# PDD常见算法题

## 54. 螺旋矩阵

```cpp
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {
        vector<int> ans;
        if (matrix.empty()) return ans;
        int n = matrix.size(), m = matrix[0].size();
        int u = 0, d = n - 1;
        int l = 0, r = m - 1;
        while(true) {
            for(int i = l; i <= r; i++) ans.push_back(matrix[u][i]);
            if (++u > d) break;
            for(int i = u; i <= d; i++) ans.push_back(matrix[i][r]);
            if (--r < l) break;
            for(int i = r; i >= l; i--) ans.push_back(matrix[d][i]);
            if (--d < u) break;
            for(int i = d; i >= u; i--) ans.push_back(matrix[i][l]);
            if (++ l > r) break;
        }
        return ans;
    }
};
```

## 146. LRU缓存机制(双向链表)

```java
public class LRUCache {
    private final int capacity;
    private final Map<Integer, Integer> cache = new LinkedHashMap<>(); // 自带双向链表

    public LRUCache(int capacity) {
        this.capacity = capacity;
    }

    public int get(int key) {
        // 删除 key，并利用返回值判断 key 是否在 cache 中
        Integer value = cache.remove(key);
        if (value != null) { // key 在 cache 中
            cache.put(key, value); // 把 key 移到链表末尾
            return value;
        }
        // key 不在 cache 中
        return -1;
    }

    public void put(int key, int value) {
        // 删除 key，并利用返回值判断 key 是否在 cache 中
        if (cache.remove(key) != null) { // key 在 cache 中
            cache.put(key, value); // 把 key 移到链表末尾
            return;
        }
        // key 不在 cache 中，那么就把 key 插入 cache，插入前判断 cache 是否满了
        if (cache.size() == capacity) { // cache 满了
            Integer oldestKey = cache.keySet().iterator().next();
            cache.remove(oldestKey); // 移除最久未使用 key
        }
        cache.put(key, value);
    }
}
```

```

```

## 56. 合并区间(双指针)

```
class Solution {
public:
    vector<vector<int>> merge(vector<vector<int>>& intervals) {
        vector<vector<int>> ans;
        int n = intervals.size();
        sort(intervals.begin(), intervals.end());
        for (int i = 0; i < n; i++) {
            int left = intervals[i][0];
            int right = intervals[i][1];
            while(i + 1 < n && right >= intervals[i + 1][0]) {
                right = max(right, intervals[i + 1][1]);
                i++;
            }
            ans.push_back({left, right});
        }
        return ans;
    }
};
```

## 3. 无重复字符的最长子串(map)

```cpp
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        if(s.size() == 0) return 0;
        unordered_set<char> lookup;
        int maxStr = 0;
        int left = 0;
        for(int i = 0; i < s.size(); i++){
            while (lookup.find(s[i]) != lookup.end()){
                lookup.erase(s[left]);
                left ++;
            }
            maxStr = max(maxStr,i-left+1);
            lookup.insert(s[i]);
        }
        return maxStr;

    }
};
```

## 53. 最大子数组和(DP)

```cpp
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        if (nums.size() == 0) return 0;
        vector<int> dp(nums.size(), 0); // dp[i]表示包括i之前的最大连续子序列和
        dp[0] = nums[0];
        int result = dp[0];
        for (int i = 1; i < nums.size(); i++) {
            dp[i] = max(dp[i - 1] + nums[i], nums[i]); // 状态转移公式
            if (dp[i] > result) result = dp[i]; // result 保存dp[i]的最大值
        }
        return result;
    }
};
```

## 215. 数组中的第K个最大元素

```cpp
class Solution {
public:
    int quickselect(vector<int> &nums, int l, int r, int k) {
        if (r == l) return nums[k];
        int par = nums[l], i = l - 1, j = r + 1;
        while(i < j) {
            do i++; while(nums[i] < par);
            do j--; while(nums[j] > par);
            if (i < j) swap(nums[i], nums[j]);
        }
        if (k <= j) return quickselect(nums, l, j, k);
        else return quickselect(nums, j + 1, r, k);
    }

    int findKthLargest(vector<int> &nums, int k) {
        int n = nums.size();
        return quickselect(nums, 0, n - 1, n - k);
    }
};
```

