# 归并排序 - 从二路到多路

## 基础知识

- 思路
  - 处理左边 得到左边的信息
  - 处理右边 得到右边的信息
  - 完成合并过程 得到横跨两边的信息
- 归并排序的实现代码

```c
void merge_sort(int *arr, int l, int r) {
    if (l >= r) return;

    int mid = (l + r) >> 1;
    merge_sort(arr, l, mid);
    merge_sort(arr, mid + 1, r);

    int *temp = (int *) malloc(sizeof(int) * (r - l + 1));

    int k = 0, p1 = l, p2 = mid + 1;
    while (p1 <= mid || p2 <= r) {
        if ((p2 > r) || (p1 < mid && arr[p1] <= arr[p2])) {
            temp[k] = arr[p1];
            k += 1;
            p1 += 1;
        } else {
            temp[k] = arr[p2];
            k += 1;
            p2 += 1;
        }
    }

    for (int i = l; i <= r; i++) arr[i] = temp[i - l];
    free(temp);
    return;
}
```

- 有序文件的合并

```c
typedef struct Data {
    FILE *fin;
    int val, flag;
} Data;

int main(int argc, char *argv[]) {
    int n = argc - 1;

    Data *data = (Data *) malloc(sizeof(Data) * n);

    for (int i = 1; i <= n; ++i) {
        data[i - 1].fin = fopen(argv[i], "r");

        if (fscanf(data[i - 1].fin, "%d", &data[i - 1].val) == EOF) {
            data[i - 1].flag = 1;
        } else {
            data[i - 1].flag = 0;
        }
    }

    FILE *fout = fopen("output", "w");

    while (1) {
        int flag = 0;
        int ind = -1;

        for (int i = 0; i < n; ++i) {
            if (data[i].flag) continue;
            if (ind == -1 || data[i].val < data[ind].val) {
                ind = i;
            }
        }
        if (ind != -1) {
            fprintf(fout, "%d\n", data[ind].val);
            if (fscanf(data[ind].fin, "%d", &data[ind].val) == EOF) {
                data[ind].flag = 1;
            } else {
                data[ind].flag = 0;
            }
            flag = 1;
        }
        if (flag == 0) break;
    }

    return 0;
}
```

1. 数组中的逆序对

```
class Solution {
public:
    vector<int> temp;
    int countReversePairs(vector<int>& nums, int l, int r) {
        if (l >= r) return 0;
        int mid = (l + r) >> 1, ans = 0;
        ans += countReversePairs(nums, l, mid);
        ans += countReversePairs(nums, mid + 1, r);
        int k = l, p1 = l, p2 = mid + 1;
        while ((p1 <= mid) || (p2 <= r)) {
            if ((p2 > r) || (p1 <= mid && nums[p1] <= nums[p2])) {
                temp[k++] = nums[p1++];
            } else {
                temp[k++] = nums[p2++];
                ans += (mid - p1 + 1);
            }
        }
        for (int i = l; i <= r; i++) nums[i] = temp[i];
        return ans;
    }
    int reversePairs(vector<int>& nums) {
        while (temp.size() < nums.size()) temp.push_back(0);
        return countReversePairs(nums, 0, nums.size() - 1);
    }
};
```

2. 合并K个升序链表 (分治合并链表)

```
/**
 * 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:
    template<typename T>
    class Heap : public vector<T> {
    public :
        template<typename FUNC_T>
        Heap(FUNC_T cmp) : cmp(cmp) {}
        void push(T data) {
            this->push_back(data);
            push_heap(this->begin(), this->end(), cmp);
            return ;
        }
        void pop() {
            pop_heap(this->begin(), this->end(), cmp);
            this->pop_back();
            return ;
        }
        T &top() {
            return this->at(0);
        }
    private:
        function<bool(T, T)> cmp;
    };
    struct Data {
        Data(int k, ListNode *node) : k(k), node(node) {}
        int k;
        ListNode *node;
    };
    struct CMP {
        bool operator()(const Data &a, const Data &b) {
            if (a.node->val - b.node->val) return a.node->val > b.node->val;
            return a.k > b.k;
        }
    };
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        ListNode ret, *p = &ret;
        Heap<Data> h{CMP()};
        for (int i = 0; i < lists.size(); i++) {
            if (lists[i] == nullptr) continue;
            h.push(Data{i, lists[i]});
        }
        while (h.size()) {
            Data cur = h.top();
            h.pop();
            if (cur.node->next) h.push(Data{cur.k, cur.node->next});
            p->next = cur.node;
            p = p->next;
        }
        p->next = nullptr;
        return ret.next;
    }
};
```

