

Review

- Recursion
- Merge sort
 - Time complexity
 - Memory complexity

Algorithm Design

Lecture 12

Hyung-Sin Kim



SNU Graduate School of Data Science

Top-down Design Thinking

- Recall function design guideline (Module 2-1-4)
- Understand the given problem
- Think about various ways of solving the problem
- Describe your solution (recipe) logically in human language
- Divide the solution into logical steps
- Translate each logical step into computer language

Technical Interview Process

- Interviewer
 - Describes the problem and shows a few examples
 - Answer interviewee's questions kindly
 - Give feedback to interviewee's solution (optimal or not?) so that they can end up the best solution
- Interviewee
 - Make sure they understand the problem correctly
 - Think what data structure and algorithm to use to solve the problem
 - Solve the problem and **think out loud** so that interviewer can see their thought process
 - Discuss memory overhead and time complexity of the solution
 - Verify if the solution works well by using some test cases

What Companies Want

- Engineers who know what they do and describe it fluently
- “I developed a program and it works. I’m happy! The developing procedure was such a nightmare, I don’t want to go back and read it!” ☹
 - “Well...You can be happy at your home safely. Companies are dangerous for you.”
- “I analyzed the problem, planned what data structures and algorithms to use, analyzed time complexity and memory overhead, verified if it works through some representative test cases. Moreover, I can describe why I chose these data structures and algorithms.” 😊

Where We Are

- We just learnt basic data structures and algorithms
- Of course, these are not the end, there are separate courses, data structures and algorithms, offered by DS, CS, and ECE
 - Knowing more of them enables you to solve more problems more efficiently

Practice!

- <https://leetcode.com/>
 - Famous problems and solutions
 - It evaluates your algorithm, how fast and memory efficient the algorithm is

A screenshot of the LeetCode website interface. The top navigation bar includes 'LeetCode', 'Explore', 'Problems', 'Mock', 'Contest', 'Discuss', and 'Store'. A 'Day 11' badge is next to 'Problems'. On the right, there's a 'October LeetCode Challenge' banner, a 'Premium' badge, and user notification icons. Below the navigation bar, tabs for 'Description', 'Solution', 'Discuss (999+)', and 'Submissions' are visible. The 'Solution' tab is active, showing a 'Success' message and details for a submission. The runtime is 44 ms, faster than 93.40% of Python3 submissions. Memory usage is 15.3 MB, less than 15.73% of Python3 submissions. Below this, there are buttons for 'Two Sum III - Data structure design' and 'Two Sum Less Than K'. At the bottom, there's a 'Show off your acceptance' section with social media icons for Facebook, Twitter, and LinkedIn. On the right side of the screenshot, a code editor is open, showing a Python3 solution for the 'Two Sum' problem. The code defines a 'Solution' class with a 'twoSum' method that uses a hash map to find the complement of each number in the array.

LeetCode

Success Details >

Runtime: 44 ms, faster than 93.40% of Python3 online submissions for Two Sum.

Memory Usage: 15.3 MB, less than 15.73% of Python3 online submissions for Two Sum.

Next challenges:

Two Sum III - Data structure design Two Sum Less Than K

Show off your acceptance: f t in

```
1 class Solution:
2     def twoSum(self, nums: List[int], target: int) -> List[int]:
3         map = {}
4         for i in range(len(nums)):
5             if nums[i] in map:
6                 return [map[nums[i]], i]
7             complement = target - nums[i]
8             map[complement] = i
```

Example – Two Sum

- <https://leetcode.com/problems/two-sum/>

Given an array of integers, return **indices** of the two numbers such that they add up to a specific target.

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

Example:

```
Given nums = [2, 7, 11, 15], target = 9,
```

```
Because nums[0] + nums[1] = 2 + 7 = 9,  
return [0, 1].
```


Solutions – Two for loops (1)

- Algorithm design in human language
 - For every number pair in the list,
 - Check if their sum is the given target value
 - If the sum is the target value, return the numbers' indices

Solutions – Two for loops (1)

- Programming in computer language
 - `def twoSum(nums: list, target: int) -> list:`
 - `for i in range(len(nums)):`
 - `for j in range(len(nums)):`
 - `if nums[i] + nums[j] == target:`
 - `return [i, j]`
- Performance
 - Time complexity $\sim N^2$
 - Space complexity ~ 1

Solutions – Two for loops (2)

- `def twoSum(nums: list, target: int) -> list:`
 - `for i in range(len(nums)):`
 - `for j in range(i+1, len(nums)):`
 - `if nums[i] + nums[j] == target:`
 - `return [i, j]`
- Time complexity $\sim N^2/2$ (slightly better than before)
- Space complexity ~ 1

Solutions – Dictionary (1)

- `def twoSum(nums: list, target: int) -> list:`
 - **`myDict = {}`**
 - `for i in range(len(nums)):` *# Dictionary to find each value's index*
 - `myDict[nums[i]] = i`
 - `for i in range(len(nums)):`
 - `complement = target - nums[i]`
 - `if complement in myDict:` *# Search my complement from the dictionary*
 - `return [i, myDict[complement]]`
- Time complexity $\sim 2N$
- Space complexity $\sim N$

Solutions – Dictionary (2)

- `def twoSum(self, nums: list, target: int) -> list:`
 - `myDict = {}`
 - `for i in range(len(nums)):`
 - `if nums[i] in myDict:` *# Did someone select me as its complement?*
 - `return [myDict[nums[i]], i]`
 - `complement = target - nums[i]`
 - `myDict[complement] = i` *# Add an entry so that my complement can find my index*
- Time complexity $\sim N$ (slightly better than before)
- Space complexity $\sim N$

What if the list is sorted?

Solutions – Two Pointers

- `def twoSumSorted(self, nums: list, target: int) -> list:`
 - `i, j = 0, len(nums)-1`
 - `while nums[i] + nums[j] != target:`
 - `if nums[i] + nums[j] < target:`
 - `i = i+1`
 - `else:`
 - `j = j-1`
 - `return [i, j]`
- Time complexity $\sim N$
- Space complexity ~ 1

Thanks!