## 121. 买卖股票的最佳时机

```cpp
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int ans = 0, minP = INT_MAX;
        for(auto tmp : prices) {
            minP = min(minP, tmp);
            ans = max(ans, tmp - minP);
        }
        return ans;
    }
};
```

## 206. 反转链表

```cpp
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* cur = NULL, *pre = head;
        while (pre != NULL) {
            ListNode* t = pre->next;
            pre->next = cur;
            cur = pre;
            pre = t;
        }
        return cur;
    }
};
```

## 32. 最长有效括号

```cpp
class Solution {
public:
    int longestValidParentheses(string s) {
        int maxans = 0;
        stack<int> stk;
        stk.push(-1);
        for (int i = 0; i < s.length(); i++) {
            if (s[i] == '(') {
                stk.push(i);
            } else {
                // 遇到')'弹出一次
                stk.pop();
                if (stk.empty()) {
                    // 如果为空说明-1被弹出无效 写入无效的下标;
                    stk.push(i);
                } else {
                    // 有效 不断更新最长的长度
                    maxans = max(maxans, i - stk.top());
                }
            }
        }
        return maxans;
    }
};
```

## 46. 全排列

```
class Solution {
public:
    void backtrack(vector<vector<int>>& res, vector<int>& output, int first, int len){
        // 所有数都填完了
        if (first == len) {
            res.emplace_back(output);
            return;
        }
        for (int i = first; i < len; ++i) {
            // 动态维护数组
            swap(output[i], output[first]);
            // 继续递归填下一个数
            backtrack(res, output, first + 1, len);
            // 撤销操作
            swap(output[i], output[first]);
        }
    }
    vector<vector<int>> permute(vector<int>& nums) {
        vector<vector<int> > res;
        backtrack(res, nums, 0, (int)nums.size());
        return res;
    }
};
```

## 4. 寻找两个正序数组的中位数

```
class Solution {
    public static int[] help = new int[20001];// 辅助数组
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int l = 0, r = 0, i = 0;
        // 对两个数组进行和并，且有序
        while (l < nums1.length && r < nums2.length) {
            help[i++] = nums1[l] <= nums2[r] ? nums1[l++] : nums2[r++];
        }

        while(l < nums1.length) {
            help[i++] = nums1[l++];
        }

        while(r < nums2.length) {
            help[i++] = nums2[r++];
        }

        // 计算中位数即可
        double ans = 0;
        int last = i - 1;
        if (last % 2 == 0) {
            // 奇数
            ans =  help[last / 2];
        }else {
            // 偶数
            ans = (double) (help[last/2] + help[last/2 + 1]) / 2;
        }
        return ans;
    }
}
```

## 93. 复原IP地址(模拟拼字符串)

```
class Solution {
    public List<String> restoreIpAddresses(String s) {
        List<String> ret = new ArrayList<>();

        StringBuilder ip = new StringBuilder();

        for(int a = 1 ; a < 4 ; ++ a)
            for(int b = 1 ; b < 4 ; ++ b)
                for(int c = 1 ; c < 4 ; ++ c)
                    for(int d = 1 ; d < 4 ; ++ d)
                    {
                        if(a + b + c + d == s.length() )
                        {
                            int n1 = Integer.parseInt(s.substring(0, a));
                            int n2 = Integer.parseInt(s.substring(a, a+b));
                            int n3 = Integer.parseInt(s.substring(a+b, a+b+c));
                            int n4 = Integer.parseInt(s.substring(a+b+c));
                            if(n1 <= 255 && n2 <= 255 && n3 <= 255 && n4 <= 255)
                            {
                                ip.append(n1).append('.').append(n2)
.append('.').append(n3).append('.').append(n4);
                                if(ip.length() == s.length() + 3)
ret.add(ip.toString());
                                ip.delete(0, ip.length());
                            }
                        }
                    }
        return ret;
    }
}
```

