       print the front element
```

```
1 28    enqueue 28
3       print the front element
1 60    enqueue 60
1 78    enqueue 78
2       dequeue front element
2       dequeue front element
```

Sample Output

```
14
14
```

Explanation

Perform the following sequence of actions:

1. Enqueue $\mathbf{42}$; $\boldsymbol{queue} = \{42\}$.
2. Dequeue the value at the head of the queue, $\mathbf{42}$; $\boldsymbol{queue} = \{\}$.
3. Enqueue $\mathbf{14}$; $\boldsymbol{queue} = \{14\}$.
4. Print the value at the head of the queue, $\mathbf{14}$; $\boldsymbol{queue} = \{14\}$.
5. Enqueue $\mathbf{28}$; $\boldsymbol{queue} = \{14, 28\}$.
6. Print the value at the head of the queue, $\mathbf{14}$; $\boldsymbol{queue} = \{14, 28\}$.
7. Enqueue $\mathbf{60}$; $\boldsymbol{queue} = \{14, 28, 60\}$.
8. Enqueue $\mathbf{78}$; $\boldsymbol{queue} = \{14, 28, 60, 78\}$.
9. Dequeue the value at the head of the queue, $\mathbf{14}$; $\boldsymbol{queue} = \{28, 60, 78\}$.
10. Dequeue the value at the head of the queue, $\mathbf{28}$; $\boldsymbol{queue} = \{60, 78\}$.


```java
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;


public class Solution {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        //System.out.println("Enter no of Queries : ");
        int q = scanner.nextInt();
        Stack<Integer> enqueueStack = new Stack<>();
```

```
        Stack<Integer> dequeueStack = new Stack<>();

        for (int i = 0; i < q; i++) {
            int queryType = scanner.nextInt();

            switch (queryType) {
                case 1:
                    //System.out.println("Enter an element To insert : ");
                    int element = scanner.nextInt();
                    enqueueStack.push(element);
                    break;

                case 2: // Dequeue
                    if (dequeueStack.isEmpty()) {
                        // Transfer elements from enqueue stack to dequeue stack
                        while (!enqueueStack.isEmpty()) {
                            dequeueStack.push(enqueueStack.pop());
                        }
                    }
                    if (!dequeueStack.isEmpty()) {
                        dequeueStack.pop();
                    }
                    break;

                case 3: // Print front element
                    if (dequeueStack.isEmpty()) {
                        // Transfer elements from enqueue stack to dequeue stack
                        while (!enqueueStack.isEmpty()) {
                            dequeueStack.push(enqueueStack.pop());
                        }
                    }
                    if (!dequeueStack.isEmpty()) {
                        System.out.println(dequeueStack.peek());
                    }
                    break;
            }
        }

        scanner.close();
    }
}
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

H5: Min Heap

Given a priority queue (max heap) and Q queries to be performed on priority queue. The task is to perform operations based on queries. 1. p : query to push element (x, given with query) to priority_queue and print size. 2. pp : query to pop element from priority_queue and print size. 3. t : query to return top element of priority_queue, if empty -1 will be printed.

Your Task: Your task is to write the functions push_pq(), pp_pq() and pq_top(), so that the queries are performed.

Input Format

First line of input contains number of testcases T. For each testcase, first line of input contains Q queries. Next Q lines contains Q queries.

Constraints

1 <= T <= 100 1 <= Q <= 100

Output Format

For each testcase, perform the required operation, and print if anything required.

Sample Input 0

```
Input:
1
5
p 5
p 3
p 1
t
pp
```

Sample Output 0

```
1
2
3
1
2
```

Explanation 0

Testcase 1: Pushing 5, 3, 1 to queue. Now, fetching top which is 1 (according to min heap property smallest element at top). Popping back element from queue, size reduced to 2.

