Amazon Coding Interview Questions & Answers (Complete Set in C++)

Complexity Legend: - O(1): Constant time - O(n): Linear time - O(n log n): Linearithmic - O(n^2): Quadratic - O(log n): Logarithmic

1. Arrays & Strings - Two Sum

Complexity: O(n) time, O(n) space

```
vector<int> twoSum(vector<int>& nums, int target) {
    unordered_map<int,int> mp;
    for(int i=0;i<nums.size();i++) {
        int need = target - nums[i];
        if(mp.count(need)) return {mp[need], i};
        mp[nums[i]] = i;
    }
    return {};
}
```

Longest Substring Without Repeating Characters

Complexity: O(n) time, O(n) space

```
int lengthOfLongestSubstring(string s) {
    unordered_map<char,int> mp;
    int l=0, ans=0;
    for(int r=0;r<s.size();r++) {
        if(mp.count(s[r])) l = max(l, mp[s[r]]+1);
        mp[s[r]] = r;
        ans = max(ans, r-l+1);
    }
    return ans;
}
```

Trapping Rain Water

Complexity: O(n) time, O(1) space

```
int trap(vector<int>& h) {
    int n=h.size(), l=0, r=n-1, leftMax=0, rightMax=0, ans=0;
    while(l<r){
        if(h[l]<h[r]){
            leftMax=max(leftMax,h[l]);
            ans+=leftMax-h[l];
            l++;
        } else {
            rightMax=max(rightMax,h[r]);
            ans+=rightMax-h[r];
            r--;
        }
    }
    return ans;
}
```

2. Hashing & LRU - LRU Cache

Complexity: O(1) get/put

```
class LRUCache {
    int cap;
    list<pair<int,int>> dll;
    unordered_map<int, list<pair<int,int>>::iterator> mp;
public:
    LRUCache(int capacity) { cap=capacity; }
    int get(int key) {
        if(!mp.count(key)) return -1;
        auto it=mp[key];
        int val=it->second;
        dll.erase(it);
        dll.push_front({key,val});
        mp[key]=dll.begin();
        return val;
    }
    void put(int key, int value) {
        if(mp.count(key)) dll.erase(mp[key]);
        dll.push_front({key,value});
        mp[key]=dll.begin();
        if(dll.size()>cap) {
            auto last=dll.back();
            mp.erase(last.first);
            dll.pop_back();
        }
    }
};
```

Top K Frequent Elements

Complexity: O(n log n)

```
vector<int> topKFrequent(vector<int>& nums, int k) {
    unordered_map<int,int> freq;
    for(int n: nums) freq[n]++;
    priority_queue<pair<int,int>> pq;
    for(auto &p: freq) pq.push({p.second, p.first});
    vector<int> res;
    while(k--) { res.push_back(pq.top().second); pq.pop(); }
    return res;
}
```

3. Linked List - Detect Cycle

Complexity: O(n) time, O(1) space

```
ListNode *detectCycle(ListNode *head) {
    ListNode *slow=head, *fast=head;
    while(fast && fast->next){
        slow=slow->next;
        fast=fast->next->next;
        if(slow==fast){
            slow=head;
            while(slow!=fast){ slow=slow->next; fast=fast->next; }
            return slow;
        }
    }
    return NULL;
}
```

Merge K Sorted Lists

Complexity: O(N log k)

```
ListNode* mergeKLists(vector<ListNode*>& lists) {
    auto cmp=[](ListNode* a, ListNode* b){ return a->val>b->val; };
    priority_queue<ListNode*, vector<ListNode*>, decltype(cmp)> pq(cmp);
    for(auto l: lists) if(l) pq.push(l);
    ListNode dummy(0), *cur=&dummy;
    while(!pq.empty()){
        auto node=pq.top(); pq.pop();
        cur->next=node; cur=cur->next;
        if(node->next) pq.push(node->next);
    }
    return dummy.next;
}
```

4. Stacks & Queues - Min Stack

Complexity: O(1) operations

```
class MinStack {
    stack<int> s, minS;
public:
    void push(int val) {
        s.push(val);
        if(minS.empty() || val<=minS.top()) minS.push(val);
    }
    void pop() {
        if(s.top()==minS.top()) minS.pop();
        s.pop();
    }
    int top() { return s.top(); }
    int getMin() { return minS.top(); }
};
```

Sliding Window Maximum

Complexity: O(n)

```
vector<int> maxSlidingWindow(vector<int>& nums, int k) {
    deque<int> dq;
    vector<int> res;
    for(int i=0;i<nums.size();i++){
        while(!dq.empty() && dq.front()<=i-k) dq.pop_front();
        while(!dq.empty() && nums[dq.back()]<=nums[i]) dq.pop_back();
        dq.push_back(i);
        if(i>=k-1) res.push_back(nums[dq.front()]);
    }
    return res;
}
```

5. Trees & Graphs - Lowest Common Ancestor

Complexity: O(h), h = tree height

```
TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
    if(!root) return NULL;
    if(p->val<root->val && q->val<root->val) return lowestCommonAncestor(root->left,p,q);
    if(p->val>root->val && q->val>root->val) return lowestCommonAncestor(root->right,p,q);
    return root;
}
```

Number of Islands

Complexity: O(m*n)

```
void dfs(vector<vector<char>>& g,int i,int j){
    if(i<0||j<0||i>=g.size()||j>=g[0].size()||g[i][j]=='0') return;
    g[i][j]='0';
    dfs(g,i+1,j); dfs(g,i-1,j); dfs(g,i,j+1); dfs(g,i,j-1);
}
int numIslands(vector<vector<char>>& g) {
    int cnt=0;
    for(int i=0;i<g.size();i++)
        for(int j=0;j<g[0].size();j++)
            if(g[i][j]=='1'){ cnt++; dfs(g,i,j); }
    return cnt;
}
```

6. Dynamic Programming - Edit Distance

Complexity: O(m*n)

```
int minDistance(string w1, string w2) {
    int m=w1.size(), n=w2.size();
    vector<vector<int>> dp(m+1, vector<int>(n+1,0));
    for(int i=0;i<=m;i++) dp[i][0]=i;
```

```
        for(int j=0;j<=n;j++) dp[0][j]=j;
        for(int i=1;i<=m;i++){
            for(int j=1;j<=n;j++){
                if(w1[i-1]==w2[j-1]) dp[i][j]=dp[i-1][j-1];
                else dp[i][j]=1+min({dp[i-1][j], dp[i][j-1], dp[i-1][j-1]});
            }
        }
        return dp[m][n];
    }
```

7. Bit Manipulation - Single Number

Complexity: O(n) time, O(1) space

```
    int singleNumber(vector<int>& nums) {
        int x=0;
        for(int n: nums) x^=n;
        return x;
    }
```

Count Set Bits

Complexity: O(log n)

```
    int hammingWeight(uint32_t n) {
        int cnt=0;
        while(n){ cnt+=n&1; n>>=1; }
        return cnt;
    }
```

8. System-like - URL Shortener (Core Idea)

Complexity: O(1) encode/decode average

```
    class Codec {
        unordered_map<string,string> long2short, short2long;
        string base="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
        string getCode(){
            string s; for(int i=0;i<6;i++) s+=base[rand()%62]; return s;
        }
    public:
        string encode(string longUrl) {
            if(long2short.count(longUrl)) return long2short[longUrl];
            string code=getCode();
            while(short2long.count(code)) code=getCode();
            long2short[longUrl]=code;
            short2long[code]=longUrl;
            return code;
        }
        string decode(string shortUrl) { return short2long[shortUrl]; }
    };
```