LeetCamp Week1

Zixuan Jia/Yile Chen

Housekeeping

- Every Sunday 2pm
- Check discord regularly for updates
- Project Github: https://github.com/infknight/Leetcamp_Fall_2020

Tentative Schedule

- Array/Vector
- String
- Linked List
- Recursion (Special Speaker for this topic)
- Tree
- Stack/Queue
- HashMap
- BFS/DFS
- Graph
- Dynamic Programming (CSCE 411)

Overview

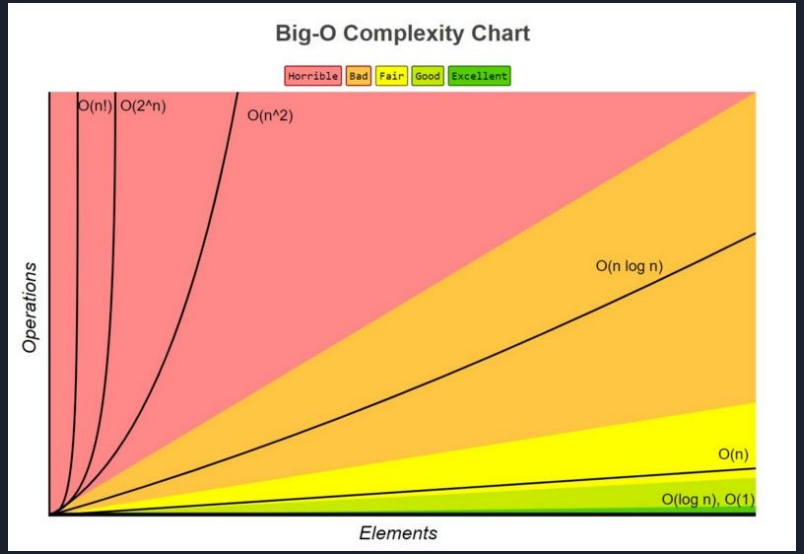
- What is LeetCode?
- Topics for Fall/Spring
- Asymptotic Notation Big (O)
- Space Complexity
- Sort()
- HashMap/Dictionary

LeetCode

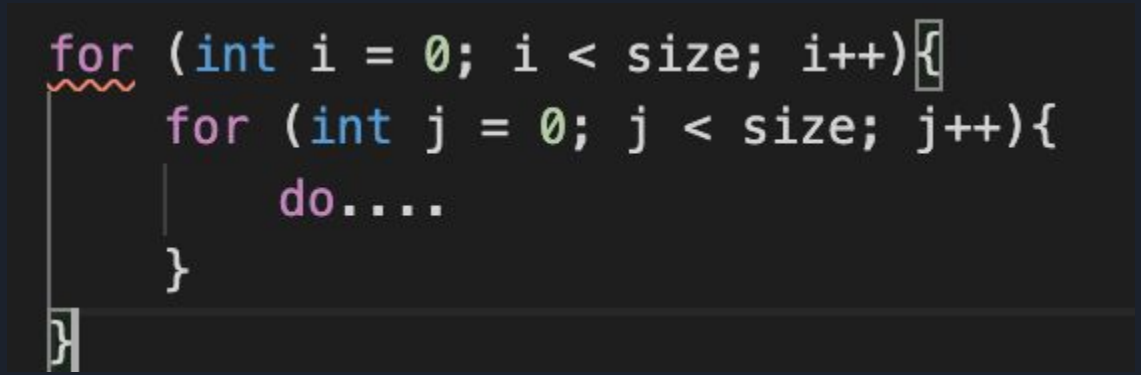
	#	Title	Solution	Acceptance	Difficulty	Frequency
✔	85	Maximal Rectangle		33.0%	Hard	
✔	621	Task Scheduler		45.2%	Medium	
✔	412	Fizz Buzz		59.3%	Easy	
✔	41	First Missing Positive		28.6%	Hard	
✔	54	Spiral Matrix		30.2%	Medium	
	642	Design Search Autocomplete System		37.1%	Hard	
✔	344	Reverse String		63.1%	Easy	
✔	127	Word Ladder		23.7%	Medium	
	124	Binary Tree Maximum Path Sum		29.8%	Hard	
✔	336	Palindrome Pairs		30.7%	Hard	
✔	141	Linked List Cycle		36.5%	Easy	
	193	Valid Phone Numbers		25.1%	Easy	

Asymptotic Notation Big (O)

- Big-O notation represents the upper bound of the running time of an algorithm. Thus, it gives the worst case complexity of an algorithm.
- Common time complexities include O(log(n)), O(n), O(n*log(n)), O(n^2).
- A standard linear search algorithm runs in O(n) because its running time is expected to increase linearly with its input size. Similarly, we know that binary searches are O(log(n))
- Example: f(n)=$2n^2$+2n+1. Drop its non-dominant terms (like +2n and +1) and its constants (like the 2 in $2n^2$) to obtain its asymptotic notation O(n^2).