# 94. 二叉树的中序遍历

```cpp
class Solution {
public:
    void dfs(TreeNode *root, vector<int> &data) {
        if(!root) return;
        dfs(root -> left, data);
        data.emplace_back(root -> val);
        dfs(root -> right, data);
    }
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> ans;
        dfs(root, ans);
        return ans;
    }
};
```

```cpp
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> res;
        stack<TreeNode*> stk;
        while (root != nullptr || !stk.empty()) {
            while (root != nullptr) {
                stk.push(root);
                root = root->left;
            }
            root = stk.top();
            stk.pop();
            res.push_back(root->val);
            root = root->right;
        }
        return res;
    }
};
```

## 148. 排序链表

```cpp
/**
 * 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) return head;
        vector<ListNode*> data;
        auto tmp = head;
        while(tmp != nullptr) {
            data.emplace_back(tmp);
            tmp = tmp -> next;
        }
        sort(data.begin(), data.end(), [&](ListNode *a, ListNode *b){
            return a -> val < b -> val;
        });
        for(int i = 0; i + 1 < data.size(); i++) {
            // cout << i << endl;
            data[i] -> next = data[i + 1];
        }
        data[data.size() - 1] -> next = nullptr;
        return data[0];
    }
};
```

## 8. 字符串转换整数 (atoi)

```cpp
class Solution {
public:
    int myAtoi(string str) {
         if (str.empty()) return 0;
        int index = 0, n = str.size(), sign = 1, res = 0;
        // 处理前置空格
        while (index < n && str[index] == ' ') {
            ++index;
        }
        // 处理符号
        if (index < n && (str[index] == '+' || str[index] == '-')) {
            sign = str[index++] == '+' ? 1 : -1;
        }
        // 处理数字
        while (index < n && isdigit(str[index])) {
            int digit = str[index] - '0';
            // 判断是否溢出
            if (res > (INT_MAX - digit) / 10) {
                return sign == 1 ? INT_MAX : INT_MIN;
            }
            res = res * 10 + digit;
            ++index;
        }
        return res * sign;
    }
};
```

## 322. 零钱兑换(dp)

```
class Solution {
public:
    int coinChange(vector<int>& coins, int amount) {
        int Max = amount + 1;
        vector<int> dp(amount + 1, Max);
        dp[0] = 0;
        for (int i = 1; i <= amount; ++i) {
            for (int j = 0; j < (int)coins.size(); ++j) {
                if (coins[j] <= i) {
                    dp[i] = min(dp[i], dp[i - coins[j]] + 1);
                }
            }
        }
        return dp[amount] > amount ? -1 : dp[amount];
    }
};
```

## 628. 三个数的最大乘积(排序)

```cpp
class Solution {
public:
    int maximumProduct(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        int n = nums.size();
        return max(nums[0] * nums[1] * nums[n - 1], nums[n - 3] * nums[n - 2] * nums[n - 1]);
    }
};
```

## 662. 二叉树最大宽度

```cpp
class Solution {
public:
    int widthOfBinaryTree(TreeNode* root) {
        unsigned long long res = 1;
        vector<pair<TreeNode *, unsigned long long>> arr;
        arr.emplace_back(root, 1L);
        while (!arr.empty()) {
            vector<pair<TreeNode *, unsigned long long>> tmp;
            for (auto &[node, index] : arr) {
                if (node->left) {
                    tmp.emplace_back(node->left, index * 2);
                }
                if (node->right) {
                    tmp.emplace_back(node->right, index * 2 + 1);
                }
            }
            res = max(res, arr.back().second - arr[0].second + 1);
            arr = move(tmp);
        }
        return res;
    }
};
```

## 92. 反转链表 II

```cpp
class Solution {
public:
    ListNode *reverseBetween(ListNode *head, int left, int right) {
        // 设置 dummyNode 是这一类问题的一般做法
        ListNode *dummyNode = new ListNode(-1);
        dummyNode->next = head;
        ListNode *pre = dummyNode;
        for (int i = 0; i < left - 1; i++) {
            pre = pre->next;
        }
        ListNode *cur = pre->next;
        ListNode *next;
        for (int i = 0; i < right - left; i++) {
            next = cur->next;
            cur->next = next->next;
            next->next = pre->next;
            pre->next = next;
        }
        return dummyNode->next;
    }
};
```