Submissions: 8
Max Score: 15
Difficulty: Medium

Rate This Challenge:

More

Java 8

```java
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {
    static void heapify(int arr[],int n,int i){
        int p=(i-1)/2;
        if(p>=0){
            if(arr[p]>arr[i]){
                int temp=arr[p];
                arr[p]=arr[i];
                arr[i]=temp;
                heapify(arr, n, p);
            }
        }
    }
    static int  insert(int[] arr,int n,int k){
        n=n+1;
        arr[n-1]=k;
        heapify(arr, n, n-1);
        return n;

    }

    static void delheapify(int[] arr,int n,int i){
         int smallest=i;
         int left=2*i+1;
         int right=2*i+2;
         if(left<n && arr[left]<arr[smallest]){
            smallest=left;
         }
         if(right<n && arr[right]<arr[smallest]){
            smallest=right;
         }
         if(smallest!=i){
            int temp=arr[smallest];
            arr[smallest]=arr[i];
            arr[i]=temp;
            delheapify(arr, n, smallest);
         }

    }
    static int delete(int[] arr,int n,int p){
        int smallest=arr[n-1];
        int x=0;
        for(int i=0;i<n;i++){
            if(arr[i]==p){
                x=i;
                break;
            }
        }

```

```
        arr[x]=smallest;
        n=n-1;
        delheapify(arr, n, x);
        return n;
    }

    static void print(int[] arr,int n){
        for(int i=0;i<n;i++){
            System.out.print(arr[i]+" ");
        }
    }


    public static void main(String args[] ) throws Exception {
        int Max=100;
        Scanner sc=new Scanner(System.in);
        sc.nextLine();
        int t=sc.nextInt();

        while(t-->0){
             int n=0;
            int[] arr=new int[Max];
            int top=0;

            int q = sc.nextInt();
            for(int i=0;i<q;i++){
                String qry = sc.next();
                if(qry.equals("p")){
                    int p=sc.nextInt();
                    n=insert(arr, n, p);
                    System.out.println(n);
                }
                else if(qry.equals("pp")){
                    n=delete(arr, n, arr[0]);
                    System.out.println(n);
                }
                else if(qry.equals("t")){
                    top=arr[0];
                System.out.println(top);
                }
            }
        }
    }
}
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

H4: Check whether a Binary Tree is BST (Binary Search Tree) or not

Given a binary tree check whether it is a binary search tree or not.

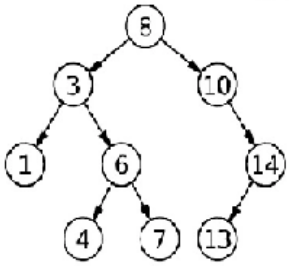
Algorithm:

Intuition says that the in-order traversal for the BST results in a sorted list of nodes and we use this in our algorithm.

- 1. Set prev to INT_MIN.
- 2. From main function call checkBST(root, prev)
- //passing prev by reference to update it every time
- checkBST(root, &prev)
- 3. if(root==NULL)
- return 1; //null tree is BST
- 4. do in-order traversal and checking whether all tree node
- data is sorted or not
- if(!(checkBST(root->left,prev))) //check left subtree
- return 0;
- //root->data must be greater than prevsince BST results in
- //sorted list after in-order traversal.
- 5. if(root->data
- return 0;
- 6. prev=root->data; //update prev value
- 7. return checkBST(root->right,prev);//check right subtree

Input Format

Tree1



Constraints

0

Output Format

Tree is built.

Contest ends in 14 hours

Submissions: 22
Max Score: 10
Difficulty: Medium

Rate This Challenge:

More

Java 8

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {

        Node root;
    int prev = Integer.MIN_VALUE;
    static class Node {
    int data;
    Node left, right;

    public Node(int data) {
        this.data = data;
        left = right = null;
    }
}
    public static void main(String args[] ) throws Exception {
        Solution T1 = new Solution();

        T1.root = new Node(8);
        T1.root.left = new Node(3);
        T1.root.right = new Node(10);
        T1.root.left.left = new Node(1);
        T1.root.left.right = new Node(6);
        T1.root.left.right.left = new Node(4);
        T1.root.left.right.right = new Node(7);
        T1.root.right.right = new Node(14);
        T1.root.right.right.left = new Node(13);

