CODING PATTERNS FOR TECHNICAL INTERVIEWS AND COMPETITIVE PROGRAMMING

By: Juan Peñuela

JUAN PEÑUELA

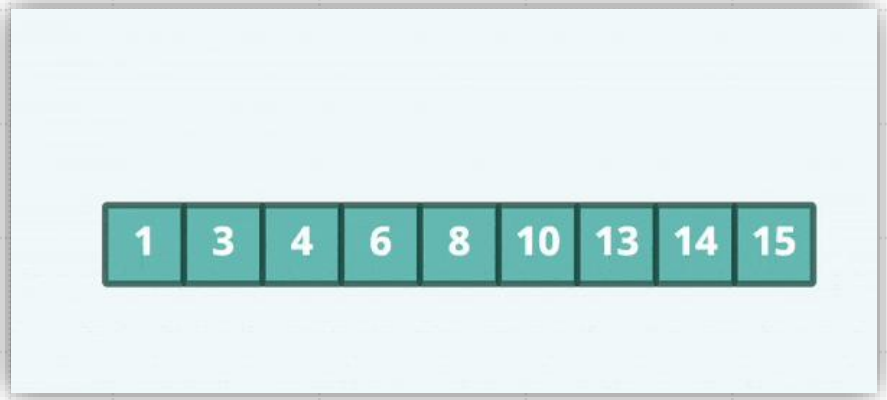
- Senior in C.S.
- 2x Intern at USAA
- Incoming SWE Analyst at BNY

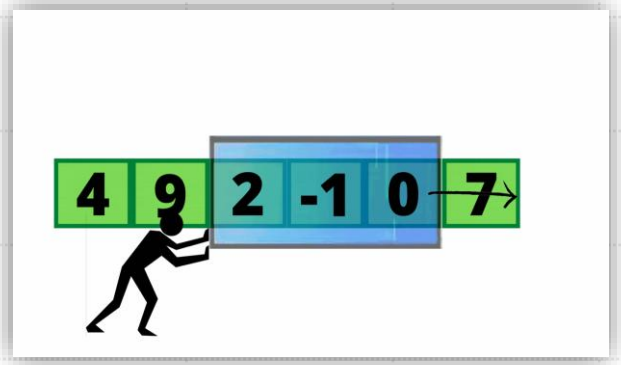
Let's Connect!

WHAT YOU'LL LEARN TODAY

- Overview of common coding patterns seen in easy/intermediate Leetcode questions for interviews or competitive programming.
- Two Pointer/Sliding Window
- Binary Search
- Breath First Search (BFS) and Depth First Search(DFS)
- Slight overview of Dynamic Programming

TWO POINTERS (SLIDING WINDOW)





- Refers to using two variables to traverse through a Data Structure (commonly arrays, strings, or linked lists)
- Commonly used to solve problems such as detecting pairs that involve a sorted list
- Usually optimizes solutions to linear time O(n)

- Common variations of this include:
  - Slow & Fast
  - Left & Right
  - Small & Big
  - You get it…

LET'S PRACTICE!

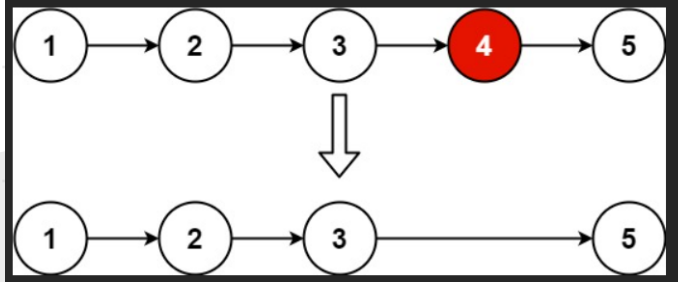
- 19. Remove Nth node from end of list
- 5. Longest Palindromic Substring

19. Remove Nth Node From End of List

Medium | Topics | Companies | Hint

Given the `head` of a linked list, remove the `n^th` node from the end of the list and return its head.