## 102. 二叉树的层序遍历

```cpp
/**
 * 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:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> ans;
        if(root == nullptr) return ans;
        queue<TreeNode *> que;
        que.push(root);
        while(!que.empty()) {
            int n = que.size();
            vector<int> cur;
            for(int i = 0; i < n; i++) {
                auto tmp = que.front(); que.pop();
                cur.emplace_back(tmp -> val);
                if (tmp -> left != nullptr) que.push(tmp -> left);
                if (tmp -> right != nullptr) que.push(tmp -> right);
            }
            ans.emplace_back(cur);
        }
        return ans;
    }
};
```

## 43. 字符串相乘

```
class Solution {
public:
    string multiply(string num1, string num2) {
        if (num1 == "0" || num2 == "0") {
            return "0";
        }
        int m = num1.size(), n = num2.size();
        auto ansArr = vector<int>(m + n);
        // 从最右侧开始乘
        for (int i = m - 1; i >= 0; i--) {
            int x = num1.at(i) - '0';
            for (int j = n - 1; j >= 0; j--) {
                int y = num2.at(j) - '0';
                ansArr[i + j + 1] += x * y;
            }
        }
        // 向前进位
        for (int i = m + n - 1; i > 0; i--) {
            ansArr[i - 1] += ansArr[i] / 10;
            ansArr[i] %= 10;
        }
        // 忽略头部为0的
        int index = ansArr[0] == 0 ? 1 : 0;
        string ans;
        while (index < m + n) {
            ans.push_back(ansArr[index]);
            index++;
        }
        for (auto &c: ans) {
            // 转为字符串
            c += '0';
        }
        return ans;
    }
};
```

## 540. 有序数组中的单一元素

```cpp
class Solution {
public:
    int singleNonDuplicate(vector<int>& nums) {
        int low = 0, high = nums.size() - 1;
        while (low < high) {
            int mid = (high - low) / 2 + low;
            // 向前移动到偶数位置
            mid -= mid & 1;
            if (nums[mid] == nums[mid + 1]) {
                // 相等说明l-mid下标正确
                low = mid + 2;
            } else {
                high = mid;
            }
        }
        return nums[low];
    }
};

```

## 752. 打开转盘锁

```cpp
class Solution {
public:
    int openLock(vector<string>& deadends, string target) {
        if (target == "0000") {
            return 0;
        }

        unordered_set<string> dead(deadends.begin(), deadends.end());
        if (dead.count("0000")) {
            return -1;
        }

        auto num_prev = [](char x) -> char {
            return (x == '0' ? '9' : x - 1);
        };
        auto num_succ = [](char x) -> char {
            return (x == '9' ? '0' : x + 1);
        };

        // 枚举 status 通过一次旋转得到的数字
        auto get = [&](string& status) -> vector<string> {
            vector<string> ret;
            for (int i = 0; i < 4; ++i) {
                char num = status[i];
                status[i] = num_prev(num);
                ret.push_back(status);
                status[i] = num_succ(num);
                ret.push_back(status);
                status[i] = num;
            }
            return ret;
        };

        queue<pair<string, int>> q;
        q.emplace("0000", 0);
        unordered_set<string> seen = {"0000"};

        while (!q.empty()) {
            auto [status, step] = q.front();
            q.pop();
            for (auto&& next_status: get(status)) {
                if (!seen.count(next_status) && !dead.count(next_status)) {
                    if (next_status == target) {
                        return step + 1;
                    }
                    q.emplace(next_status, step + 1);
                    seen.insert(move(next_status));
                }
            }
        }

        return -1;
    }
};
```

## 169. 多数元素

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int half = nums.size() / 2;
        map<int, int> cnt;
        for(auto n : nums) {
            cnt[n] ++;
            if (cnt[n] > half) {
                return n;
            }
        }
        return -1;
    }
};
```

## 165. 比较版本号