```

```
        if (T1.checkBST()) {
            System.out.println("The T1 is a Binary Search Tree.");
        } else {
            System.out.println("The T1 is not a Binary Search Tree.");
        }
    }

    boolean checkBST() {
        return checkBST(root);
    }

    boolean checkBST(Node node) {
        if (node == null) {
            return true;
        }
        if (!checkBST(node.left)) {
            return false;
        }
        if (node.data <= prev) {
            return false;
        }
        prev = node.data;

        return checkBST(node.right);
    }
    }

```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Insertion Sort - Part 1

Sorting

One common task for computers is to sort data. For example, people might want to see all their files on a computer sorted by size. Since sorting is a simple problem with many different possible solutions, it is often used to introduce the study of algorithms.

Insertion Sort

These challenges will cover *Insertion Sort*, a simple and intuitive sorting algorithm. We will first start with a nearly sorted list.

Insert element into sorted list

Given a sorted list with an unsorted number e in the rightmost cell, can you write some simple code to insert e into the array so that it remains sorted?

Since this is a learning exercise, it won't be the most efficient way of performing the insertion. It will instead demonstrate the brute-force method in detail.

Assume you are given the array $arr = [1, 2, 4, 5, 3]$ indexed $0 \ldots 4$. Store the value of $arr[4]$. Now test lower index values successively from 3 to 0 until you reach a value that is lower than $arr[4]$, at $arr[1]$ in this case. Each time your test fails, copy the value at the lower index to the current index and print your array. When the next lower indexed value is smaller than $arr[4]$, insert the stored value at the current index and print the entire array.

Example

$n = 5$

$arr = [1, 2, 4, 5, 3]$

Start at the rightmost index. Store the value of $arr[4] = 3$. Compare this to each element to the left until a smaller value is reached. Here are the results as described:

```
1 2 4 5 5
1 2 4 4 5
1 2 3 4 5
```

Function Description

Complete the *insertionSort1* function in the editor below.

insertionSort1 has the following parameter(s):

- *n*: an integer, the size of arr
- *arr*: an array of integers to sort

Returns

- *None:* Print the interim and final arrays, each on a new line. No return value is expected.

Input Format

The first line contains the integer n, the size of the array arr.
The next line contains n space-separated integers $arr[0]...arr[n-1]$.

Constraints

$1 \leq n \leq 1000$
$-10000 \leq arr[i] \leq 10000$

Output Format

Print the array as a row of space-separated integers each time there is a shift or insertion.

Sample Input

```
5
2 4 6 8 3
```

Sample Output

```
2 4 6 8 8
2 4 6 6 8
2 4 4 6 8
2 3 4 6 8
```

Explanation

3 is removed from the end of the array.
In the 1^{st} line $8 > 3$, so 8 is shifted one cell to the right.
In the 2^{nd} line $6 > 3$, so 6 is shifted one cell to the right.
In the 3^{rd} line $4 > 3$, so 4 is shifted one cell to the right.
In the 4^{th} line $2 < 3$, so 3 is placed at position 1.

Next Challenge

In the next Challenge, we will complete the insertion sort.

```
class Result {

    public static void insertionSort1(int n, List<Integer> arr) {
        for (int i = 1; i < n; i++) {
            int pos = arr.get(i);
            int j;
            for (j = i - 1; j >= 0 && arr.get(j) > pos; j--) {
                arr.set(j + 1, arr.get(j));
                System.out.println(display(arr));

            }
            arr.set(j + 1, pos);
        }
        System.out.println(display(arr));
    }

    private static String display(List<Integer> newArr) {
        int[] newArr1 = newArr.stream().mapToInt(i -> i).toArray();
        return Arrays.stream(newArr1)
                .mapToObj(j -> String.valueOf(j))
                .collect(Collectors.joining(" "));
    }

}

public class Solution {
    public static void main(String[] args) throws IOException {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

        int n = Integer.parseInt(bufferedReader.readLine().trim());

        List<Integer> arr = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split("
"))
            .map(Integer::parseInt)
            .collect(toList());

        Result.insertionSort1(n, arr);

        bufferedReader.close();
    }
}