3. 排序链表

```
/**
 * 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 *mergeSort(ListNode *head, int n) {
        if (n <= 1) return head;
        int l_cnt = (n >> 1), r_cnt = n - l_cnt;
        ListNode ret, *l = head, *r = l, *p = l;
        for (int i = 1; i < l_cnt; i++) p = p->next;
        r = p->next; p->next = nullptr;
        l = mergeSort(l, l_cnt);
        r = mergeSort(r, r_cnt);
        p = &ret;
        while (l || r) {
            if (r == nullptr || (l && l->val <= r->val)) {
                p->next = l; p = l; l = l->next;
            } else {
                p->next = r; p = r; r = r->next;
            }
        }
        return ret.next;
    }
    ListNode* sortList(ListNode* head) {
        int n = 0;
        ListNode *p = head;
        while (p) n += 1, p = p->next;
        return mergeSort(head, n);
    }
};
```

4. 两棵二叉搜索树中的所有元素 (树)

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    void getNums(TreeNode *root, vector<int> &nums) {
        if (root == nullptr) return ;
        getNums(root->left, nums);
        nums.push_back(root->val);
        getNums(root->right, nums);
        return ;
    }
    vector<int> getAllElements(TreeNode* root1, TreeNode* root2) {
        vector<int> l, r;
        getNums(root1, l);
        getNums(root2, r);
        vector<int> temp;
        int p1 = 0, p2 = 0;
        while (p1 < l.size() || p2 < r.size()) {
            if ((p2 >= r.size()) || (p1 < l.size() && l[p1] <= r[p2])) {
                temp.push_back(l[p1++]);
            } else {
                temp.push_back(r[p2++]);
            }
        }
        return temp;
    }
};
```

5. 子数组和排序后的区间和

```
class Solution {
public:
    struct Data {
        Data(int i, int j, int val) : i(i), j(j), val(val) {}
        int i, j, val;
    };
    struct CMP {
        bool operator()(const Data &a, const Data &b) {
            if (a.val - b.val) return a.val > b.val;
            return a.i > b.i;
        }
    };
    int rangeSum(vector<int>& nums, int n, int left, int right) {
        priority_queue<Data, vector<Data>, CMP> q;
        for (int i = 0; i < n; i++) {
            q.push(Data{i, i, nums[i]});
        }
        int ans = 0, mod_num = 1e9 + 7;
        for (int i = 1; i <= right; i++) {
            Data cur = q.top();
            q.pop();
            if (i >= left) ans = (ans + cur.val) % mod_num;
            if (cur.j + 1 < n) {
                q.push(Data{cur.i, cur.j + 1, cur.val + nums[cur.j + 1]});
            }
        }
        return ans;
    }
};
```

6. 区间和的个数 (难)

