# Problem – 56. Merge Intervals

Medium

LeetCode leetcode.com/problems/merge-intervals

## Problem Statement

- You are given an array of intervals, example:

```
intervals = [[1,3],[2,6],[8,10],[15,18]]
```

- Merge all overlapping intervals. So the output should be:

```
[[1,6],[8,10],[15,18]]
```

- Interval [1,3] was merged with [2,6]

# Solution – 56. Merge Intervals

Medium

LeetCode leetcode.com/problems/merge-intervals

## Solution

- Sort the array based on the beginning of the interval
- In C++, when applying `sort(intervals.begin(), intervals.end())`

  the default comparator compares `vector<vector<int>>` lexicographically:

  - it first compares the first element [0] of each sub-vector

  - if those are equal, it compares the second element [1] and so on
- Go over each interval and compare
- `interval[i][begin] <= interval[i – 1][end]` ? then merge
- To merge, set the current `interval[i][begin] to interval[i -1][begin]` and set the `interval[i][end]` to the maximum value between `interval[i][end]` and `interval[i -1][end]`
- If no merge is necessary, push the previous interval to the result array
- Once the loop finishes, add the last element and return the result

# Code – 56. Merge Intervals

Medium

LeetCode leetcode.com/problems/merge-intervals

## Code

Time: **O(n log n)** Space: **O(n)**

```
vector<vector<int>> merge(vector<vector<int>>& intervals) {
    if (intervals.empty()) return {};
    sort(intervals.begin(), intervals.end());
    vector<vector<int>> result;

    result.push_back(intervals[0]);

    for (int i = 1; i < intervals.size(); ++i) {
        vector<int>& current = intervals[i];
        vector<int>& previous = result.back();

        // check if they overlap, if so merge...
        // they're sorted, we know that:
        // previous[0] >= current[0]
        // 1 --- 3 (previous)
        //   2 ----- 6 (current)
        if (current[0] <= previous[1]) {
            // merge
            previous[1] = max(previous[1], current[1]);
        } else {
            result.push_back(current);
        }
    }
    return result;
}
```

## Problem – 435. Non-overlapping Intervals

Medium

LeetCode leetcode.com/problems/non-overlapping-intervals

### Problem Statement

- You are given an array of intervals vector<vector<int>> with start and end, Example:

  ```
  intervals = [[1,2],[2,3],[3,4],[1,3]]
  ```
- The intervals **must not** overlap each over
- You have to remove the minimum number of pairs to make it non-overlapping

# Solution – 435. Non-overlapping Intervals

Medium

LeetCode leetcode.com/problems/non-overlapping-intervals

## Solution

- Sort the array by the ending time:

  [[1,2],[2,3],[3,4],[1,3]] → [[1,2],[2,3],[1,3],[3,4]]
- In C++ a lambda function can be used with sort:

```
sort(intervals.begin(), intervals.end(), [](const vector<int>& a, const vector<int>&b) {
   return a[1] < b[1];
});
```
- Note that **std::sort** is not stable (opposite of **std::stable_sort**), so there is no guarantees that [2,3] comes before [1,3]. But for this algorithm, it doesn't matter
- Iterate over the array and check overlaps by comparing the end[i] with begin[i - 1]
- If they overlap, logically remove the current pair and count + 1
- Logically removing means just setting the end to compare to the previous element, so "skip" the current interval

# Code – 435. Non-overlapping Intervals

Medium

LeetCode leetcode.com/problems/non-overlapping-intervals

## Code

Time: **O(n log n)** Space: **O(1)**

```cpp
int eraseOverlapIntervals(vector<vector<int>>& intervals) {
    // sort by the ending time O(log n)
    sort(intervals.begin(), intervals.end(), [](const auto& a, const auto& b) {
            return a[1] < b[1];
    });
    int result= 0;
    int end = intervals[0][1];

    // O(n)
    for (int i = 1; i < intervals.size(); ++i) {
        // does it overlaps?
        if (intervals[i][0] < end) {
            ++result;
        } else {
            // it doesn't overlap, just 'skip'
            // the current interval
            end = intervals[i][1];
        }
    }
    return result;
}
```