```
class Solution {
public:
    int compareVersion(string version1, string version2) {
        int n = version1.length(), m = version2.length();
        int i = 0, j = 0;
        while (i < n || j < m) {
            long long x = 0;
            for (; i < n && version1[i] != '.'; ++i) {
                x = x * 10 + version1[i] - '0';
            }
            ++i; // 跳过点号
            long long y = 0;
            for (; j < m && version2[j] != '.'; ++j) {
                y = y * 10 + version2[j] - '0';
            }
            ++j; // 跳过点号
            if (x != y) {
                return x > y ? 1 : -1;
            }
        }
        return 0;
    }
};
```

## 155. 最小栈

```
class MinStack {
public:
    stack<int> xStack;
    stack<int> mStack;
    MinStack() {

    }

    void push(int val) {
        xStack.push(val);
        if(!mStack.empty()) {
            mStack.push(min(mStack.top(), val));
        } else {
            mStack.push(val);
        }

    }

    void pop() {
        xStack.top(); xStack.pop();
        mStack.pop();
    }

    int top() {
        return xStack.top();
    }

    int getMin() {
        return mStack.top();
    }
};

/**
 * Your MinStack object will be instantiated and called as such:
 * MinStack* obj = new MinStack();
 * obj->push(val);
 * obj->pop();
 * int param_3 = obj->top();
 * int param_4 = obj->getMin();
 */
```

## 5. 最长回文子串

```
class Solution {
public:
    string longestPalindrome(string s) {
        int n = s.size();
        vector<vector<int>> dp(n, vector<int>(n));
        string ans;
        for (int l = 0; l < n; ++l) {
            for (int i = 0; i + l < n; ++i) {
                int j = i + l;
                if (l == 0) {
                    dp[i][j] = 1;
                } else if (l == 1) {
                    dp[i][j] = (s[i] == s[j]);
                } else {
                    dp[i][j] = (s[i] == s[j] && dp[i + 1][j - 1]);
                }
                if (dp[i][j] && l + 1 > ans.size()) {
                    ans = s.substr(i, l + 1);
                }
            }
        }
        return ans;
    }
};
```

## 面试题 17.24. 最大子矩阵

```
class Solution {
    public int[] getMaxMatrix(int[][] matrix) {
        int n = matrix.length, m = matrix[0].length;
        //二维前缀和
        int[][] preSum = new int[n + 1][m + 1];
        for(int i = 1; i < n + 1; i ++) {
            for(int j = 1; j < m + 1; j ++) {
                preSum[i][j] = matrix[i - 1][j - 1] + preSum[i - 1][j] + preSum[i][j - 1] - preSum[i - 1][j - 1];
            }
        }
        //开始最大子序和
        int gobalMax = Integer.MIN_VALUE;
        int[] ret = new int[4];
        //先固定上下两条边
        for(int top = 0; top < n; top ++) {
            for(int bottom = top; bottom < n; bottom ++) {
                int localMax = 0, left = 0;
                //然后从左往右一遍扫描找最大子序和
                for(int right = 0; right < m; right ++) {
                    //利用presum快速求出localMax
                    localMax = preSum[bottom + 1][right + 1] + preSum[top][left] - preSum[bottom + 1][left] - preSum[top][right + 1];
                    //如果比gobal大，更新
                    if(gobalMax < localMax) {
```

```java
                        gobalMax = localMax;
                        ret[0] = top;
                        ret[1] = left;
                        ret[2] = bottom;
                        ret[3] = right;
                    }
                    //如果不满0，前面都舍弃，从新开始计算，left更新到right+1，right下一轮递增之后left==right
                    if(localMax < 0) {
                        localMax = 0;
                        left = right + 1;
                    }
                }
            }
        }
        return ret;
    }
}
```

## 面试题 16.25. LRU缓存

```java
class LRUCache {

    Map<Integer,Integer> cache = null;