```
class Solution {
public:
    int countTwoPart(vector<long long>& sum, int l1, int r1, int l2, int r2, int lower, int upper) {
        int ans = 0;
        for (int i = l2, a = l1, b = l1; i <= r2; i++) {
            while (a <= r1 && sum[i] - sum[a] > upper) a += 1;
            while (b <= r1 && sum[i] - sum[b] >= lower) b += 1;
            ans += b - a;
        }
        return ans;
    }
    int lower, upper;
    vector<long long> temp;
    int mergeSort(vector<long long>& nums, int l, int r) {
        if (l >= r) return 0;
        int mid = (l + r) >> 1, ans = 0;
        ans += mergeSort(nums, l, mid);
        ans += mergeSort(nums, mid + 1, r);
        ans += countTwoPart(nums, l, mid, mid + 1, r, lower, upper);
        int k = l, p1 = l, p2 = mid + 1;
        while (p1 <= mid || p2 <= r) {
            if ((p2 > r) || (p1 <= mid && nums[p1] <= nums[p2])) {
                temp[k++] = nums[p1++];
            } else {
                temp[k++] = nums[p2++];
            }
        }
        for (int i = l; i <= r; i++) nums[i] = temp[i];
        return ans;
    }
    int countRangeSum(vector<int>& nums, int lower, int upper) {
        this->lower = lower;
        this->upper = upper;
        vector<long long> sum(nums.size() + 1);
        while (temp.size() < sum.size()) temp.push_back(0);
        sum[0] = 0;
        for (int i = 0; i < nums.size(); i++) sum[i + 1] = sum[i] + nums[i];
        return mergeSort(sum, 0, sum.size() - 1);
    }
};
```

7. 计算右侧小于当前元素的个数 (难)

```
class Solution {
public:
    struct Data {
        Data(int val, int ind, int cnt) : val(val), ind(ind), cnt(cnt) {}
        bool operator>(const Data &a) const {
            return val > a.val;
        }
        int val, ind, cnt;
    };
    vector<Data> temp;
    void mergeSort(vector<Data> &arr, int l, int r) {
        if (l >= r) return ;
        int mid = (l + r) >> 1;
        mergeSort(arr, l, mid);
        mergeSort(arr, mid + 1, r);
        int k = l, p1 = l, p2 = mid + 1;
        while (p1 <= mid || p2 <= r) {
            if ((p2 > r) || (p1 <= mid && arr[p1] > arr[p2])) {
                arr[p1].cnt += (r - p2 + 1);
                temp[k++] = arr[p1++];
            } else {
                temp[k++] = arr[p2++];
            }
        }
        for (int i = l; i <= r; i++) arr[i] = temp[i];
        return ;
    }
    vector<int> countSmaller(vector<int>& nums) {
        while (temp.size() < nums.size()) temp.push_back(Data{0, 0, 0});
        vector<Data> arr;
        for (int i = 0; i < nums.size(); i++) {
            arr.push_back(Data{nums[i], i, 0});
        }
        mergeSort(arr, 0, arr.size() - 1);
        vector<int> ret(nums.size());
        for (int i = 0; i < arr.size(); i++) {
            ret[arr[i].ind] = arr[i].cnt;
        }
        return ret;
    }
};
```

8. 最大子序和 : dfs 分治讨论

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        vector<long long> sum;
        sum.push_back(0);
        for (auto x : nums) sum.push_back(sum[sum.size() - 1] + x);
        long long ans = sum[1];
        for (long long pre = 0, i = 1; i < sum.size(); i++) {
            ans = max(sum[i] - pre, ans);
            pre = min(sum[i], pre);
        }
        return ans;
    }
};
```

9. 首个共同祖先

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if (root == nullptr) return nullptr;
        if (root == p || root == q) return root;
        TreeNode *l = lowestCommonAncestor(root->left, p, q);
        TreeNode *r = lowestCommonAncestor(root->right, p, q);
        if (l != nullptr && r != nullptr) return root;
        if (l != nullptr && r == nullptr) return l;
        return r;
    }
};
```

10. 层数最深叶子节点的和

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    void getAns(TreeNode *root, int k, int& max_k, int& ans) {
        if (root == NULL) return ;
        if (k == max_k) ans += root->val;
        else if (k > max_k) {
            max_k = k, ans = root->val;
        }
        getAns(root->left,  k + 1, max_k, ans);
        getAns(root->right, k + 1, max_k, ans);
        return ;
    }
    int deepestLeavesSum(TreeNode* root) {
        int ans = 0, max_k = 0;
        getAns(root, 0, max_k, ans);
        return ans;
    }
};
```