What is the Big O for the following code?



ANS: O(N^2)

What is the Big O for the following code?

```
while (first + 1 < last){
    mid = first + (last - first) / 2
    if(mid > target)
        first = mid
    if (mid < target)
        last = mid
}
```

ANS: O(logn)

What is the Big O for the following code?

```
int k = 5
while (first < last){
    while(first > 0 && first < k)
        first++
    last--
}
```

ANS: O(N)

Space Complexity

- Measure of the amount of working storage an algorithm needs.
- Not as important as Asymptotic Notation
- Extra Space Complexity is not good, but sometimes is needed

Sort()

- During a technical interview, if the question is NOT DIRECTLY sorting questions. You can use the build in sort to sort your input
- Python sort() uses Timesort method
- C++ sort() uses Quicksort
- Assume Time Complexity for sort O(nlogn)
- It is good to know Quicksort in the future; related topics: quick select

HashMap

- A data structure that contains a key value pair
- In python hash is also a dictionary
  - Hash = { }
- In C++, it is different
  - unordered_map <int, int> Hash;
- Why using HashMap during coding interview?
  - Search: O(1)
  - Insert: O(1)
  - Space: O(n)
  - Delete: O(1)

CONTAIN DUPLICATE

217. Contains Duplicate

Easy 1113 777 Add to List Share

Given an array of integers, find if the array contains any duplicates.

Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

Example 1:

```
Input: [1,2,3,1]
Output: true
```

Example 2:

```
Input: [1,2,3,4]
Output: false
```

Example 3:

```
Input: [1,1,1,3,3,4,3,2,4,2]
Output: true
```

CONTAIN DUPLICATE (Approach 1): Native Linear Search

- Given an array of n numbers, check all possible pairs for duplicates.
- There are $C(n,2) = n(n+1)/2$ total number of pairs. Therefore time complexity is O(n^2)
- Use two nested for loops to iterate through the array, and compare each pair. If two numbers are equal, return True.

CONTAIN DUPLICATE (Approach 2) : Sorting (Optional Method)

- Use built in .sort() function to sort the array, which takes O(nlogn) worst time. Then sweep through the sorted array to find duplicates in one iteration, which takes O(n) complexity.

CONTAIN DUPLICATE (Approach 3): HashMap

- Utilize a hash map, using the search() and insert() function, we can reduce complexity to O(n) overall

TWO SUM

1. Two Sum

Easy 17180 615 Add to List Share

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to* `target`.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Output: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Example 2:

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

Example 3:

```
Input: nums = [3,3], target = 6
Output: [0,1]
```

TWO SUM (Approach 1) : Brute Force

- Given an array, and a target. Find the 2 integers sum equal to the target. Return the index of the 2 integer.
- Nested Loop to go through the array, find the target, return the index of the 2
- What is the Complexity?
- Can we come up with a better solution?

TWO SUM (Approach 2) : Sort the array (Optional Method)

- It is very difficult to implement this idea since we are returning the index of 2 target
- However it is a very important idea for 3 sum
- Using the idea 2 pointers
  - Not the address, but uses the left and right pointers to accessing the index
- Original array
  - 15, 11, 2, 7 target: 9
- After sort
  - 2, 7, 11, 15 target: 9
- Start pointer: int start = 0. End pointer: int end = array.size() - 1
- If the array[start] + array[end] > target
  - end-- ;
- If the array[start] + array[end] < target
  - start++;

TWO SUM (Approach 3): HashMap

- HashMap allows us to use O(1) to search
- How can we utilize HashMap to solve this problem?
- Time Complexity?

// 1 0 -1 2 -6 7 target = 0 O(N^2)

Sort

-6 , -1 , 0, 1 ,2 ,7