    // 这个是匿名内部类
    // LinkedHashMap的三个构造函数分别是初始容量，扩容因子和是否移除旧的元素
    public LRUCache(int capacity) {
        cache = new LinkedHashMap<>(capacity,0.75f,true){
            // 必须覆盖该方法来保证移除旧的元素
            // 返回false，不删除
            @Override
            protected boolean removeEldestEntry(Map.Entry eldest) {
                if(this.size() > capacity){
                    return true;
                }
                return false;
            }
        };
    }

    public int get(int key) {
        Integer v = this.cache.get(key);
        return v==null?-1:v.intValue();
    }

    public void put(int key, int value) {
        this.cache.put(key,value);
    }
}

```

## 39. 组合总和

```
class Solution {
public:
    void dfs(vector<int>& candidates, int target, vector<vector<int>>& ans, vector<int>& combine, int idx) {
        if (idx == candidates.size()) {
            return;
        }
        if (target == 0) {
            ans.emplace_back(combine);
            return;
        }
        // 直接跳过
        dfs(candidates, target, ans, combine, idx + 1);
        // 选择当前数
        if (target - candidates[idx] >= 0) {
            combine.emplace_back(candidates[idx]);
            dfs(candidates, target - candidates[idx], ans, combine, idx);
            combine.pop_back();
        }
    }

    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        vector<vector<int>> ans;
        vector<int> combine;
        dfs(candidates, target, ans, combine, 0);
        return ans;
    }
};
```

## 15. 三数之和

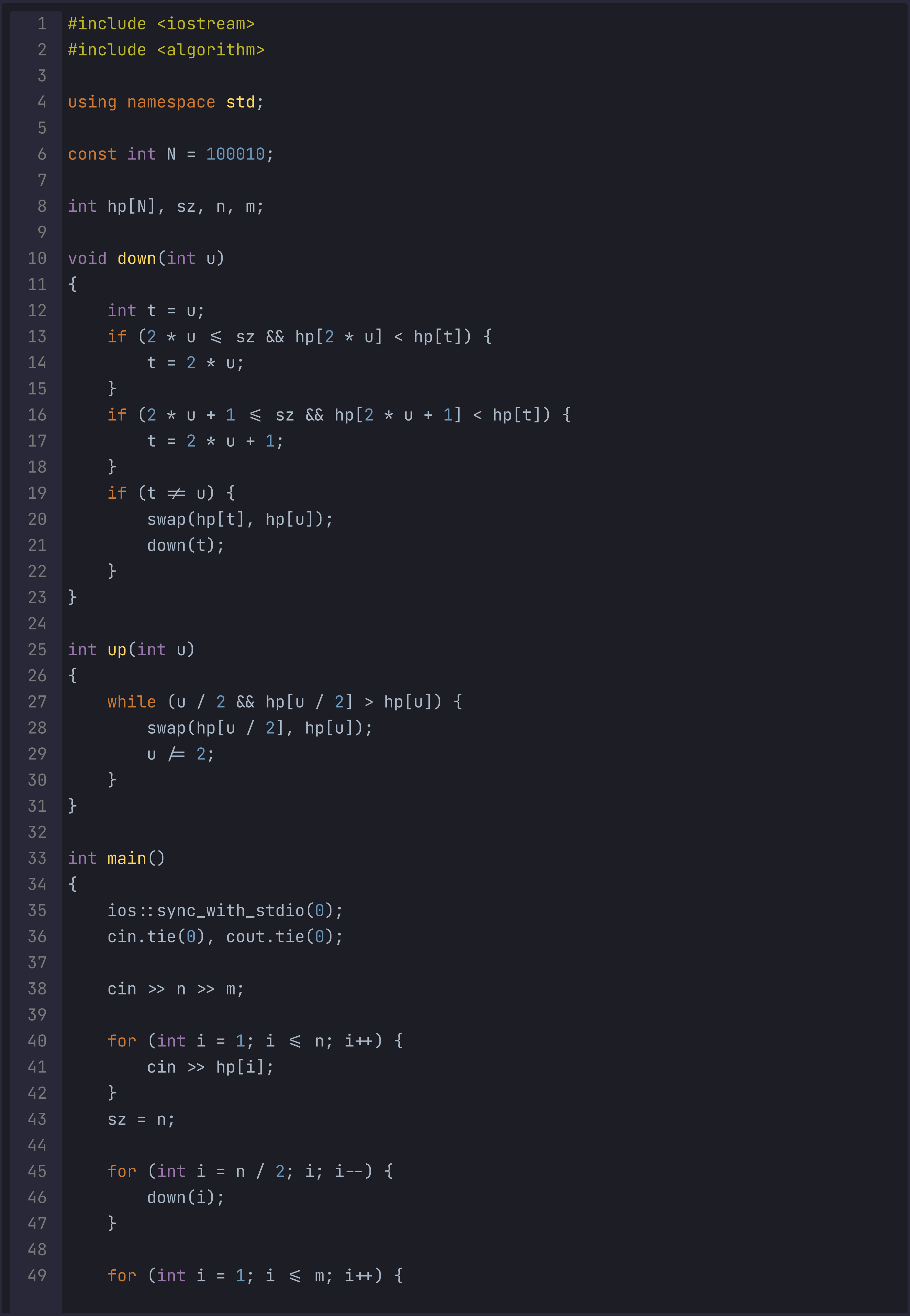
```cpp
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        vector<vector<int>> ans;
        int n = nums.size();
        for(int i = 0; i < n - 2; i++) {
            if (i && nums[i] == nums[i - 1]) continue;
            int j = i + 1, k = n - 1;
            while(j < k) {
                int x = nums[i] + nums[j] + nums[k];
                if (x < 0) j++;
                else if (x > 0) k--;
                else {
                    ans.push_back({nums[i] , nums[j++] , nums[k--]});
                    while(j < n && nums[j] == nums[j - 1]) j++;
                    while(j < k && nums[k] == nums[k + 1]) k--;
                }
            }
        }
        return ans;
    }
};
```

## 21. 合并两个有序链表