Example 1:




```
Input: head = [1,2,3,4,5], n = 2
Output: [1,2,3,5]
```

Example 2:

```
Input: head = [1], n = 1
Output: []
```

Example 3:

```
Input: head = [1,2], n = 1
Output: [1]
```

5. Longest Palindromic Substring

Medium | Topics | Companies | Hint

Given a string `s`, return *the longest palindromic substring* in `s`.

Example 1:

```
Input: s = "babad"
Output: "bab"
Explanation: "aba" is also a valid answer.
```

Example 2:

```
Input: s = "cbbd"
Output: "bb"
```

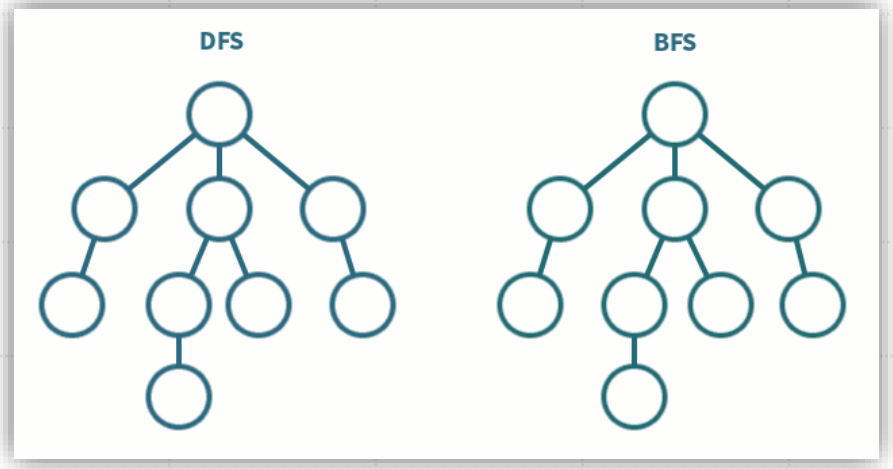
TRAVERSING GRAPHS

Depth First Search (DFS)

- Explores as far down a branch as possible, then backtracking to the next node.
- This backtracking is normally done using a stack or recursion.
- Often used for tree and **graph traversal**.

Breath First Search (BFS)

- Explore a graph by level, visiting all the nodes at the current height before moving on to the next one
- Useful in **tree problems** that involve processing nodes by level (or height)



LET'S PRACTICE!

- 100. Same Tree
- 101. Symmetric Tree

- *Hint: These are not the same*

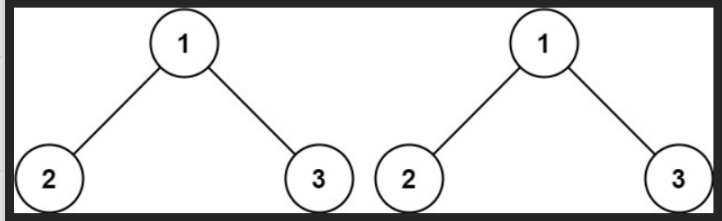
100. Same Tree

Easy | Topics | Companies

Given the roots of two binary trees `p` and `q`, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

Example 1:



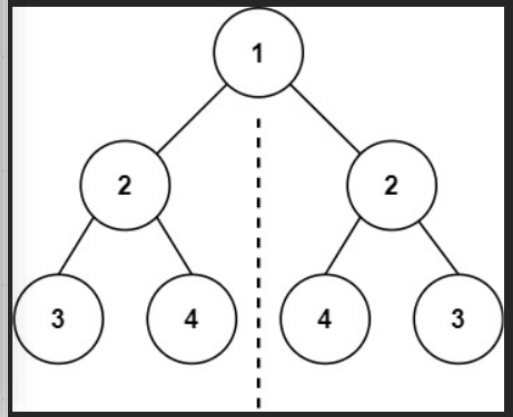

```
Input: p = [1,2,3], q = [1,2,3]
Output: true
```

101. Symmetric Tree

Easy | Topics | Companies

Given the `root` of a binary tree, *check whether it is a mirror of itself* (i.e., symmetric around its center).