```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Quicksort 1 - Partition

The previous challenges covered Insertion Sort, which is a simple and intuitive sorting algorithm with a running time of $O(n^2)$. In these next few challenges, we're covering a *divide-and-conquer* algorithm called Quicksort (also known as *Partition Sort*). This challenge is a modified version of the algorithm that only addresses partitioning. It is implemented as follows:

Step 1: Divide
Choose some pivot element, p, and partition your unsorted array, arr, into three smaller arrays: $left$, $right$, and $equal$, where each element in $left < p$, each element in $right > p$, and each element in $equal = p$.

Example
$arr = [5, 7, 4, 3, 8]$

In this challenge, the pivot will always be at $arr[0]$, so the pivot is 5.

arr is divided into $left = \{4, 3\}$, $equal = \{5\}$, and $right = \{7, 8\}$.
Putting them all together, you get $\{4, 3, 5, 7, 8\}$. There is a flexible checker that allows the elements of $left$ and $right$ to be in any order. For example, $\{3, 4, 5, 8, 7\}$ is valid as well.

Given arr and $p = arr[0]$, partition arr into $left$, $right$, and $equal$ using the *Divide* instructions above. Return a 1-dimensional array containing each element in $left$ first, followed by each element in $equal$, followed by each element in $right$.

Function Description

Complete the *quickSort* function in the editor below.

quickSort has the following parameter(s):

- *int arr[n]:* $arr[0]$ is the pivot element

Returns

- *int[n]:* an array of integers as described above

Input Format

The first line contains n, the size of arr.
The second line contains n space-separated integers $arr[i]$ (the unsorted array). The first integer, $arr[0]$, is the pivot element, p.

Constraints

- $1 \leq n \leq 1000$
- $-1000 \leq arr[i] \leq 1000$ where $0 \leq i < n$
- All elements are distinct.

Sample Input

```
STDIN       Function
-----       --------
5           arr[] size n =5
4 5 3 7 2   arr =[4, 5, 3, 7, 2]
```

Sample Output

```
3 2 4 5 7
```

Explanation

$arr = [4, 5, 3, 7, 2]$ *Pivot*: $p = arr[0] = 4$.
$left = \{\}$; $equal = \{4\}$; $right = \{\}$

$arr[1] = 5 > p$, so it is added to $right$.
$left = \{\}$; $equal = \{4\}$; $right = \{5\}$

$arr[2] = 3 < p$, so it is added to $left$.
$left = \{3\}$; $equal = \{4\}$; $right = \{5\}$

$arr[3] = 7 > p$, so it is added to $right$.
$left = \{3\}$; $equal = \{4\}$; $right = \{5, 7\}$

$arr[4] = 2 < p$, so it is added to $left$.
$left = \{3, 2\}$; $equal = \{4\}$; $right = \{5, 7\}$

Return the array $\{32457\}$.