# Problem – 110. Balanced Binary Tree

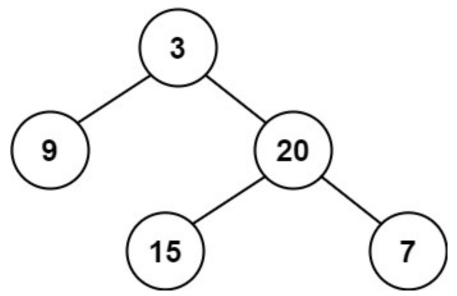
Easy

LeetCode leetcode.com/problems/balanced-binary-tree

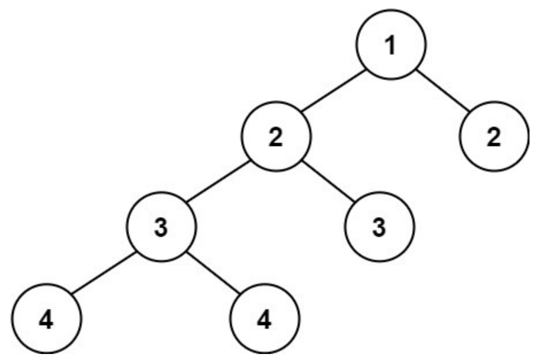
## Problem

- You are given the root of a binary tree
- Return true if it is height-balanced
- A tree is height-balanced when the height of two subtrees does not differ by two

### Height balanced



### Not Height balanced



# Problem – 110. Balanced Binary Tree

Easy

LeetCode leetcode.com/problems/balanced-binary-tree

## Solution

- Recursive approach: go all the way down
- Calculate the height of the left subtree
- Calculate the height of the right subtree
- Compare both to check if they differ by more than one
- Continue going up the tree to check all the nodes

# Problem – 110. Balanced Binary Tree

Easy

LeetCode leetcode.com/problems/balanced-binary-tree

## Code

Time: **O(n)** Space: **O(h)** where n is the number of the nodes and h is the height of the tree

```cpp
int checkHeight(TreeNode* node) {
    if (!node) return 0;

    int left = checkHeight(node->left);
    // left tree is unbalanced
    if (left == -1) return -1;

    int right = checkHeight(node->right);
    // right tree is unbalanced
    if (right == -1) return -1;

    // check the different, -1 is unbalanced
    if (abs(left - right) > 1) return -1;

    return max(left, right) + 1;
}

bool isBalanced(TreeNode* root) {
    return checkHeight(root) != -1;
}
```

## LINKED LIST

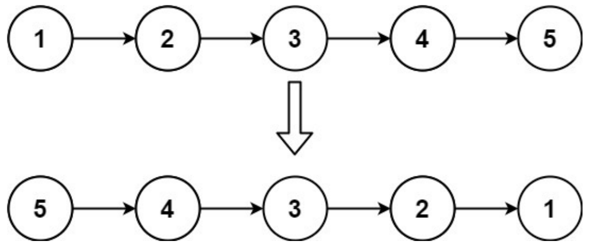
# Problem – 206. Reverse Linked List

Easy

LeetCode leetcode.com/problems/reverse-linked-list

## Problem

- This is a classic problem
- Given a singly linked list, reverse its order



# Solution – 206. Reverse Linked List

Easy

LeetCode leetcode.com/problems/reverse-linked-list

## Solution

- Use recursive approach
- Looking at the pseudo-code, this recursion will return the last node:

```
reverseList(head) {
    if (!head->next) return head
    node = reverseList(head->next);
    return node
}
```

- From end to beginning, each head will be a node in the list
- Therefore, you can change this node by setting a new head:

```
head->next->next = head;
head->next = nullptr;
```

## Code – 206. Reverse Linked List

Easy

LeetCode leetcode.com/problems/reverse-linked-list

### Code  Time: **O(n)**  Space: **O(1)**

```
ListNode* reverseList(ListNode* head) {
    if (!head->next) return head;
    ListNode* node = reverseList(head->next);
    head->next->next = head;
    head->next = nullptr;
    return node;
}
```

## Problem – 141. Linked List Cycle

Easy

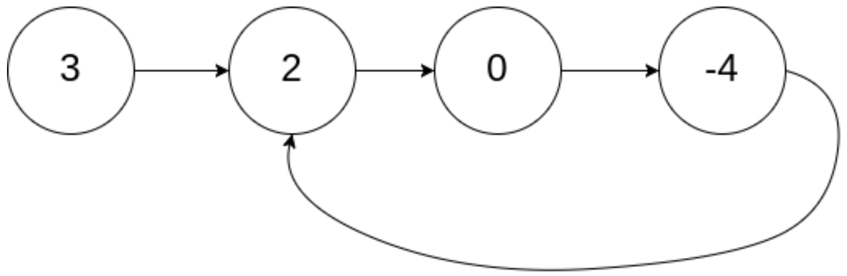
LeetCode leetcode.com/problems/linked-list-cycle

### Problem

- You are given the head of a linked list
- Return **true** if there is a cycle, false otherwise
- **Example:**

  In the image below, there is a cycle (-4 to 2)

  **Output**: true



# Solution – 141. Linked List Cycle

Easy

LeetCode leetcode.com/problems/linked-list-cycle

**Solution**

- Have two pointers: fast and slow
- Slow will go over each item in the linked list
- Fast will go twice as fast as slow (`fast = fast->next->next`)
- If fast reach at the end, there is no cycle
- If fast encounter slow, there is a cycle, return true