```cpp
/**
 * 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* mergeTwoLists(ListNode* list1, ListNode* list2) {
        ListNode* dunmmy = new ListNode(-1);
        ListNode* dunmmyNext = dunmmy;
        while (list1 != nullptr && list2 != nullptr) {
            if (list1 -> val >= list2 -> val) {
                dunmmyNext -> next = list2;
                list2 = list2 -> next;
            } else {
                dunmmyNext -> next = list1;
                list1 = list1 -> next;
            }
            dunmmyNext = dunmmyNext -> next;
        }
        if(list1 != nullptr) {
            dunmmyNext -> next = list1;
        }
        if(list2 != nullptr) {
            dunmmyNext -> next = list2;
        }
        return dunmmy -> next;
    }
};
```

## 补充题6． 手撕堆排序

```
class Solution {
public:
    void quickSort(vector<int> &nums, int l, int r) {
        if (l == r) return;
        int mid = ((l - r) >> 1) + r;
        int i = l - 1, j = r + 1;
        int par = nums[mid];
        while(i < j) {
            do {
                i++;
            } while(nums[i] < par);
            do {
                j--;
            } while(nums[j] > par);
            if(i < j) swap(nums[i], nums[j]);
        }
        quickSort(nums, l, j);
        quickSort(nums, j + 1, r);
    }
    vector<int> sortArray(vector<int>& nums) {
        quickSort(nums, 0, nums.size() - 1);
        return nums;
    }
};
```



