SHIFTKEY ACADEMY

Intro to Algorithmic Problem Solving

Week 2

SHIFTKEY ACADEMY

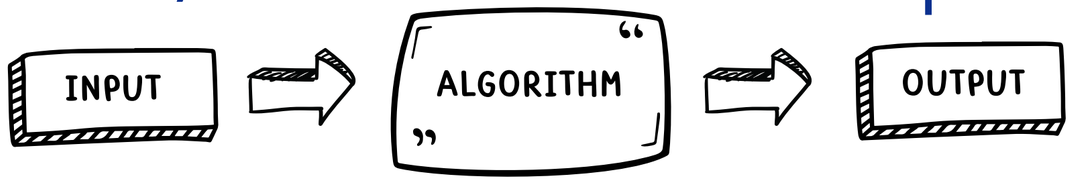
Housekeeping

- **Contact me:** yuhan@dal.ca
- **Language for assignments:** Any language you prefer
- **DDL extension for A1:** Sunday, Jun 15
- **Exam language & make-up exam:** Your choice (lmk asap)

SHIFTKEY ACADEMY

How AI Do Reasoning?

- **Semantic Parsing:** Convert natural language input into structured concepts, intent, and implicit assumptions.
- **Logical Modeling:** Build an internal reasoning model (e.g. logic tree, flow graph, or function-like structure).
- **Inference Execution:** Run the model to derive conclusions, using rules, conditions, and consistency checks.
- **Natural Language Generation:** Translate the reasoning trace into coherent, human-readable output.



SHIFTKEY ACADEMY

Then vs Now

Then (Pre-AI Era)	Now (AI Era)
Learn to solve problems by hand	Learn to design problem solvers and guide AI thinking
Focused on coding implementations	Focused on algorithmic thinking and strategy
Tested through whiteboard coding	Applied through prompt engineering and tool building
Algorithms as end-goal of computer science learning	Algorithms as meta-tools to control intelligent systems
Human solves → machine executes	Human defines → machine solves

SHIFTKEY ACADEMY

Course Overview

Week 1

Algorithm Intro
Time/Space Complexity
Data Structures
Two pointers
Prefix Sum

Week 2

Recursion
Divide and Conquer
Sorting
Binary Search

Week 3

Math Problems
Bit Manipulation
Greedy
Dynamic Programing

Week 4

Tree & Graph
BFS/DFS
Backtracking
Union-Find

SHIFTKEY ACADEMY

Recursion

SHIFTKEY ACADEMY

Recursion

A method where a function calls itself to solve a problem by breaking it into smaller subproblems.

Russian nesting dolls

SHIFTKEY ACADEMY

Factorial

$$n! = \begin{cases} 1, & \text{if } n = 0 \\ n \times (n-1)!, & \text{if } n > 0 \end{cases}$$

```python
def factorial(n):
    if n == 0:
        return 1  # base case
    else:
        return n * factorial(n - 1)  # recursive call
```

Time complexity: O(n)
Space complexity: O(n)
(using stack with depth n)

factorial(3)
= 3 * factorial(2)
= 3 * 2 * factorial(1)
= 3 * 2 * 1 * factorial(0)
= 3 * 2 * 1 * 1
= 6

SHIFTKEY ACADEMY

Time Complexity

$$T(n) = T(n-1) + \mathcal{O}(1)$$

Using recursion:

$$\begin{aligned} T(n) &= T(n-1) + 1 \\ &= T(n-2) + 1 + 1 \\ &= T(n-3) + 1 + 1 + 1 \\ &\dots \\ &= T(0) + n \\ &= 1 + n = \mathcal{O}(n) \end{aligned}$$

SHIFTKEY ACADEMY

Fibonacci (Naive Recursion)

