**Leetcode Study Notes**

**Week 1**

Array

Python, JavaScript, Ruby, PHP: array or list size is dynamic.

**Advantages:**

1. Store multiple elements of the same type with only one variable name.
2. Accessing elements is fast as long as you have the index.

**Disadvantages:**

1. Addition and removal of elements is slow, the remaining elements need to be shifted (an exception, insert or remove at the end).
2. For some languages where size is fixed, it cannot alter the size after initialization. A new array has to be allocated and existing elements have to be copied over, which takes O(n) time.

**Array in Data Structure**

**\*Array in Python (Same datatype)**

import array

模板：arrayName = array.array(type code for data type, [array,items])

arrayName[indexNum]

arrayName.insert(index, value)

arrayName.remove(value)

arrayName.index(value)

arrayName.udpate(index, value)

**Array Operations in C++**

main() {

int pos = 2;

int size = 4;

int balance[] = {300,200,100,50,0};

for(int i = 0; i<5; i++) {

printf("%d\n",balance[i]);

}

/\* FOR SHIFTING ITEMS TO A GREATER INDEX \*/

for(int i = size; i >= pos; i--) {

balance[i+1]=balance[i];

}

/\* FOR INSERTING VALUE AT OUR DESIRED INDEX \*/

balance[pos] = 150;

}

}

**Array Operations in Java**

import java.util.Scanner;

public class AddElements {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

System.out.println("Enter the size of the array");

int n=sc.nextInt();

int arr[]=new int[n];

System.out.println("Enter Elements in the array");

for(int i=0;i<n;i++)

{

arr[i]=sc.nextInt();

}

for(int j=0;j<n;j++)

{

System.out.print(arr[j]+" ");

}

}

}

**Common Terms**

Subarray: a range of continuous values within an array [2, 3, 6, 1, 5, 4], [3, 6, 1]

Subsequence: a sequence that can be derived from the given sequence by deleting some or no elements without changing the order of the remaining elements. [2, 3, 6, 1, 5, 4], [3, 1, 5]

**Time Complexity**

|  |  |  |
| --- | --- | --- |
| Operation | Big-O | Note |
| Access | O(1) |  |
| Search | O(n) |  |
| Search (sorted array) | O(log(n)) |  |
| Insert | O(n) | Shifting all the subsequent elements |
| Insert (at the end) | O(1) | No shifting |
| Remove | O(n) | Shifting all the subsequent elements |
| Remove (at the end) | O(1) | No shifting |

**Things to Look Out for during Interviews**

Clarify if there are duplicate values in the array. Would the presence of duplicate values affect the answer? Does it make the question simpler or harder?

When using an index to iterate through array elements, be careful not to go out of bounds.

Be mindful about slicing or concatenating arrays in your code. Typically, slicing and concatenating arrays would take O(n) time. Use start and end indices to demarcate a subarray/range where possible.

**Corner cases**

Empty sequence

Sequence with 1 or 2 elements

Sequence with repeated elements

Duplicated values in the sequence

**Sliding Window**

Two pointers usually move in the same direction will never overtake each other. This ensures that each value is only visited at most twice and the time complexity is still O(n)

class Solution {

public:

int minSubArrayLen(int target, vector<int>& nums) {

int i=0;int j=0;int sum=0;int mn=INT\_MAX;

while(j<nums.size()){

sum+=nums[j];

while(sum>=target){

sum-=nums[i];

mn=min(j-i+1,mn);

i++;

}

j++;

}

if(mn==INT\_MAX){

return 0;

}

return mn;

}

};

Time complexity: $$O(n)$$

Space complexity: O(1)

**Two pointers**

Two pointers is a more general version of sliding window where the pointers can cross each other and can be on different arrays. When you are given two arrays to process, it is common to have **one index per array (pointer) to traverse/compare the both of them**, incrementing one of the pointers when relevant.

**Traversing from the right**

Sometimes you can traverse the array starting from the right instead of the conventional approach of from the left.

MONOTONIC STACK

A black board with blue writing

Description automatically generated

MONOTONIC STACK （Traversing from the right to left, stack from bottom to top）

A group of cartoon characters

Description automatically generated

Greedy Algorithm

A computer screen with text and green lines

Description automatically generated

**Sorting the array**

Is the array sorted or partially sorted? If it is, some form of binary search should be possible. This also usually means that the interviewer is looking for a solution that is faster than O(n).

**Precomputation**

For questions where summation or multiplication of a subarray is involved, pre-computation using hashing or a prefix/suffix sum/product might be useful.

A blackboard with numbers and arrows

Description automatically generated

**Index as a hash key**

If you are given a sequence and the interviewer asks for O(1) space, it might be possible to use the array itself as a hash table. For example, if the array only has values from 1 to N, where N is the length of the array, negate the value at that index (minus one) to indicate presence of that number.

**Traversing the array more than once**

This might be obvious, but traversing the array twice/thrice (as long as fewer than n times) is still O(n). Sometimes traversing the array more than once can help you solve the problem while keeping the time complexity to O(n).

**Week 2**