```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010;

int hp[N], sz, n, m;

void down(int u)
{
    int t = u;
    if (2 * u <= sz && hp[2 * u] < hp[t]) {
        t = 2 * u;
    }
    if (2 * u + 1 <= sz && hp[2 * u + 1] < hp[t]) {
        t = 2 * u + 1;
    }
    if (t != u) {
        swap(hp[t], hp[u]);
        down(t);
    }
}

int up(int u)
{
    while (u / 2 && hp[u / 2] > hp[u]) {
        swap(hp[u / 2], hp[u]);
        u /= 2;
    }
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0), cout.tie(0);

    cin >> n >> m;

    for (int i = 1; i <= n; i++) {
        cin >> hp[i];
    }
    sz = n;

    for (int i = n / 2; i; i--) {
        down(i);
    }

    for (int i = 1; i <= m; i++) {
        cout << hp[1] << ' ';
        swap(hp[1], hp[sz--]);
        down(1);
    }

    return 0;
}
```

## 239. 滑动窗口最大值

```
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
        if(nums.size() == 0 || k == 0) return {};
        deque<int> deque;
        vector<int> res(nums.size() - k + 1);
        // 未形成窗口
        for(int i = 0; i < k; i++) {
            while(!deque.empty() && deque.back() < nums[i])
                deque.pop_back();
            deque.push_back(nums[i]);
        }
        res[0] = deque.front();
        // 形成窗口后
        for(int i = k; i < nums.size(); i++) {
            if(deque.front() == nums[i - k])
                // 最前面的等于当前值 所以可以直接弹出
                deque.pop_front();
            while(!deque.empty() && deque.back() < nums[i])
                // 下标值更大并且值更大 可以弹出末位
                deque.pop_back();
            deque.push_back(nums[i]);
            res[i - k + 1] = deque.front();
        }
        return res;
    }
};
```

## 179. 最大数

```cpp
class Solution {
public:
    static bool cmp(int a, int b) {
        string a_str = to_string(a);
        string b_str = to_string(b);
        string x = a_str + b_str, y = b_str + a_str;
        return x > y;
    }
    string largestNumber(vector<int>& nums) {
        sort(nums.begin(), nums.end(),  cmp);
        string ans = "";
        for (auto num : nums) {
            ans += to_string(num);
        }
        // 去除末尾0
        while(ans[0] == '0' && ans.length() > 1) {
            ans.erase(ans.begin());
        }
        return ans;
    }
};
```

## 33. 搜索旋转排序数组

```cpp
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int left = 0, right = nums.size() - 1;
        while (left <= right) {
            int mid = (left + right) >> 1;
            if (nums[mid] == target) return mid;
            if (nums[left] <= nums[mid]) {
                // left 到 mid 是顺序区间
                (target >= nums[left] && target < nums[mid]) ? right = mid - 1 : left = mid + 1;
            }
            else {
                // mid 到 right 是顺序区间
                (target > nums[mid] && target <= nums[right]) ? left = mid + 1 : right = mid - 1;
            }
        }
        return -1;
    }
};
```

## 1143. 最长公共子序列

```cpp
class Solution {
public:
    int longestCommonSubsequence(string text1, string text2) {
        int m = text1.length(), n = text2.length();
        // dp[i][j]表示前text1前i个字符，text2前j个字符的最长公共子序列
        vector<vector<int>> dp(m + 1, vector<int>(n + 1));
        for (int i = 1; i <= m; i++) {
            char c1 = text1[i - 1];
            for (int j = 1; j <= n; j++) {
                char c2 = text2[j - 1];
                if (c1 == c2) {
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                } else {
                    // i前面 或者j
                    dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
                }
            }
        }
        return dp[m][n];
    }
};
```

## 78. 子集

```cpp
class Solution {
public:
    void dfs(vector<vector<int>> &ans, vector<int>& nums,
        vector<int> cur, int start) {
        if (nums.size() < start) return;
        ans.emplace_back(cur);
        for(int i = start; i < nums.size(); i++) {
            cur.emplace_back(nums[i]);
            dfs(ans, nums, cur, i + 1);
            cur.pop_back();
        }
    }
    vector<vector<int>> subsets(vector<int>& nums) {
        vector<vector<int>> ans;
        dfs(ans, nums, {}, 0);
        return ans;
    }
};
```

## 226. 翻转二叉树

```
/**
 * 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:
    TreeNode* invertTree(TreeNode* root) {
        if (root == nullptr) {
            return nullptr;
        }
        auto tmp = root -> left;
        root -> left = invertTree(root -> right);
        root -> right = invertTree(tmp);
        return root;
    }
};
```