1 Divide And Conquer

1.1 problem list

- 来源列表：LeetCode – 分治算法。
- 要求尽可能使用分治算法。

id	title	label
1	p53 最大子序和 easy	推荐
2	p169 求众数 easy	推荐
3	p215 数组中的第K个最大元素 normal	推荐
4	p240 搜索二维矩阵 II normal	
5	p23 合并K个排序链表 hard	推荐
6	p493 翻转对 hard	
7	p4 两个排序数组的中位数 hard	推荐
8	p312 戳气球 hard 动态规划	推荐
9	p241 为运算表达式设计优先级 hard	
10	p282 给表达式添加运算符 hard	

1.2 unofficial solution

1.2.1 p53 最大子序和

分治做法

- $O(NlogN)$。
- 数组分成左右两部分，考虑三种情况：
  - 答案数组完全落在左边。
  - 。。。。完全落在右边。
  - 。。。。落在两边。
- 前两者递归解决。
- 第三种为左边的一个后缀数组 + 右边的一个前缀数组。

- $O(N)$ 求数组的最大前缀/后缀数组。

```
class Solution {
private:
    int maxPreArray(vector<int> nums) {
        int ans = -INT_MAX, tmp = 0;
        for (int e : nums) {
            tmp += e;
            ans = max(ans, tmp);
        }
        return ans;
    }
    int maxSufArray(vector<int> nums) {
        reverse(nums.begin(), nums.end());
        return maxPreArray(nums);
    }
public:
    int maxSubArray(vector<int> nums) {
        if (nums.empty()) return -INT_MAX;
        if (nums.size() == 1) return nums[0];
        int mid = nums.size() >> 1;
        vector<int> nums_left(nums.begin(), nums.begin() + mid);
        vector<int> nums_right(nums.begin() + mid, nums.end());;
        int ans = max(maxSubArray(nums_left), maxSubArray(nums_right));
        ans = max(ans, maxSufArray(nums_left) + maxPreArray(nums_right));
        return ans;
    }
};
```

贪心做法

- 数字能拿就拿，拿完和若为负数，则根据贪心思想前面拿的全部舍弃。

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int ans = -INT_MAX, tmp = 0;
        for (auto e : nums) {
            ans = max(ans, tmp += e);
            tmp = max(tmp, 0);
        }
        return ans;
    }
};
```

1.2.2 p169 求众数

分治做法

$O(NlogN)$。

把数组拆成左右两部分，最终的众数在左边或右边至少其中一边里面也必然是众数。

递归处理左右两边。

假设现在众数只有两个可能值，则遍历一遍数组看下谁出现的次数就好。

机智做法 1

$O(N)$。

相像一个打擂台的过程：

- 相同数字的为统一派系
- 每上台一个数字
  - 如果和擂主同派系，则留下来，势力++；
  - 如果和擂主异派系，则台上要花一个人的代价守擂，势力--；
  - 若台上没人，则成为新的擂主。
- 无关上台顺序，拥有超过总人数一半的人数的派系必然为最终的擂主。

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int ans = nums[0], cnt = 0;
        for (auto e : nums) {
            if (e == ans) cnt++;
            else if (--cnt < 0) {
                ans = e;
                cnt = 1;
            }
        }
        return ans;
    }
};
```

机智做法 2

$O(NlogN)$。

因为出现次数严格超过一半，所以排序后中位数即众数。

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        return nums[nums.size() >> 1];
    }

};
```

1.2.3 p215 数组中的第K个最大元素

第K大函数

$O(N)$。

手动实现即为分治过程，具体做法搜索STL nth_element内部实现，大题过程类似快排。

没实现过的建议自己实现一下。

