

TECHNICAL TRAINING DSA - CODING PRACTICE PROBLEMS

Name: Sachin A
Dept: CSBS
Date: 21-11-2024

Valid Palindrome

CODE:

```
bool isPalindrome(string s) {  
    int start=0;  
    int end=s.size()-1;  
    while(start<=end){  
        if(!isalnum(s[start])){start++; continue;}  
        if(!isalnum(s[end])){end--;continue;}  
        if(tolower(s[start])!=tolower(s[end]))return false;  
        else{  
            start++;  
            end--;  
        }  
    }  
    return true;  
}
```

Time Complexity: $O(n)$
Space Complexity: $O(1)$

Is Subsequence

CODE:

```
bool isSubsequence(string s, string t) {  
    int n=t.size();  
    int end=s.size();  
    int i=0;  
    int j=0;  
    int count=0;  
    while(i<end && j<n){  
        if(s[i]==t[j]){
```

```

i++;
count++;
}
j++;
}
if(count==end){
return true;
}
return false;
}

```

Time Complexity: $O(n)$
Space Complexity: $O(1)$

Two Sum II - Input Array Is Sorted

CODE

```

vector<int> twoSum(vector<int>& numbers, int target) {
int n=numbers.size();
int i=0;
int j=n-1;
vector<int> ans;
while(i<n){
int total=numbers[i]+numbers[j];
if(total==target){
ans.push_back(i+1);
ans.push_back(j+1);
break;
}
else if(total>target){
j--;
}
else{
i++;
}
}
return ans;
}

```

Time Complexity: $O(n \log n)$
Space Complexity: $O(1)$

Container With Most Water

CODE:

```
int maxArea(vector<int>& nums) {
    int i=0;
    int j=nums.size()-1;
    int ans=0;
    while(i<j){
        ans=max(min(nums[i],nums[j])*(j-i),ans);
        if(nums[i]>nums[j]) j--;
        else i++;
    }
    return ans;
}
```

Time Complexity: $O(n)$
Space Complexity: $O(1)$

3Sum

CODE:

```
vector<vector<int>> threeSum(vector<int>& nums) {
    int n = nums.size();
    sort(nums.begin(), nums.end());
    vector<vector<int>> output;
    for(int i=0; i<n-1; i++){
        int low = i+1, high = n-1;
        while(low < high){
            if(nums[i] + nums[low] + nums[high] < 0){
                low++;
            }
            else if(nums[i] + nums[low] + nums[high] > 0){
                high--;
            }
            else{

```

```

        output.push_back({nums[i], nums[low], nums[high]});
        int tempIndex1 = low, tempIndex2 = high;
        while(low < high && nums[low] == nums[tempIndex1])
low++;
        while(low < high && nums[high] == nums[tempIndex2])
high--;
    }
}
while(i+1 < n && nums[i] == nums[i+1]) i++;
}
return output;
}

```

Time Complexity: $O(n \log n)$

Space Complexity: $O(1)$

Minimum Size Subarray Sum

CODE:

```

int minSubArrayLen(int target, vector<int>& nums) {
    int start=0;
    int n=nums.size();
    int minn=INT_MAX;
    int sum=0;
    for(int i=0;i<n;i++){
        sum+=nums[i];
        while(sum>=target){
            minn=min(minn,i-start+1);
            sum-=nums[start];
            start++;
        }
    }
    if(minn==INT_MAX) return 0;
    else return minn;
}

```

Time Complexity: $O(n)$

Space Complexity: $O(1)$

Longest Substring Without Repeating Characters

CODE:

```
int lengthOfLongestSubstring(string s) {
    int n = s.length();
    int maxLength = 0;
    unordered_set<char> charSet;
    int left = 0;
    for (int right = 0; right < n; right++) {
        if (charSet.count(s[right]) == 0) {
            charSet.insert(s[right]);
            maxLength = max(maxLength, right - left + 1);
        } else {
            while (charSet.count(s[right])) {
                charSet.erase(s[left]);
                left++;
            }
            charSet.insert(s[right]);
        }
    }
    return maxLength;
}
```

Time Complexity: $O(n)$
Space Complexity: $O(n)$

*Valid Paranthesis***CODE:**

```
bool isValid(string s) {
    stack<char> st;
    for(char x:s){
        if(!st.empty()){
            if(x=='') && st.top()=='(') st.pop();
            else if(x=='}' && st.top()=='{') st.pop();
            else if(x==']' && st.top()=='[') st.pop();
            else st.push(x);
        }
        else st.push(x);
    }
}
```

```
return st.empty();  
}
```

