# Uber Career Prep Homework Assignment 2

Due: Wednesday, May 2, 2023

## Linked List & Tree Problem-Solving Techniques

These questions should be completed without Googling the answer, which in most cases is easily available on the internet. That would defeat the point of the assignment, which is for *you* to practice interviewing skills. You use your own knowledge as it stands. There are two exceptions to this rule. Firstly, if you are unsure of syntax (e.g., method names, constructor arguments, etc.), you may look it up. Secondly, if you do not know a term, you may look it up. In some cases, depending on your academic schedule, you may not yet have covered a data structure yet, in which case we *encourage* you to learn and study the general concepts *before* attempting the problem.

Some questions may seem vague. This is intentional, as interview questions are often vague. Avoid making unnecessary assumptions, including based on example input/output, which is not exhaustive. If you *must* make an assumption, state your assumption in a comment in your code.

# Part 1: Data Structure Implementations (~3 hrs)

## Instructions

For each problem, implement the data structure from scratch (i.e., not using the built-in version in your language of choice even if one exists) according to the provided API. For each method, please indicate the time complexity of your implementation in a comment.

If you are unfamiliar with any of the data structures you are asked to implement, please read the appropriate article defining the data structure listed below before attempting the problem. You're welcome to consult other internet resources on the definition and use cases of these data structures, but **do not** read any implementations before attempting these problems. (If you've seen or created implementations in the past, of course that's okay, but do these problems without consulting them.) Even if you do not do the bonus questions, make sure you are familiar with the data structures because you will need them for Part 2 of the assignment.

- Linked Lists (Questions 1-2)
- Tree Traversals (Question 3; Part 2)
- Binary Search Trees (Question 3)
    - BST operations visualization
- Queues (Bonus Question 1)
- Stacks (Bonus Question 2)

You also might find the videos such as the following helpful for visiualizng recursion and the tree traversals:

- Recursion
- Pre-order traversal
- In-order traversal
- Post-order traversal

The core implementations do not have a maximum time; you will need them to understand in order to do the rest of the assignment. If you are stuck after 40 min, please get help from your mentor and/or consult resources. If you do need to consult other resources, be sure to then re-implement what you learn on your own without consulting resources while coding.

## Question 1: Singly Linked List

Implement the following methods. Rather than having a separate linked list class, we will pass a `Node` struct that represents the head of the list (this is common practice in interview questions). The linked list article includes a `Node` struct definition in a number of common languages (C++, Python, Java, JavaScript); feel free to use it in your implementation. For simplicity, you can make your nodes store integers rather than generic data types. In each of the methods, you should use pointers in languages that support pointers (e.g., `Node *` in C++) or a reference in languages that support references (e.g., Python).

```
Node insertAtFront(Node head, int val) // creates new Node with data val at front; returns
new head
void insertAtBack(Node head int val) // creates new Node with data val at end
void insertAfter(Node head, int val, Node loc) // creates new Node with data val after Node
loc
Node deleteFront(Node head) // removes first Node; returns new head
void deleteBack(Node head) // removes last Node
Node deleteNode(Node head, Node loc) // deletes Node loc; returns head
int length(Node head) // returns length of the list
Node reverseIterative(Node head) // reverses the linked list iteratively
Node reverseRecursive(Node head) // reverses the linked list recursively (Hint: you will
need a helper function)
```

## Question 2: Doubly Linked List

Copy-paste your implementation from Question 1 and modify it. Your `Node` struct should have an additional `prev` reference as well as a `next`.

```
Node insertAtFront(Node head, int val) // creates new Node with data val at front; returns
new head
void insertAtBack(Node head int val) // creates new Node with data val at end
void insertAfter(Node head, int val, Node loc) // creates new Node with data val after Node
loc
Node deleteFront(Node head) // removes first Node; returns new head
void deleteBack(Node head) // removes last Node
Node deleteNode(Node head, Node loc) // deletes Node loc; returns head
int length(Node head) // returns length of the list
Node reverseIterative(Node head) // reverses the linked list iteratively
Node reverseRecursive(Node head) // reverses the linked list recursively (Hint: you will
need a helper function)
```

## Question 3: Binary Search Tree

In each of the methods, use pointers in languages that support pointers (e.g., `Node *` in C++) or references in languages that support references (e.g., Python). Your `Node` struct will be the same as for your doubly linked list except the two pointers/references should be `left` and `right` rather than `next` and `prev`. Note that the `delete` method is more difficult than the `insert` method; you won't need it for the rest of the assignment so either stop or get help from your mentor if you are stuck after 40 minutes.

```
class BinarySearchTree {
    int min() // returns the minimum value in the BST
    int max() // returns the maximum value in the BST
    bool contains(int val) // returns a boolean indicating whether val is present in the BST
    // For simplicity, do not allow duplicates. If val is already present, insert is a no-op
    void insert(int val) // creates a new Node with data val in the appropriate location
    int delete(int val) // deletes the Node with data val, if it exists
}
```

