Code – 141. Linked List Cycle

Easy

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

Code

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

```
bool hasCycle(ListNode *head) {
    if (!head || !head->next) return false;
    ListNode* slow = head;
    ListNode* fast = head;
    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast) return true;
    }
    return false;
}
```

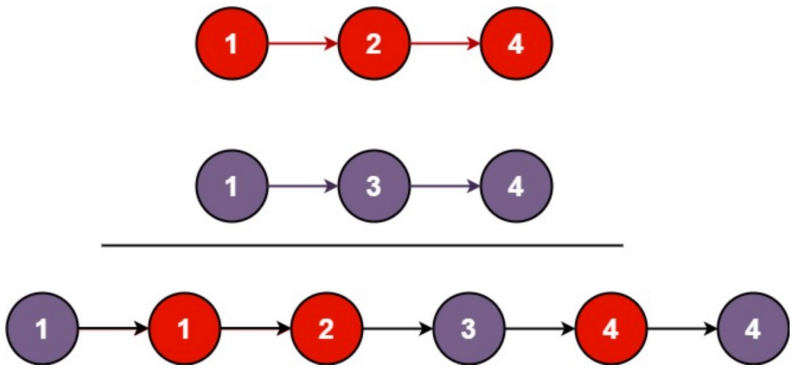
Problem – 21. Merge Two Sorted Lists

Easy

LeetCode leetcode.com/problems/merge-two-sorted-lists

Problem

- You are given the head of two linked lists (list1 and list2)
- Merge the two lists into one **sorted** list



Solution – 21. Merge Two Sorted Lists

Easy

LeetCode leetcode.com/problems/merge-two-sorted-lists

Solution

- Recursively explore the two lists. Base case:

  ```
  if (!list1) return list2;
  ```

  ```
  if (!list2) return list1;
  ```

- Compare the value of the current node of list 1 and list 2

  ```
  if (list1->val > list2->val) { ...
  ```

- Set the next node of the node with the minimum value:

  assume the previous condition is true, so

  ```
  list2->next = mergeTwoLists(list1, list2->next);
  ```

  ```
  return list2;
  ```

  meaning, we want list2->next to come before list1. But we do this recursively since we need the next result

Code – 21. Merge Two Sorted Lists

Easy

LeetCode leetcode.com/problems/merge-two-sorted-lists

Code

Time: **O(n + m)** Space: **O(n + m)** where n is the length of list1 and m is the length of list2

```
ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
    if (!list1) return list2;
    if (!list2) return list1;

    if (list1->val < list2->val) {
        list1->next = mergeTwoLists(list1->next, list2);
        return list1;
    } else {
        list2->next = mergeTwoLists(list2->next, list1);
        return list2;
    }
}
```

Problem – 23. Merge k Sorted Lists

Hard

LeetCode leetcode.com/problems/merge-k-sorted-lists

Problem

- You are given an **array of k linked lists**
- Each linked list is **sorted** in ascending order
- Merge all linked lists into one **sorted** linked-lists

Solution – 23. Merge k Sorted Lists

Hard

LeetCode leetcode.com/problems/merge-k-sorted-lists

Solution

- Create a function to merge two lists
- Go over the lists and merge with each over; **or**
- Use divide and conquer to merge (more optimal)
- Divide and conquer is more efficient because it avoids merging a big list with a small one multiple times