```
class Solution {
public:
    int findKthLargest(vector<int>& nums, int k) {
        k = nums.size() - k;
        nth_element(nums.begin(), nums.begin() + k, nums.end());
        return nums[k];
    }
};
```

1.2.4 p240 搜索二维矩阵 II

类似滑窗的做法

$O(N)$。

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        if (matrix.empty() || matrix.begin()->empty()) {
            return false;
        }
        int n = matrix.size();
        int m = matrix[0].size();
        int col = m - 1;
        for (int row = 0; row < n; row++) {
            while (col >= 0 && matrix[row][col] > target) {
                col--;
            }
            if (col < 0) {
                return false;
            }
            else if (matrix[row][col] == target) {
                return true;
```

```
            }
        }
        return false;
    }
};
```

1.2.5 p23 合并K个排序链表

分治

$O(NlogK)$，N 为链表总元素数量。

每次拆成两边各自做，每个链表最多参与合并 $logK$ 次（想象一颗二叉树）。

```
class Solution {
private:
    ListNode * merge2Lists(ListNode * list1, ListNode * list2) {
        ListNode head(0);
        ListNode * tail = &head;
        while (list1 && list2) {
            if (list1->val > list2->val) {
                swap(list1, list2);
            }
            tail->next = list1;
            tail = list1;
            list1 = list1->next;
        }
        if (list1) tail->next = list1;
        if (list2) tail->next = list2;
        return head.next;
    }

public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        if (lists.empty()) return NULL;
        if (lists.size() == 1) return lists[0];
        vector<ListNode*> nxt_lists;
        int mid = lists.size() + 1 >> 1;
        for (int i = mid; i < lists.size(); i++) {
            lists[i - mid] = merge2Lists(lists[i - mid], lists[i]);
        }
        lists.resize(mid);
        return mergeKLists(lists);

    }
};
```

1.2.6 p493 翻转对

分治

$O(NlogN)$。

分成左右两部分，先考虑每个部分各自里面的逆序对。

刚好也是归并排序的过程，所以递归中边做便排序。

接着考虑左右两部分构成的逆序（此时两部分都有序），滑窗过去。

答案最大为 $O((\frac{N}{2})^2)$ 不会超 int ，但中间可能超。

```
class Solution {
private:
    vector<int> merge(vector<int> left, vector<int> right) {
        vector<int> ans;
        int temp = 0;
        for (auto e : left) {
            while (temp < right.size() & e > right[temp]) {
                ans.push_back(right[temp++]);
            }
            ans.push_back(e);
        }
        while (temp < right.size()) {
            ans.push_back(right[temp++]);
        }
        return ans;
    }
public:
    int reversePairs(vector<int>& nums) {
        if (nums.size() < 2) return 0;
        int mid = nums.size() >> 1;
        vector<int> left(nums.begin(), nums.begin() + mid);
        vector<int> right(nums.begin() + mid, nums.end());
        int ans = reversePairs(left) + reversePairs(right);
        int temp = 0;
        for (auto e : left) {
            while (temp < right.size() && e > right[temp] * 2ll) {
                temp++;
            }
            ans += temp;
        }
        nums = merge(left, right);
        return ans;
    }
};
```

树状数组

$O(NlogN)$。

离散，$\frac{nums[i]}{2} > nums[j]$ 进行树状数组查询。

```
class Fenwick {
    private:
        static const int N = 5e4 + 7;
        int n, a[N];
    public:
        Fenwick(int _n) {
            fill_n(a + 1, n = _n, 0);
        }
        void add(int p) {
            for (; p <= n; p += p & -p) a[p]++;
        }
        int qry(int p) {
            int r = 0;
            for (; p; p -= p & -p) r += a[p];
            return r;
        }
};
class Solution {
public:
    int reversePairs(vector<int>& nums) {
        vector<int> table = nums;
        sort(table.begin(), table.end());
        Fenwick fen(nums.size());
        int ans = 0;
        for (int i = nums.size() - 1; i >= 0; i--) {
            int pos_qry = upper_bound(table.begin(), table.end(), nums[i] - 1 >> 1) -
table.begin() - 1 + 1;
            ans += fen.qry(pos_qry);
            int pos_add = lower_bound(table.begin(), table.end(), nums[i]) - table.begin() + 1;
            fen.add(pos_add);
        }
        return ans;
    }
};
```