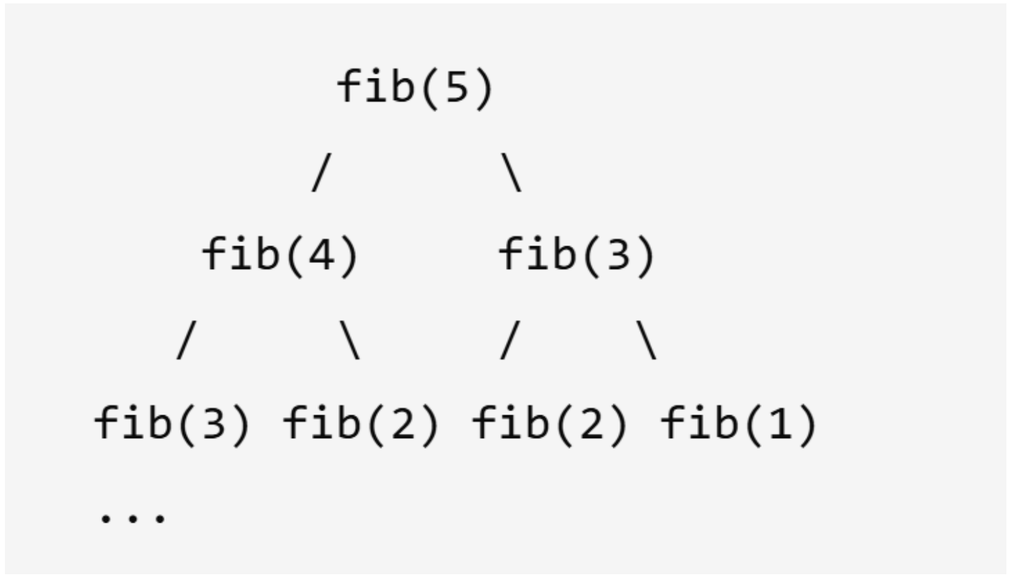
$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{if } n \geq 2 \end{cases}$$

```python
def fib(n):
    if n == 0: return 0
    if n == 1: return 1
    return fib(n - 1) + fib(n - 2)
```

Time complexity: $O(2^n)$ <--Disaster!!!

Space complexity: O(n)

(using stack with depth n)



SHIFTKEY ACADEMY

Fibonacci (Memoization)

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{if } n \geq 2 \end{cases}$$

```python
def fib(n, memo={}):
    if n in memo:
        return memo[n]
    if n == 0:
        return 0
    if n == 1:
        return 1
    memo[n] = fib(n - 1, memo) + fib(n - 2, memo)
    return memo[n]
```

Time complexity: O(n)
Space complexity: O(n)
(at most n elements in memory)

This is a Top-Down recursion.

SHIFTKEY ACADEMY

Top-Down vs. Bottom-Up

- **Top-Down:**
  - Start from the original problem and recursively break it down into subproblems. Often combined with memoization to avoid redundant work.
- **Bottom-Up:**
  - Start from the base cases and iteratively build up to the final answer. This is typically implemented using loops and often called Dynamic Programming (DP). → Week 3

SHIFTKEY ACADEMY

Sorting

SHIFTKEY ACADEMY

Bubble Sort

Repeatedly swaps adjacent elements if they are in the wrong order.

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(n - 1 - i):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
```

Time complexity:
Worst/Avg: $O(n^2)$,
Best: O(n) (already sorted, **with optimization**)

Space complexity: O(1)

SHIFTKEY ACADEMY

Bubble Sort (with optimizaton)

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False  # flag to check if any swap happened
        for j in range(n - 1 - i):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:
            break  # exit early if already sorted
```

SHIFTKEY ACADEMY

Selection Sort

Repeatedly selects the smallest element and places it at the beginning.

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
```

Time complexity:
Worst/Avg/Best: $O(n^2)$

Space complexity: $O(1)$

SHIFTKEY ACADEMY

Insertion Sort

Builds the sorted array one element at a time by inserting into the correct position.

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```

Time complexity:
Worst/Avg: $O(n^2)$,
Best: $O(n)$

Space complexity: $O(1)$

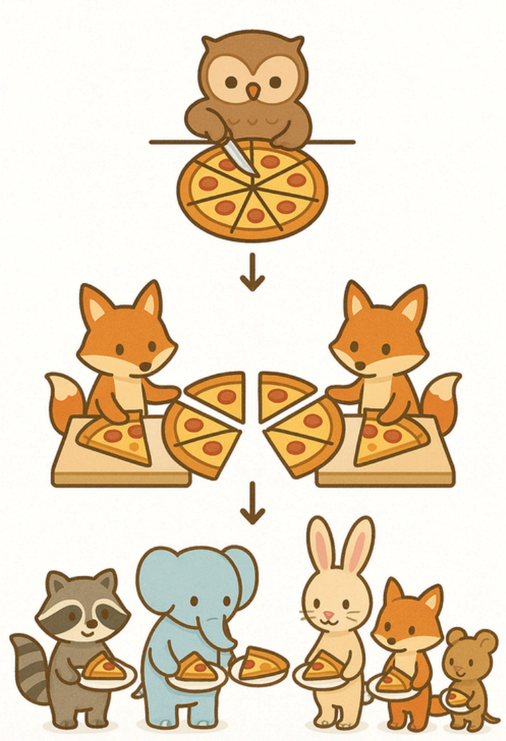
SHIFTKEY ACADEMY

Divide and Conquer

SHIFTKEY ACADEMY

Divide and Conquer

- Divide-and-conquer is a common use case of recursion, but not all recursive algorithms divide problems (e.g., factorial is recursive but not divide-and-conquer).
- **Structure:**
  - **Divide**: Split the problem into smaller parts
  - **Conquer**: Solve each part recursively
  - **Combine**: Merge the solutions of subproblems
- **Common use cases**: Merge Sort, Quick Sort
- **Time Complexity**: Depends on the problem, often analyzed using recurrence relations



SHIFTKEY ACADEMY

QUICK INSIGHTS

First Principles

"First principles is a physics way of looking at the world. You boil things down to the most fundamental truths and then reason up from there.

(For example) people may say 'battery packs are really expensive and that is the way they will always be'. No, that's pretty dumb. If you apply that reasoning to anything new, you wouldn't ever be able to get to that new thing."

– Elon Musk

SHIFTKEY ACADEMY

Merge Sort

Divides the array into halves, recursively sorts, and merges them.

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)
```

```
def merge(left, right):
    res = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            res.append(left[i])
            i += 1
        else:
            res.append(right[j])
            j += 1
    return res + left[i:] + right[j:]
```