The order of the elements to the left and right of 4 does not need to match this answer. It is only required that 3 and 2 are to the left of 4, and 5 and 7 are to the right.


```
1  import java.io.*;
2  import java.math.*;
3  import java.security.*;
4  import java.text.*;
5  import java.util.*;
6  import java.util.concurrent.*;
7  import java.util.function.*;
8  import java.util.regex.*;
9  import java.util.stream.*;
10 import static java.util.stream.Collectors.joining;
11 import static java.util.stream.Collectors.toList;
12
13 class Result {
14
15     public static List<Integer> quickSort(List<Integer> arr) {
16     Deque<Integer> deque = new ArrayDeque<>();
```

```
    int pivot = arr.get(0);

    // Iterate through arr to add pivot items
    for(int value : arr) {
        if(value == pivot) deque.addFirst(pivot);
    }

    // Iterate through arr to devide left and right
    for(int value : arr) {
        if(value < pivot) deque.addFirst(value);
        if(value > pivot) deque.addLast(value);
    }

    return new ArrayList<>(deque);
}

}

public class Solution {
    public static void main(String[] args) throws IOException {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));

        int n = Integer.parseInt(bufferedReader.readLine().trim());

        List<Integer> arr = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split("
"))
            .map(Integer::parseInt)
            .collect(toList());

        List<Integer> result = Result.quickSort(arr);

        bufferedWriter.write(
            result.stream()
                .map(Object::toString)
                .collect(joining(" "))
            + "\n"
        );

        bufferedReader.close();
        bufferedWriter.close();
    }
}

```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Merge List

Shashank is very excited after learning about the *linked list*. He learned about how to *merge* two linked lists. When we merge two linked lists, the order of the elements of each list doesn't change. For example, if we merge $[1, 2, 3]$ and $[4, 5, 6]$, $[1, 4, 2, 3, 5, 6]$ is a valid merge, while $[1, 4, 3, 2, 5, 6]$ is not a valid merge because 3 appears before 2.

Shashank wants you to solve a problem for him: You are given two lists having sizes N and M. How many ways can we merge both the lists? It is given that all $N + M$ elements are distinct. As your answer can be quite large, Shashank wants you to print it $\mod 10^9 + 7$.

Input Format

The first line contains an integer T, the number of test cases.
Each of the next T lines contains two integers N and M.

Constraints

- $1 \leq T \leq 10$
- $1 \leq N \leq 100$
- $1 \leq M \leq 100$

Output Format

Print the value of the answer $\mod 10^9 + 7$.

Sample Input 0

```
1
2 2
```

Sample Output 0

```
6
```

Explanation 0

Suppose the two lists are $[1, 2]$ and $[3, 4]$. The different ways of merging these lists are given below:

$[1, 2, 3, 4]$
$[1, 3, 2, 4]$
$[3, 4, 1, 2]$
$[3, 1, 4, 2]$
$[1, 3, 4, 2]$
$[3, 1, 2, 4]$

Contest ends in 12 days

Submissions: 3
Max Score: 40
Difficulty: Medium

Rate This Challenge:

