1. 排序数组

```
class Solution {
public:
    vector<int> sortArray(vector<int>& nums) {
        int l = nums[0], r = nums[0];
        for (auto x : nums) l = min(l, x), r = max(r, x);
        int *raw_cnt = new int[r - l + 5], *cnt = raw_cnt - l;
        memset(raw_cnt, 0, sizeof(int) * (r - l + 5));
        for (auto x : nums) cnt[x] += 1;
        vector<int> ret(nums.size());
        int k = 0;
        for (int i = l; i <= r; i++) {
            if (cnt[i] == 0) continue;
            for (int j = 0; j < cnt[i]; j++) ret[k++] = i;
        }
        delete[] raw_cnt;
        return ret;
    }
};
```

2. 调整数组顺序使奇数位于偶数前面

```
class Solution {
public:
    vector<int> exchange(vector<int>& nums) {
        if (nums.size() == 0) return nums;
        int x = 0, y = nums.size() - 1;
        do {
            while (x < nums.size() && nums[x] % 2) ++x;
            while (y >= 0 && nums[y] % 2 == 0) --y;
            if (x <= y) {
                swap(nums[x], nums[y]);
                ++x, --y;
            }
        } while(x <= y);
        return nums;
    }
};
```

3. 最小K个数

```
class Solution {
public:
    inline int median(int a, int b, int c) {
        if (a > b) swap(a, b);
```

```
        if (a > c) swap(a, c);
        if (b > c) swap(b, c);
        return b;
    }
    void quick_select(vector<int> &arr, int l, int r, int k) {
        if (l >= r) return ;
        int x = l, y = r, z = median(arr[l], arr[(l + r) >> 1], arr[r]);
        do {
            while (arr[x] < z) ++x;
            while (arr[y] > z) --y;
            if (x <= y) {
                swap(arr[x], arr[y]);
                ++x, --y;
            }
        } while (x <= y);
        if (y - l == k - 1) return ; // 左区间数量等于k，直接返回
        if (y - l >= k) quick_select(arr, l, y, k); // 左区间数量大于k，继续
        else quick_select(arr, x, r, k - x + l); // 左区间数量小于k，缩小范围
        return ;
    }
    vector<int> smallestK(vector<int>& arr, int k) {
        if (k == 0) return vector<int>();
        quick_select(arr, 0, arr.size() - 1, k);
        vector<int> ret;
        while (k) ret.push_back(arr[--k]);
        return ret;
    }
};
```

4. 颜色分类 (三路快排)

```
// 通过两次遍历，第一遍遍历首先将 0 归位，第二遍遍历将 1 归位，自然 2 也就被归位了
// 两次遍历实现是十分容易的，也很容易实现，那么我们可不可以一次遍历就将其归位呢?
// 其实这里也利用了我们双指针的思想，我们首先定义两个指针，一个位于数组头部，一个位于数组尾部
// 当我们遇到 0 时则给我们头部指针交换，遇到 2 时，则给尾部指针交换。

class Solution {
public:
    void three_partition(vector<int> &arr, int l, int r, int z) {
        if (l >= r) return ;
        int x = 0, y = r, idx = l;
        while (idx <= y) {
            if (arr[idx] == z) idx++;
            else if (arr[idx] < z) {
                swap(arr[x], arr[idx]);
                x += 1;
                idx += 1;
            } else if (arr[idx] > z) {
```

```
                swap(arr[y], arr[idx]);
                y -= 1;
            }
        }
        return ;
    }
    void sortColors(vector<int>& nums) {
        three_partition(nums, 0, nums.size() - 1, 1);
        return ;
    }

};
```

5. 盛最多水的容器 (双指针思想)

```
class Solution {
public:
    int maxArea(vector<int>& height) {
        int ans = 0, i = 0, j = height.size() - 1;
        while(i < j) {
            ans = max(ans, (j - i) * min(height[i], height[j]));

            if(height[i] < height[j]) i++; // 哪个边短，就舍弃哪个边
            else j--;
        }
        return ans;
    }
};
```

6. 排序链表 (快排/归并)

```
/**
 * 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* sortList(ListNode* head) {
        if (head == NULL) return head;
        int l = head->val, r = head->val;
        ListNode *p = head, *q, *h1 = NULL, *h2 = NULL;
        while (p) l = min(p->val, l), r = max(p->val, r), p = p->next;
```

```
        if (l == r) return head;
        z = (l + r) >> 1;
        p = head;
        while (p) {
            q = p->next;
            if (p->val <= z) {
                p->next = h1;
                h1 = p;
            } else {
                p->next = h2;
                h2 = p;
            }
            p = q;
        }
        h1 = sortList(h1);
        h2 = sortList(h2);
        p = h1;
        while (p->next) p = p->next;
        p->next = h2;
        return h1;
    }
};
```

7. 不同的二叉搜索树 II (二叉树复习)

```
class Solution {
public:
    vector<TreeNode*> dfs(int l, int r) {
        vector<TreeNode*> ans;
        if(l > r) {
            ans.push_back(nullptr);
            return ans;
        }

        for(int i = l; i <= r; i++) {
            vector<TreeNode*> left_tree = dfs(l, i - 1);
            vector<TreeNode*> right_tree = dfs(i + 1, r);

            for(TreeNode* left : left_tree) {
                for(TreeNode* right : right_tree) {
                    TreeNode* t = new TreeNode(i, left, right);
                    ans.push_back(t);
                }
            }
        }
        return ans;
    }

```

```
    vector<TreeNode*> generateTrees(int n) {
        if(n == 0) return vector<TreeNode*> {};
        return dfs(1, n);
    }
};
```

8. 滑动窗口最大值

```
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
        deque<int> q;
        vector<int> ans;

        for(int i = 0; i < nums.size(); i++) {
            while(!q.empty() && nums[i] > nums[q.back()]) {
                q.pop_back();
            }
            q.push_back(i);
            if(i - q.front() == k) q.pop_front();
            if(i + 1 < k) continue;
            ans.push_back(nums[q.front()]);
        }
        return ans;
    }
};
```

9. 字符串解码 (栈复习)

```
class Solution {
public:
    string decodeString(string s) {
        string ret;
        int i = 0;
        while (s[i]) {
            if (s[i] < '0' || s[i] > '9') {
                ret += s[i++];
            } else {
                int num = 0;
                while (s[i] <= '9' && s[i] >= '0') num = num * 10 + s[i++] -
'0';
                i += 1;
                int l = i, r = i, cnt = 1;
                while (cnt) {
                    r += 1;
                    if (s[r] == '[') cnt += 1;
                    else if (s[r] == ']') cnt -= 1;
                }
```

```
                string temp = decodeString(s.substr(l, r - l));
                while (num--) ret += temp;
                i = r + 1;
            }
        }
        return ret;
    }
};
```

10. 用 Rand7() 实现 Rand10()

```
// The rand7() API is already defined for you.
// int rand7();
// @return a random integer in the range 1 to 7

class Solution {
public:
    int rand10() {
        int tmp = rand7(); // 1 - 7
        tmp = (tmp - 1) * 7 + rand7(); // 1 - 49
        if(tmp <= 40) return tmp % 10 + 1; // 丢弃41 - 49
        else return rand10();
    }
};
```