Time complexity:
Worst/Avg/Best: O(n log n)

Space complexity: O(n)

SHIFTKEY ACADEMY

Time Complexity of Merge Sort

$$T(n) = 2T(n/2) + \mathcal{O}(n)$$

Two sub-problem of size n/2

Merge

$$T(n) = aT(n/b) + f(n)$$

SHIFTKEY ACADEMY

Mater Theorem

Theorem 4.1 (Master theorem)

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

SHIFTKEY ACADEMY

Quick Sort

Picks a pivot and partitions the array into smaller and larger parts, then recursively sorts.

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[0]
    less = [x for x in arr[1:] if x <= pivot]
    greater = [x for x in arr[1:] if x > pivot]
    return quick_sort(less) + [pivot] + quick_sort(greater)
```

Time complexity:
Avg/Best: O(n log n), Worst: $O(n^2)$

Space complexity: O(log n) (due to recursion)

SHIFTKEY ACADEMY

Built-in Python Sort (Timsort)

Hybrid of merge sort and insertion sort, optimized for real-world data.

```
nums = [5, 2, 9, 1]
nums.sort()
print(nums)  # Output: [1, 2, 5, 9]
```

```
nums = [5, 2, 9, 1]
sorted_nums = sorted(nums)
print(sorted_nums)  # Output: [1, 2, 5, 9]
print(nums)         # Output: [5, 2, 9, 1] (original unchanged)
```

For string, use sorted() only.

Time complexity:
Best: O(n), Avg/Worst: O(n log n)

Space complexity: O(log n) (due to recursion)

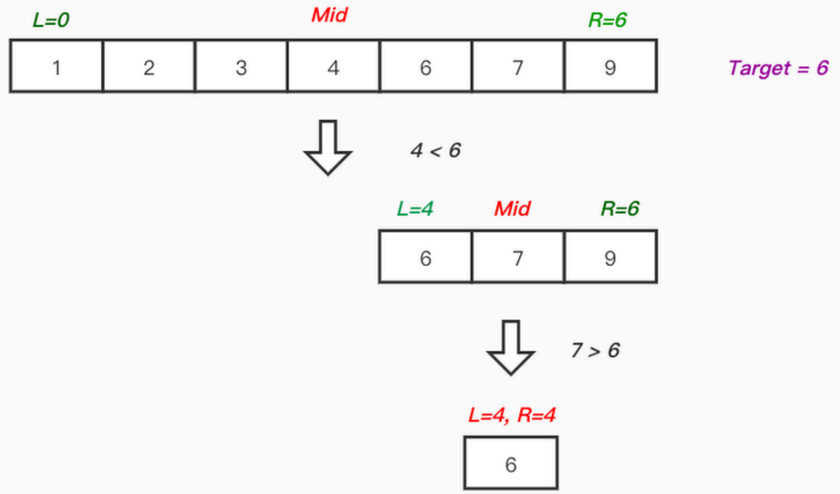
SHIFTKEY ACADEMY

Binary Search

SHIFTKEY ACADEMY

Binary Search

- **Idea**
  - Find an element in a sorted list by repeatedly dividing the search interval in half.
- **Key Requirement**
  - Input must be sorted
- **Process**
  - Check the middle element
  - If it's too small, search the right half
  - If it's too big, search the left half



SHIFTKEY ACADEMY

Binary Search

```python
def binary_search(arr, target):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = left + (right - left) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

Time complexity: O(logn)
Space complexity: O(1)

Q: Why we use
mid = left + (right - left) // 2
instead of
mid = (left + right) // 2 ?