Code – 23. Merge k Sorted Lists

Hard

LeetCode leetcode.com/problems/merge-k-sorted-lists

Code

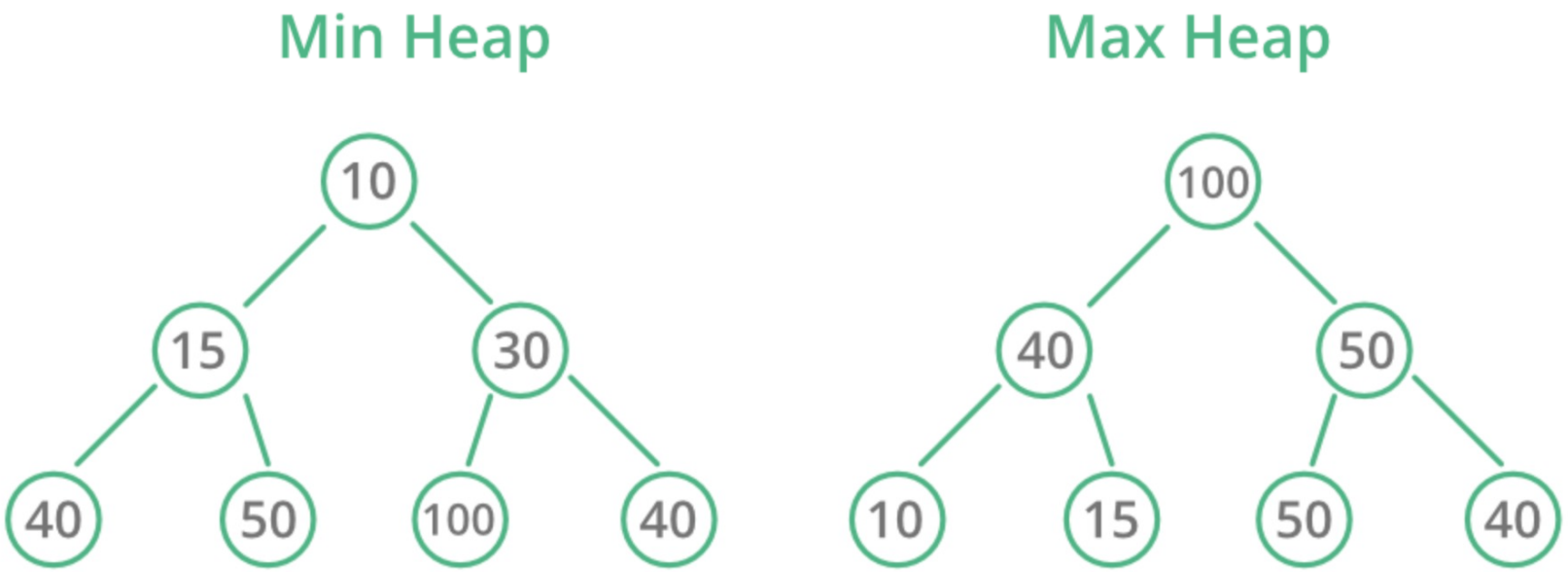
Time: **O(N log k)** Space: **O(log k)** where **N** is the total number of nodes across all lists and **k** is the number of lists

```
ListNode* mergeKLists(vector<ListNode*>& lists) {
    if (lists.empty()) return nullptr;
    return divideAndConquer(lists, 0 /* left */, lists.size() - 1 /* right */);
}

ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
    if (!l1) return l2;
    if (!l2) return l1;

    if (l1->val < l2->val) {
        l1->next = mergeTwoLists(l1->next, l2);
        return l1;
    } else {
        l2->next = mergeTwoLists(l2->next, l1);
        return l2;
    }
}

ListNode* divideAndConquer(vector<ListNode*> lists, int left, int right) {
    if (left == right) return lists[right];

    int mid = left + (right - left) / 2;
    ListNode* l1 = divideAndConquer(lists, left, mid);
    ListNode* l2 = divideAndConquer(lists, mid + 1, right);
    return mergeTwoLists(l1, l2);
}
```

HEAP / PRIORITY QUEUE

Heap

- **Heap** is a complete binary tree that satisfy the heap property (max or min)
- **Min heap**: root node contains the minimum value
- **Max heap**: root node contains the maximum value



Heap in C++

Two main ways to implement:

1. Using **std::make_heap** from **<algorithm>**

```
std::make_heap(RandomIt first, RandomIt last)
std::push_heap(RandomIt first, RandomIt last)
std::pop_heap(RandomIt first, RandomIt last)
std::sort_heap(RandomIt first, RandomIt last)
```