1.2.7 p4 两个排序数组的中位数

二分

$O(NlogN)$

若把两个数组合并成有序，称小于中位数的部分为👈部分，反之为👉部分。

两个原始数组必然各自存在一个截断处，使得截断后一部分全在👈部分，另一部分全在👉部分。

二分其中一个数组的截断位置，根据中位数的设定，可以计算出另一个数组的截断位置。

进而判定下一个二分范围。

```
class Solution {
```

```
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        int n1 = nums1.size(), n2 = nums2.size(), m = n1 + n2 >> 1;
        nums1.push_back(INT_MAX);
        nums2.push_back(INT_MAX);
        for (int l = 0, r = min(m, n1), mid1 = l + r >> 1; l <= r; mid1 = l + r >> 1) {
            int mid2 = m - mid1;
            if (mid2 > n2) {
                l = mid1 + 1;
                continue;
            }
            if (!mid2 || nums1[mid1] >= nums2[mid2 - 1]) {
                if (!mid1 || nums2[mid2] >= nums1[mid1 - 1]) {
                    int elementl = max(nums1[mid1 - 1], nums2[mid2 - 1]);
                    int elementr = min(nums1[mid1], nums2[mid2]);
                    return (n1 + n2) & 1 ? elementr : (elementl + elementr) / 2.;
                }
                else {
                    r = mid1 - 1;
                }
            }
            else {
                l = mid1 + 1;
            }
        }
    }
};
```

1.2.8 p312 戳气球

动态规划

$O(N^3)$。

设 $dp[l][r]$ 为把 $[l+1, r-1]$ 这些气球全部戳破的最大收益。

则原问题的解为 $dp[-1][n]$。

考虑 $[l+1, r-1]$ 最后戳破的气球为 k，则 $[l, r]$ 可以变成两个子问题 $[l][k]$，$[k][r]$。

```
class Solution {
public:
    int maxCoins(vector<int>& nums) {
        nums.insert(nums.begin(), 1);
        nums.push_back(1);
        int n = nums.size();
        int** dp = new int*[n];
        for (int i = 0; i < n; i++) dp[i] = new int [n];


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

```
            dp[i][i + 1] = 0;
        }
        for (int lenth = 3; lenth <= n; lenth++) {
            for (int l = 0; l + lenth <= n; l++) {
                int r = l + lenth - 1;
                dp[l][r] = -INT_MAX;
                for (int k = l + 1; k < r; k++) {
                    dp[l][r] = max(dp[l][r], dp[l][k] + dp[k][r] + nums[k] * nums[l] * nums[r]);
                }
            }
        }
        return dp[0][n - 1];
    }
};
```

1.2.9 p241 为运算表达式设计优先级

分治枚举

$O(n!)$。

由于可以随意加括号，所以运算顺序可以随意定。

枚举最后进行运算的符号，然后就变成了两个子问题。

```
class Solution {
private:
    bool isOperator(char ch) {
        return ch == '+' || ch == '-' || ch == '*';
    }
public:
    vector<int> diffWaysToCompute(string input) {
        vector<int> result;
        for (int i = 0; i < input.size(); i++) {
            char ch = input[i];
            if ( ! isOperator(ch)) {
                continue;
            }
            auto v1 = diffWaysToCompute(input.substr(0, i));
            auto v2 = diffWaysToCompute(input.substr(i + 1));
            for (auto e1 : v1) {
                for (auto e2 : v2) {
                    switch (ch) {
                        case '+' :
                            result.push_back(e1 + e2);
                            break;
                        case '-' :
                            result.push_back(e1 - e2);
                            break;
                        case '*' :
```

```
                            result.push_back(e1 * e2);
                            break;
                        default :
                            assert(false);
                            break;
                    }
                }
            }
        }
        if (result.empty()) {
            result.push_back(stoi(input));
        }
        return result;
    }
};
```