Time Complexity: $O(n)$
Space Complexity: $O(n)$

Simplify Path

CODE:

```
void buildAns(stack<string>&s, string&ans) {  
    if(s.empty()) {  
        return;  
    }  
    string minPath = s.top(); s.pop();  
    buildAns(s, ans);  
    ans += minPath;  
}  
string simplifyPath(string path) {  
    stack<string>s;  
    int i = 0;  
    while(i < path.size()) {  
        int start = i;  
        int end = i+1;  
        while(end < path.size() && path[end] != '/') {  
            ++end;  
        }  
        string minPath = path.substr(start, end-start);  
        i = end;  
        if(minPath == "/" || minPath == "/.") {  
            continue;  
        }  
        if(minPath != "/.") {  
            s.push(minPath);  
        }  
        else if(!s.empty()) {  
            s.pop();  
        }  
    }  
    string ans = s.empty() ? "/" : "";
```

```
    buildAns(s, ans);  
    return ans;  
}
```

Time Complexity: $O(n)$
Space Complexity: $O(n)$

Min Stack

CODE:

```
    stack<pair<int,int>> st;  
    MinStack() {  
    }  
    void push(int val) {  
        if(st.empty()) st.push({val,val});  
        else{  
            int curr_min=getMin();  
            if(val<curr_min){  
                st.push({val,val});  
            }  
            else st.push({val,curr_min});  
        }  
    }  
    void pop() {  
        st.pop();  
    }  
    int top() {  
        if(!st.empty()) return st.top().first;  
        return -1;  
    }  
    int getMin() {  
        if(st.empty()) return -1;  
        else return st.top().second;  
    }  
}
```

Time Complexity: $O(n)$
Space Complexity: $O(n)$

Search Insert Position

CODE:

```
int searchInsert(vector<int>& nums, int target) {
    int n = nums.size();
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (nums[mid] == target) {
            return mid;
        } else if (nums[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return low;
}
```

Time Complexity: $O(\log n)$
Space Complexity: $O(1)$

Search 2D Matrix

CODE:

```
bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int row = matrix.size();
    int col = matrix[0].size();
    int left = 0, right = row*col - 1, mid = -1, value;
    while (left <= right) {
        mid = left + (right-left)/2;
        value = matrix[mid/col][mid%col];
        cout << "Value: " << value << endl;
        if (value == target) {
            return true;
        }
        else if (target < value) {
            right = mid-1;
        }
        else {

```



```

left = mid + 1;
}
}
return false;
}

```

Time Complexity: $O((\log n)^2)$
Space Complexity: $O(1)$

Search in rotated sorted array

CODE:

```

int search(vector<int>& arr, int target) {
int n=arr.size();
int low=0,high=n-1;
while(low<=high){
int mid=(low+high)/2;
if(arr[mid]==target) return mid;
else if(arr[low]<=arr[mid]){
if(arr[low]<=target && target<=arr[mid]){
high=mid-1;
}
else{
low=mid+1;
}
}
else{
if(arr[mid]<=target && target<=arr[high]){
low=mid+1;
}
else{
high=mid-1;
}
}
}
return -1;
}

```

Time Complexity: $O(\log n)$
Space Complexity: $O(1)$

Find First and Last Position of Element in Sorted Array:

CODE:

```
int firstocc(vector<int> nums,int n,int target){
int low=0,high=n-1,first=-1;
while(low<=high){
int mid=(low+high)/2;
if(nums[mid]==target){
high=mid-1;
first=mid;
}
else if(nums[mid]<target){
low=mid+1;
}
else{
high=mid-1;
}
}
return first;
}
```

```
int lastocc(vector<int> nums,int n,int target){
int low=0,high=n-1,last=-1;
while(low<=high){
int mid=(low+high)/2;
if(nums[mid]==target){
low=mid+1;
last=mid;
}
else if(nums[mid]<target){
low=mid+1;
}
else{
high=mid-1;
}
}
}
```

```
return last;
}
```

```
vector<int> searchRange(vector<int>& nums, int target) {
    int n=nums.size();
    int first=firstocc(nums,n,target);
    if(first==-1) return vector<int> {-1,-1};
    int last=lastocc(nums,n,target);
    return vector<int> {first,last};
}
```

Time Complexity: $O(\log n)$
Space Complexity: $O(1)$

Minimum in Rotated Sorted Array

CODE:

```
int findMin(vector<int>& nums) {
    int n=nums.size();
    int low=0,high=n-1,ans=INT_MAX;
    while(low<=high){
        int mid=(low+high)/2;
        if(nums[low]<=nums[mid]){
            ans=min(ans,nums[low]);
            low=mid+1;
        }
        else{
            ans=min(ans,nums[mid]);
            high=mid-1;
        }
    }
    return ans;
}
```

Time Complexity: $O(\log n)$
Space Complexity: $O(1)$

Median of Two Sorted Arrays

CODE:

```
double findMedianSortedArrays(vector<int>& nums1, vector<int>&
nums2) {
    nums1.insert(nums1.end(),nums2.begin(),nums2.end());
    sort(nums1.begin(),nums1.end());
    if(nums1.size()%2==1){
        int a=nums1.size()/2;
        return static_cast<double>(nums1[a]);
    }
    else{
        int mid1=nums1.size()/2;
        int mid2=mid1-1;
        double add=nums1[mid1]+nums1[mid2];
        double med=add/2;
        return (med);
    }
}
```

Time Complexity: $O(\log n)$

Space Complexity: $O(1)$