Example 1:



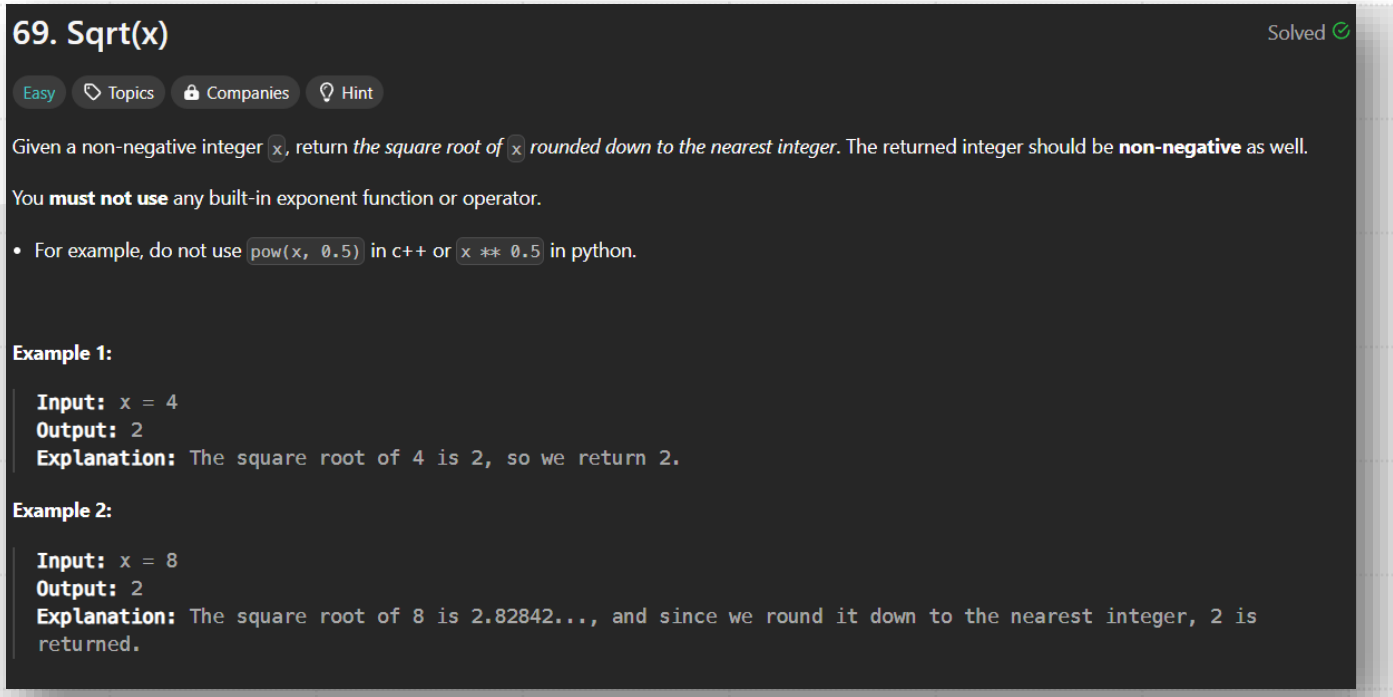

```
Input: root = [1,2,2,3,4,4,3]
Output: true
```

BINARY SEARCH

- Efficient for finding an element in a **sorted** list by repeatedly dividing the search in half.
- It compares the target value with the middle of the current search range and eliminating half of the search at each step.
- This is ideal for search questions, in which you are given a sorted list.
- It is very time efficient (O(log n)), much faster than a normal O(n) search.

LET'S PRACTICE!

- 69. Sqrt(x)
- (yes, this is a binary search question)



69. Sqrt(x)

Solved

Easy | Topics | Companies | Hint

Given a non-negative integer `x`, return *the square root of* `x` *rounded down to the nearest integer*. The returned integer should be **non-negative** as well.

You **must not use** any built-in exponent function or operator.

- For example, do not use `pow(x, 0.5)` in c++ or `x ** 0.5` in python.