More

Java 7

```
import java.io.*;
import java.math.*;
import java.security.*;
import java.text.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.regex.*;

class Result {

   public static int solve(int n, int m) {
        int MOD = 1000000007; // Modulus to ensure the result doesn't overflow

        // Create a 2D array to store the number of ways to merge lists of size i and j
        int[][] dp = new int[n + 1][m + 1];

        // Initialize the base cases
        for (int i = 0; i <= n; i++) {
            dp[i][0] = 1;
        }
        for (int j = 0; j <= m; j++) {
            dp[0][j] = 1;
        }

        // Calculate the number of ways using dynamic programming
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                dp[i][j] = (dp[i - 1][j] + dp[i][j - 1]) % MOD;
            }
        }

        return dp[n][m];
    }

}

public class Solution {
    public static void main(String[] args) throws IOException {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));

        int t = Integer.parseInt(bufferedReader.readLine().trim());

        for (int tItr = 0; tItr < t; tItr++) {
            String[] firstMultipleInput = bufferedReader.readLine().replaceAll("\\s+$",
"").split(" ");

            int n = Integer.parseInt(firstMultipleInput[0]);

            int m = Integer.parseInt(firstMultipleInput[1]);
```

```
50
51              int result = Result.solve(n, m);
52
53              bufferedWriter.write(String.valueOf(result));
54              bufferedWriter.newLine();
55          }
56
57          bufferedReader.close();
58          bufferedWriter.close();
59      }
60  }
61
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Is This a Binary Search Tree?

For the purposes of this challenge, we define a binary tree to be a binary search tree with the following ordering requirements:

- The $data$ value of every node in a node's left subtree is *less than* the data value of that node.
- The $data$ value of every node in a node's right subtree is *greater than* the data value of that node.

Given the root node of a binary tree, can you determine if it's also a binary search tree?

Complete the function in your editor below, which has 1 parameter: a pointer to the root of a binary tree. It must return a *boolean* denoting whether or not the binary tree is a binary search tree. You may have to write one or more helper functions to complete this challenge.

Input Format

You are not responsible for reading any input from stdin. Hidden code stubs will assemble a binary tree and pass its root node to your function as an argument.

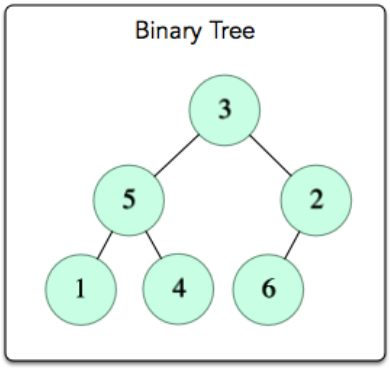
Constraints

- $0 \leq data \leq 10^4$

Output Format

You are not responsible for printing any output to stdout. Your function must return *true* if the tree is a binary search tree; otherwise, it must return *false*. Hidden code stubs will print this result as a *Yes* or *No* answer on a new line.

Sample Input



Sample Output

No

Contest ends in 12 days

Submissions: 15
Max Score: 30
Difficulty: Medium

Rate This Challenge:

More

Java 7

```
25
26 boolean checkBST(Node root) {
27     return check(root, Integer.MIN_VALUE, Integer.MAX_VALUE);
28 }
29
30 boolean check(Node root, int min, int max) {
31     if(root.data <= min || root.data >= max) return false;
32
33     boolean isLeftBST;
34     if(root.left == null) isLeftBST = true;
35     else isLeftBST = check(root.left, min, root.data);
36
37     boolean isRightBST;
38     if(root.right == null) isRightBST = true;
39     else isRightBST = check(root.right, root.data, max);
40       return isLeftBST && isRightBST;
41     }
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

ADS4:Bubble Sort

Bubble Sort Overview:

Bubble sort is a stable, in place srting algorithm named for smaller or larger elements "bubble" to the top of the list. Although the algorithm is simple, it is too slow and impractical for most problems even comparred to insertion sort, and is not recommended for large input.

Input Format

Input: arr[] = { 3, 5, 8, 4, 1, 9, -2 };

Constraints

.

Output Format

Output:[-2,1,3,4,5,8,9]

Sample Input 0

```
3, 5, 8, 4, 1, 9, -2
```

Sample Output 0

```
[-2, 1, 3, 4, 5, 8, 9]
```


Java 8

```
1 import java.io.*;
2 import java.util.*;
3 import java.text.*;
4 import java.math.*;
```

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

public class Solution {

    public static void main(String[] args) {

        int arr[] = {3,5,8,4,1,9,-2};

        int n = arr.length;
        bsort(arr);
        System.out.print("[");
        display(arr);
        System.out.print("]");
    }
    static void display(int arr[])
    {
        for (int i = 0; i < arr.length; i++)

        {
            System.out.print(arr[i]);
            if ( i < arr.length -1)
                System.out.print(", ");

        }
}
    public static void bsort(int arr[])

    {
        int n = arr.length;

        for (int i = 0; i < n - 1; i++)

        {
            for (int j = 0; j < n - i - 1; j++)
            {
                if (arr[j] > arr[j + 1])
                {
                    int temp = arr[j];

                    arr[j] = arr[j + 1];

                    arr[j + 1] = temp;
                }
            }
        }
    }
}
```

Line: 1 Col: 1

Upload Code as File | Test against custom input | Run Code | Submit Code