Description | Editorial | Solutions (11.6K) | Submissions

21. Merge Two Sorted Lists

Easy | 18.4K | 1.7K

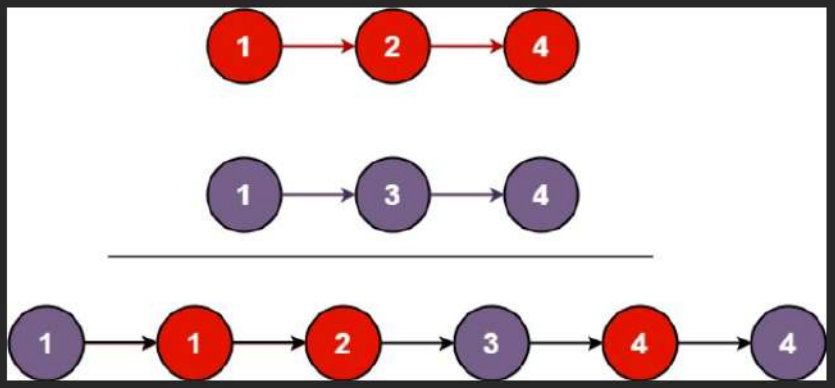
Companies

You are given the heads of two sorted linked lists `list1` and `list2`.

Merge the two lists in a one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list.*

Example 1:




```
Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]
```

Example 2:

```
Input: list1 = [], list2 = []
Output: []
```

Java | Auto

Press Esc to exit full screen

```
 * }
 */
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {

        if(list1!=null && list2!=null){
        if(list1.val<list2.val){
            list1.next=mergeTwoLists(list1.next,list2);
            return list1;
            }
            else{
                list2.next=mergeTwoLists(list1,list2.next);
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

list1 =
[1,2,4]

list2 =
[1,3,4]

Output

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

Console | Run | Submit

Description | Editorial | Solutions (15.9K) | Submissions

20. Valid Parentheses

Hint

Easy 20K 1.2K

Companies

Given a string `s` containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets.

Open brackets must be closed in the correct order.

Every close bracket has a corresponding open bracket of the same type.

Example 1:

```
Input: s = "()"
Output: true
```

Example 2:

```
Input: s = "()[]{}"
Output: true
```

Example 3:

```
Input: s = "(]"
Output: false
```

Constraints:

Java | Auto

Press Esc to exit full screen

```
6                stack.push(c);
7            } else {
8                if (stack.empty()) {
9                    return false;
10               }
11               if (c == ')' && stack.peek() == '(') {
12                   stack.pop();
13               } else if (c == '}' && stack.peek() == '{') {
14                   stack.pop();
15               } else if (c == ']' && stack.peek() == '[') {
16                   stack.pop();
17               } else {
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

```
s =
"()"
```

Output

```
true
```

Expected

```
true
```

Console | Run | Submit

Description | Editorial | Solutions (830) | Submissions

2101. Detonate the Maximum Bombs

Hint

Medium | 2K | 111

Companies

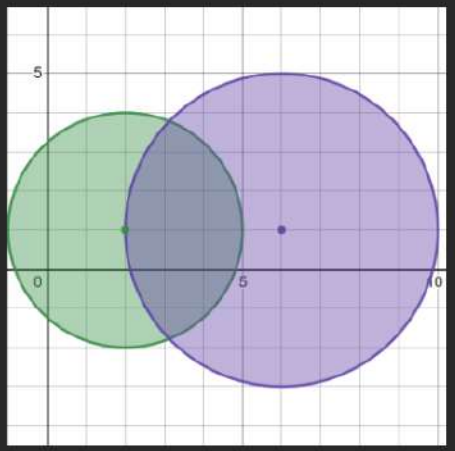
You are given a list of bombs. The **range** of a bomb is defined as the area where its effect can be felt. This area is in the shape of a **circle** with the center as the location of the bomb.

The bombs are represented by a **0-indexed** 2D integer array `bombs` where `bombs[i] = [x_i, y_i, r_i]`. `x_i` and `y_i` denote the X-coordinate and Y-coordinate of the location of the `i^th` bomb, whereas `r_i` denotes the **radius** of its range.

You may choose to detonate a **single** bomb. When a bomb is detonated, it will detonate **all bombs** that lie in its range. These bombs will further detonate the bombs that lie in their ranges.

Given the list of `bombs`, return *the **maximum** number of bombs that can be detonated if you are allowed to detonate **only one** bomb.*

Example 1:



Java | Auto

```java
class Solution {
    private void dfs(List<List<Integer>> gr, boolean[] visited, int[] c, int i) {
        visited[i] = true;
        c[0]++;
        for (int j : gr.get(i)) {
            if (!visited[j]) {
                dfs(gr, visited, c, j);
            }
        }
    }

    public int maximumDetonation(int[][] bombs) {
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

bombs =
`[[2,1,3],[6,1,4]]`

Output

2

Expected

2

Console | Run | Submit