Example 1:

```
Input: x = 4
Output: 2
Explanation: The square root of 4 is 2, so we return 2.
```

Example 2:

```
Input: x = 8
Output: 2
Explanation: The square root of 8 is 2.82842..., and since we round it down to the nearest integer, 2 is returned.
```

DYNAMIC PROGRAMMING

- Scenario: Imagine solving the typical Fibonacci number problem. It works for small numbers.
- When the input gets too large, this solution is very slow (O(n!))
- Dynamic programming comes in when subproblems like this one overlap. It involves saving solutions to the subproblems that build up to the solution of the main problem

```
public int slowfibonacci(int n)
{
    // Base Case
    if (n <= 1)
        return n;

    // Working recursive fibonacci
    return slowfibonacci(n - 1) + slowfibonacci(n - 2);
}
```

DYNAMIC PROGRAMMING

- We call this process **memoization**, and it involves using an extra data structure to save the solutions to the smaller subproblems, this makes solving the big problems faster (Generally O(n))
- There's two types of DP:
  - Top down
  - Bottom up

```java
public int coolfibonacci(int n)
{
    // Base case
    if (n <= 1) return n;

    // Create an array to store Fibonacci values up to n
    int[] dp = new int[n + 1];
    dp[0] = 0; // Fibonacci(0) = 0
    dp[1] = 1; // Fibonacci(1) = 1

    // Fill the dp array by iterating from 2 to n
    for (int i = 2; i <= n; i++) {
        dp[i] = dp[i - 1] + dp[i - 2];
    }

    return dp[n]; // Return Fibonacci(n)
}
```

PRACTICE PROBLEM (NOT LEETCODE)

- Doodoo Dynamics buys long steel rods, cuts them into shorter rods, and sells them. Cutting doesn't have a cost and only happens in whole inches
- This table shows the current sale prices for different length rods

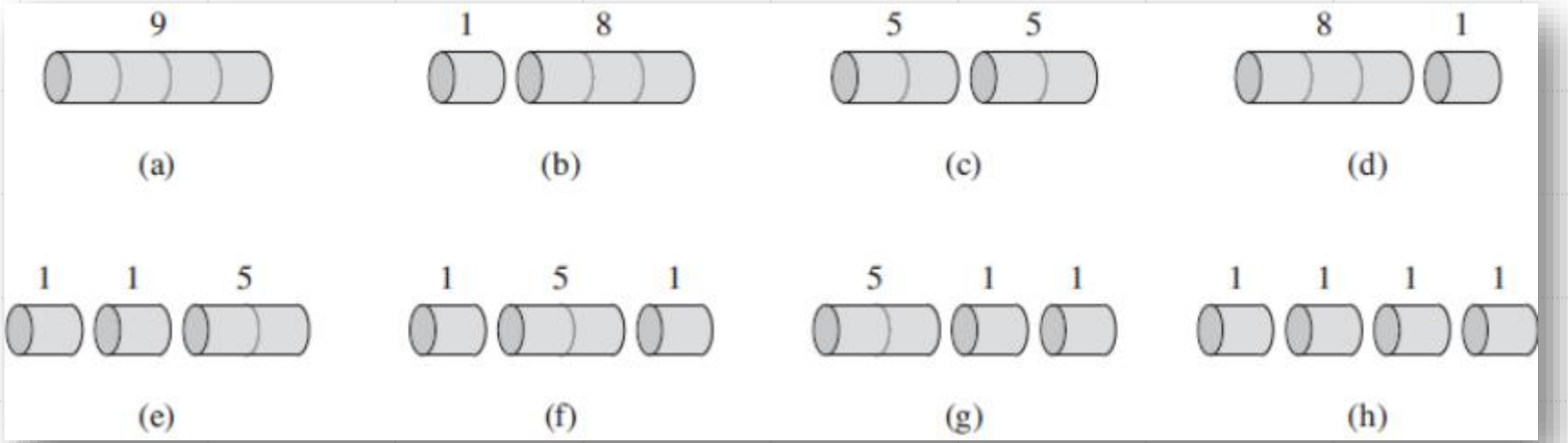
length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

- Given a rod of length n, determine the maximum revenue that can be obtained

HINTS

- Which of these rods yields the most revenue?

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30



SOLUTION

```
CUT-ROD(p, n)
1  if n == 0
2      return 0
3  q = -∞
4  for i = 1 to n
5      q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```

QUESTIONS?