A: Using mid = left + (right - left) // 2 prevents potential integer overflow that can occur with (left + right) // 2, especially in languages with fixed-size integers like Java or C++.
Even though Python is safe, this is considered a best practice in algorithm writing.

SHIFTKEY ACADEMY

Practice Questions

SHIFTKEY ACADEMY

167. Two Sum II - Input Array Is Sorted

Medium | Topics | Companies

Given a **1-indexed** array of integers `numbers` that is already ***sorted in non-decreasing order***, find two numbers such that they add up to a specific `target` number. Let these two numbers be `numbers[index1]` and `numbers[index2]` where `1 <= index1 < index2 <= numbers.length`.

Return *the indices of the two numbers,* `index1` *and* `index2`*,* ***added by one*** *as an integer array* `[index1, index2]` *of length 2.*

The tests are generated such that there is **exactly one solution**. You **may not** use the same element twice.

Your solution must use only constant extra space.

Example 1:

> **Input:** numbers = [2,7,11,15], target = 9
> **Output:** [1,2]
> **Explanation:** The sum of 2 and 7 is 9. Therefore, $index_1 = 1$, $index_2 = 2$. We return [1, 2].

Example 2:

> **Input:** numbers = [2,3,4], target = 6
> **Output:** [1,3]
> **Explanation:** The sum of 2 and 4 is 6. Therefore $index_1 = 1$, $index_2 = 3$. We return [1, 3].

Example 3:

> **Input:** numbers = [-1,0], target = -1
> **Output:** [1,2]
> **Explanation:** The sum of -1 and 0 is -1. Therefore $index_1 = 1$, $index_2 = 2$. We return [1, 2].

SHIFTKEY ACADEMY

Binary Search

Use binary search to find the complement of each element, achieving better efficiency than brute force by using the sorted order of the array.

```python
# binary search
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        for i in range(len(numbers)):
            l, r = i+1, len(numbers)-1
            tmp = target - numbers[i]
            while l <= r:
                mid = l + (r-l)//2
                if numbers[mid] == tmp:
                    return [i+1, mid+1]
                elif numbers[mid] < tmp:
                    l = mid+1
                else:
                    r = mid-1
```

Time complexity: O(nlogn) Space complexity: O(1)

SHIFTKEY ACADEMY

21. Merge Two Sorted Lists

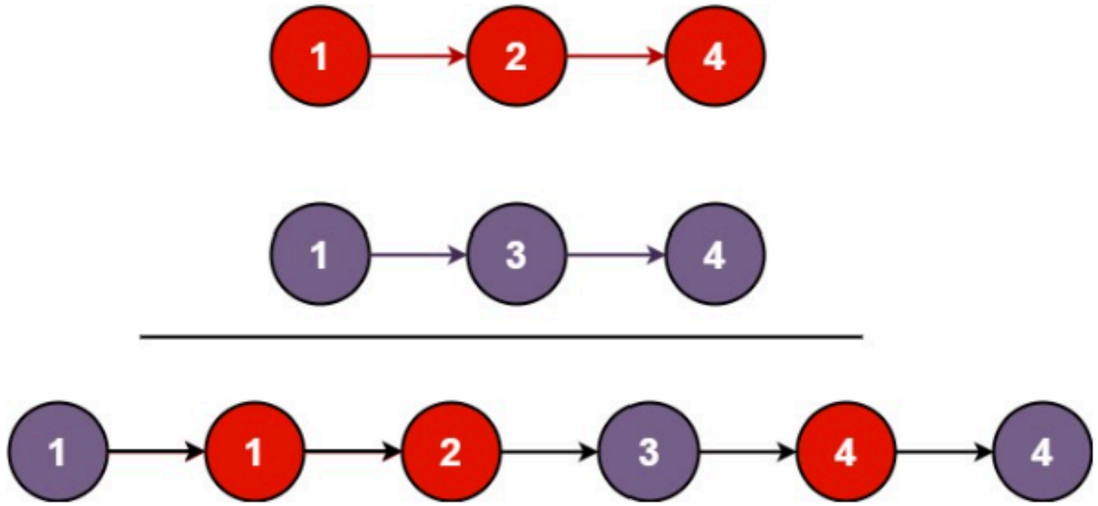
Easy | Topics | Companies

You are given the heads of two sorted linked lists `list1` and `list2`.

Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list.*

Example 1:




```
Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]
```

Example 2:

```
Input: list1 = [], list2 = []
Output: []
```

Example 3:

```
Input: list1 = [], list2 = [0]
Output: [0]
```

SHIFTKEY ACADEMY

Iterative

Iteratively merges two sorted lists by comparing heads and building a new list using a dummy node.

```python
class Solution:
    def mergeTwoLists(self, list1: Optional[ListNode], list2: Optional[ListNode]) -> Optional[ListNode]:
        curr = dummy = ListNode()
        while list1 and list2:
            if list1.val <= list2.val:
                curr.next = list1
                list1, curr = list1.next, list1
            else:
                curr.next = list2
                list2, curr = list2.next, list2
        if list1 or list2:
            curr.next = list1 if list1 else list2
        return dummy.next
```

Time complexity: O(m+n) Space complexity: O(m+n)

*m and n are the lengths of the two linked lists

SHIFTKEY ACADEMY

Recursive

Recursively merges two sorted lists by always taking the smaller head and advancing its pointer.

```python
class Solution:
    def mergeTwoLists(self, list1: Optional[ListNode], list2: Optional[ListNode]) -> Optional[ListNode]:
        if not list1:
            return list2
        if not list2:
            return list1
        if list1.val <= list2.val:
            list1.next = self.mergeTwoLists(list1.next, list2)
            return list1
        else:
            list2.next = self.mergeTwoLists(list1, list2.next)
            return list2
```

Time complexity: O(m+n) Space complexity: O(1)

SHIFTKEY ACADEMY

169. Majority Element

Solved

Easy | Topics | Companies

Given an array `nums` of size `n`, return *the majority element*.

The majority element is the element that appears more than `⌊n / 2⌋` times. You may assume that the majority element always exists in the array.

Example 1:

```
Input: nums = [3,2,3]
Output: 3
```

Example 2:

```
Input: nums = [2,2,1,1,1,2,2]
Output: 2
```

SHIFTKEY ACADEMY

Sorting

Sort the array and return the middle element, which must be the majority due to its frequency.

```python
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        nums.sort()
        return nums[len(nums) // 2]
```

Time complexity: O(n log n) Space complexity: O(1)

SHIFTKEY ACADEMY

Hash Map (Frequency Counting)

Use a hash map to count occurrences of each element and return the one that appears more than ⌊ n / 2 ⌋ times.

```python
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        count = {}
        for num in nums:
            count[num] = count.get(num, 0) + 1
            if count[num] > len(nums) // 2:
                return num
```

Time complexity: O(n) Space complexity: O(n)

SHIFTKEY ACADEMY

Moore’s Voting Algorithm

Maintain a candidate and a counter, incrementing or decrementing based on matches to efficiently find the majority in one pass.

```python
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        candidate = 0
        count = 0
        for num in nums:
            if count == 0:
                candidate = num
            if candidate == num:
                count +=1
            else:
                count -=1
        return candidate
```

Time complexity: O(n) Space complexity: O(1)

SHIFTKEY ACADEMY

278. First Bad Version

Solved

Easy | Topics | Companies

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have `n` versions `[1, 2, ..., n]` and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API `bool isBadVersion(version)` which returns whether `version` is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Example 1:

```
Input: n = 5, bad = 4
Output: 4
Explanation:
call isBadVersion(3) -> false
call isBadVersion(5) -> true
call isBadVersion(4) -> true
Then 4 is the first bad version.
```

Example 2:

```
Input: n = 1, bad = 1
Output: 1
```

SHIFTKEY ACADEMY

Binary Search

Use binary search to efficiently find the first bad version by narrowing down the search range based on isBadVersion(mid).

```cpp
// The API isBadVersion is defined for you.
// bool isBadVersion(int version);

class Solution {
public:
    int firstBadVersion(int n) {
        int low = 0;
        int high = n;
        while (low <= high) {
            int mid = low + (high - low)/2;
            if (isBadVersion(mid) == false) {
                low = mid + 1;
            }
            else{
                high = mid -1;
            }
        }
        return low;
    }
};
```

Time complexity: O(log n)
Space complexity: O(1)

SHIFTKEY ACADEMY

Quiz

SHIFTKEY ACADEMY

Reference

https://github.com/changgyhub/leetcode_101

https://github.com/halfrost/LeetCode-Go

https://github.com/krahets/hello-algo

SHIFTKEY ACADEMY

THANK YOU TO OUR PARTNERS

cognizant®

NOVA SCOTIA

DALHOUSIE UNIVERSITY | COMPUTER SCIENCE

SHIFTKEY ACADEMY

We want to hear from you!

Scan the QR Code to provide your feedback