2. Using **std::priority_queue** from **<queue>** **(recommended)**

```
std::priority_queue<T, Container, Compare>
```

Heap in C++ – std::priority_queue example

Min heap

`std::priority_queue<int, std::vector<int>, std::greater<int>>`

Max heap

`std::priority_queue<int>` or

`std::priority_queue<int, std::vector<int> std::less<int>>`

```
// Min heap
std::priority_queue<int, std::vector<int>, std::greater<int>> minHeap;

minHeap.push(3);
minHeap.push(6);
minHeap.push(4);
// remove top element (3)
minHeap.pop();
// root node (top) is now 4
std::cout << minHeap.top();
```

Problem – 215. Kth Largest Element in an Array

Medium

LeetCode leetcode.com/problems/kth-largest-element-in-an-array

Problem

- You are given an array of integers **nums** and an integer **k**
- Find the **kth** largest element
- **Example:**

  Input

  `nums = [3, 2, 1, 5, 6, 4]`

  `k = 2`

  Output: 5

Solution – 215. Kth Largest Element in an Array

Medium

LeetCode leetcode.com/problems/kth-largest-element-in-an-array

Solution

- Start with a **min heap**
- Loop through the **nums** array:

  Add the element to the min heap

  Check if the size of the heap is always less than **k**. If the size is greater, pop the minimum element
- Return the element from the top: the k^{th} largest element

Code – 215. Kth Largest Element in an Array

Medium

LeetCode leetcode.com/problems/kth-largest-element-in-an-array

Code

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

```cpp
int findKthLargest(vector<int>& nums, int k) {
    std::priority_queue<int, std::vector<int>, std::greater<int>> minHeap;
    // O(n)
    for (const auto& num : nums) {
        // O(log k) since the heap is bounded to k elements
        minHeap.push(num);
        if (minHeap.size() > k) {
            minHeap.pop();
        }
    }
    return minHeap.top();
}
```

Problem – 347. Top K Frequent Elements

Medium

LeetCode leetcode.com/problems/top-k-frequent-elements

Problem

- You are given an **array** of numbers and an **integer** k
- Return an array with the **k** most frequent elements

Example

Input:

```
nums = [1,1,1,2,2,3], k = 2
```

Output:

```
[1,2]
```

Solution – 347. Top K Frequent Elements

Medium

LeetCode leetcode.com/problems/top-k-frequent-elements

Solution (1) - hashmap + array sort

- Go over the array, count the numbers and store them in an *unordered_map*

  Example:

  ```
  nums = [1,1,1,2,2,3], k = 2
  freq[1] = 3
  freq[2] = 2
  ...
  ```

- Go over the *unordered_map*, add to an array and sort descending
- Create another array adding the **k** first elements and return

Code – 347. Top K Frequent Elements

Medium

LeetCode leetcode.com/problems/top-k-frequent-elements

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

```
vector<int> topKFrequent(vector<int>& nums, int k) {
    // 1. Create the number's frequency map
    // O(n)
    unordered_map<int, int> freq;
    for (const auto& num : nums) {
        freq[num] += 1;
    }
    // 2. Create an array with the frequencies
    vector<pair<int, int>> freqVec(freq.begin(), freq.end());
    // 3. Sort by the frequency O(n log n)
    sort(freqVec.begin(), freqVec.end(), [](auto& a, auto& b) {
            return a.second > b.second;
            });
    // 4. Create the result with the k first elements
    // O(k)
    vector<int> result;
    for (int i = 0; i < k; ++i) {
        result.push_back(freqVec[i].first);
    }
    return result;
}
```

Solution – 347. Top K Frequent Elements

Medium

LeetCode leetcode.com/problems/top-k-frequent-elements

Solution (2) - hashmap + min heap

- Go over the array, count the numbers and store them in an *unordered_map*

  Example:

  ```
  nums = [1,1,1,2,2,3], k = 2
  freq[1] = 3
  freq[2] = 2
  ...
  ```

- Go over the frequencies, add to a min heap. If the size of the heap exceeds **k**, remove the top one (the minimum value)
- Create another array result adding all elements from the heap and return it