

LeetCode Answers

Keaton Clark

April 25, 2022

Contents

I	Medium	1
1	2. Add Two numbers	3
1.1	Description	3
1.2	Results	3
1.3	Attempt 1 - 2022-04-23	4
1.4	Attempt 2 - 2022-04-23	5
II	Hard	7
2	4. Median of Two Sorted Arrays	9
2.1	Description	9
2.2	Results	9
2.3	Attempt 1 - 2022-04-24	10
3	23. Merge K Sorted Lists	11
3.1	Description	11
3.2	Results	11
3.3	Attempt 1	12
3.4	Attempt 2	13

Part I

Medium

Chapter 1

2. Add Two numbers

1.1 Description

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list. You may assume the two numbers do not contain any leading zero, except the number 0 itself.

1.2 Results

Runtime: 47 ms, faster than 55.66% of C++ online submissions for Add Two Numbers.

Memory Usage: 71.4 MB, less than 50.94% of C++ online submissions for Add Two Numbers.

1.3 Attempt 1 - 2022-04-23

Worked great until I realized that the numbers can be up to $9 * 10^{100}$. Turn each list into ints, sums them, and uses a recursive function to convert it back into a reversed linked list.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode *addTwoNumbers(ListNode* l1, ListNode* l2) {
        int place = 1;
        int s1 = 0, s2 = 0;
        while(l1) {
            s1 += l1->val * place;
            place *= 10;
            l1 = l1->next;
        }
        place = 1;
        while(l2) {
            s2 += l2->val * place;
            place *= 10;
            l2 = l2->next;
        }
        int sum = s1 + s2;
        return createNodes(sum, nullptr);
    }
    ListNode *createNodes(int x, ListNode *curr) {
        if (x >= 10)
            curr = createNodes(x / 10, curr);
        return new ListNode(x % 10, curr);
    }
};
```


1.4 Attempt 2 - 2022-04-23

The correct way to do it. Sums them just like an adder in a cpu would. Sums each digit and has a carry digit to keep track of overflows.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */

class Solution {
public:
    ListNode *addTwoNumbers(ListNode* l1, ListNode* l2) {
        int sum = l1->val + l2->val;
        int carry = (sum > 9) ? 1 : 0;
        auto output = new ListNode();
        auto curr = output;
        l1 = l1->next;
        l2 = l2->next;
        if (carry == 1) {
            curr->val = sum % 10;
        } else {
            curr->val = sum;
        }
        while (l1 && l2) {
            sum = l1->val + l2->val + carry;
            carry = (sum > 9) ? 1 : 0;
            curr->next = new ListNode();
            curr = curr->next;
            if (carry == 1) {
                curr->val = sum % 10;
            } else {
                curr->val = sum;
            }
            l1 = l1->next;
            l2 = l2->next;
        }
        while (l1) {
            sum = l1->val + carry;
            carry = (sum > 9) ? 1 : 0;
            curr->next = new ListNode((carry) ? sum % 10 : sum);
            curr = curr->next;
            l1 = l1->next;
        }
        while (l2) {
            sum = l2->val + carry;
            carry = (sum > 9) ? 1 : 0;
            curr->next = new ListNode((carry) ? sum % 10 : sum);
            curr = curr->next;
            l2 = l2->next;
        }
        if (carry) {
            curr->next = new ListNode(carry);
        }
        return output;
    }
};
```


Part II

Hard

Chapter 2

4. Median of Two Sorted Arrays

2.1 Description

Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return the median of the two sorted arrays. The overall run time complexity should be $O(\log(m+n))$.

2.2 Results

Runtime: 44 ms, faster than 69.41% of C++ online submissions for Median of Two Sorted Arrays.

Memory Usage: 89.7 MB, less than 42.98% of C++ online submissions for Median of Two Sorted Arrays.

2.3 Attempt 1 - 2022-04-24

Just use the given cpp merge functions. This can be further optimized.

```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        auto total = nums1.size() + nums2.size();
        vector<int> nums(total);
        merge(nums1.begin(), nums1.end(), nums2.begin(), nums2.end(), nums.begin());
        if (total % 2 == 0) {
            return (nums[total/2] + nums[total/2 - 1]) / 2.0;
        } else {
            return nums[total/2];
        }
    }
};
```

Chapter 3

23. Merge K Sorted Lists

3.1 Description

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order. Merge all the linked-lists into one sorted linked-list and return it.

3.2 Results

Attempt 1 Runtime: 30 ms, faster than 63.70% of C++ online submissions for Merge k Sorted Lists.

Memory Usage: 13.6 MB, less than 36.67% of C++ online submissions for Merge k Sorted Lists.

Attempt 2 Runtime: 20 ms, faster than 94.42% of C++ online submissions for Merge k Sorted Lists.

Memory Usage: 13.4 MB, less than 45.00% of C++ online submissions for Merge k Sorted Lists.

3.3 Attempt 1

Had to learn priority queues for this one, that's something they don't teach you in school. Loads each link into the queue which is then sorted as they are added then one by one links the top node of the queue to the next node in the queue.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
using namespace std;
class Compare {
public:
    bool operator() (ListNode* A, ListNode* B) {
        return (A->val > B->val);
    }
};
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        if (lists.size() == 1) return *lists.begin();
        priority_queue<ListNode*, vector<ListNode*>, Compare> pq;
        for (auto i = lists.begin(); i < lists.end(); i++) {
            auto curr = *i;
            if (!curr) continue;
            do {
                pq.push(curr);
            } while ((curr = curr->next));
        }
        auto output = (pq.empty()) ? nullptr : pq.top();
        while (!pq.empty()) {
            auto curr = pq.top();
            pq.pop();
            curr->next = (pq.empty()) ? nullptr : pq.top();
        }
        return output;
    }
};
```


3.4 Attempt 2

I had so much fun with this I decided to optimize this. Since we know the linked lists are already sorted we can just pull the leading value from each list and everytime we use a value from that list off the queue we pull another value off that corresponding list. This means we only have at most k comparisons per each insertion.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
typedef pair<ListNode*, int> p;
using namespace std;
class Compare {
public:
    bool operator() (p A, p B) {
        return (A.first->val > B.first->val);
    }
};
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        if (lists.size() == 1) return *lists.begin();
        priority_queue<p, vector<p>, Compare> pq;
        for (int i = 0; i < lists.size(); i++) {
            if (!lists[i]) continue;
            pq.push({lists[i], i});
            lists[i] = lists[i]->next;
        }
        auto output = (pq.empty()) ? nullptr : pq.top().first;
        while (!pq.empty()) {
            auto curr = pq.top();
            pq.pop();
            if (lists[curr.second]) {
                pq.push({lists[curr.second], curr.second});
                lists[curr.second] = lists[curr.second]->next;
            }
            curr.first->next = (pq.empty()) ? nullptr : pq.top().first;
        }
        return output;
    }
};
```