***The following are bonus questions. They are optional.***

## Bonus Question 1: Queue

Queues are not a "first-class" data structure – they are an API or interface implemented using another underlying data structure. Queues should have O(1) insertion and deletion; because insertion and deletion occur at opposite ends, the underlying data structure is normally a linked list to satisfy this requirement. Implement a queue class according to the following definition; you may copy-paste and reuse any parts of your solution to Question 1. Again, for simplicity, you can make your queue store integers rather than generic data types. Queues are necessary auxiliary data structures to implement breadth-first traversals of graphs and trees. For those problems, you should use your language's built-in; we are asking you to implement a queue here for two reasons: to demonstrate one of the use cases of linked lists and to give you an understanding of how they work.

```
class Queue {
    int peek() // returns the first item in the queue
    void enqueue(int x) // adds x to the back of the queue
    int dequeue() // removes and returns the first item in the queue
    bool isEmpty() // returns a boolean indicating whether the queue is empty
}
```

## Bonus Question 2: Stack

Stacks are also not a "first-class" data structure and should also have O(1) insertion and deletion. Because insertion and deletion occur at the same end, this can be achieved with either an array or a linked list. Implement a queue class according to the following definition using a linked list as the underlying data structure; you may copy-paste and reuse any parts of your solution to Question 1. Again, for simplicity, you can make your stack store integers rather than generic data types. You will likely implement your depth-first traversals of graphs and trees recursively; however, if you chose to do them iteratively, you would need to use a stack.

```
class Stack {
    int top() // returns the top item in the stack
    void push(int x) // adds x to the top of the stack
    int pop() // removes and returns the top item in the stack
    bool isEmpty() // returns a boolean indicating whether the stack is empty
}
```

# Part 2: Linked List & Tree Problems (~5 hrs)

## Instructions

For each problem, identify the appropriate linked list or tree technique from Workshop 2. **State the technique in a comment at the top of your file**. As a reminder, the options are:

**Linked List Two Pointer Techniques:**

1. Linked list fast-slow two-pointer
2. Linked list fixed-distance two-pointer
3. Linked list reset/catch-up two-pointer
4. Simultaneous iteration two-pointer
5. Doubly linked list forward-backward two-pointer

**Other Linked List Techniques:**

1. Hash linked list nodes
2. Linked list recursion

**Tree Traversal Techniques:**

1. Depth-first traversal
   a. Pre-order
   b. In-order
   c. Post-order
   d. Generic (type doesn't matter – all work)
2. Level-order (breadth-first) traversal
3. Search binary search tree (BST)

Then, write a function to solve the problem and write test cases to check your function. When run, the file you submit should execute your function on your test cases (e.g., through a main method, if applicable in your language). **State the time and space complexity of your solution in a comment at the top of your file**.

Time how long you spend on each problem. You should actively **work on each problem for a MAXIMUM of 40 minutes**. Once 40 minutes has elapsed, submit whatever you have, regardless of whether you are finished. Please indicate in a comment at the bottom of your file how long you spent on the problem. It is important that you are honest about how long each problem took you as it will help your mentor help you!

## Question 4: CopyTree

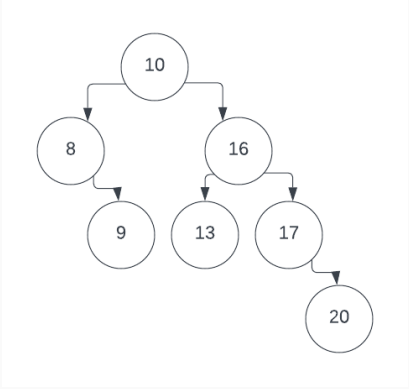
Given a binary tree, create a deep copy. Return the root of the new tree.

## Question 5: IsBST

Given a binary tree, determine if it is a binary search tree.

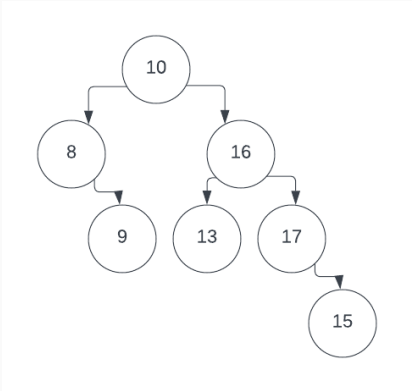
Examples:

```
Input:
```




Output: True

Input:




Output: False

## Question 6: DedupSortedList

Given a sorted singly linked list, remove any duplicates so that no value appears more than once.

Examples:

Input:


Output:


Input:


Output:


## Question 7: MoveNthLastToFront

Given a singly linked list, move the nth from the last element to the front of the list.

Examples:

Input: k=2


Output:


Input:k=7

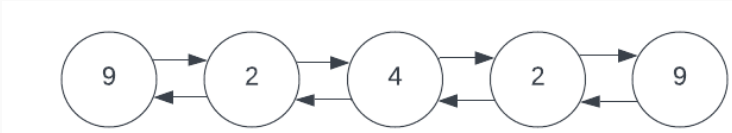

Output:


## Question 8: IsPalindrome

Given a doubly linked list, determine if it is a palindrome.

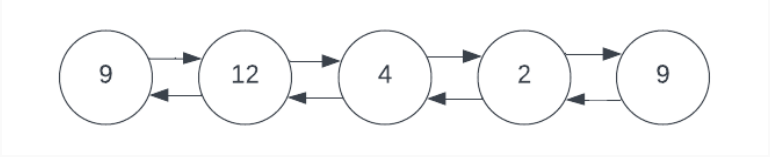
Examples:

Input:




Output: True

Input:



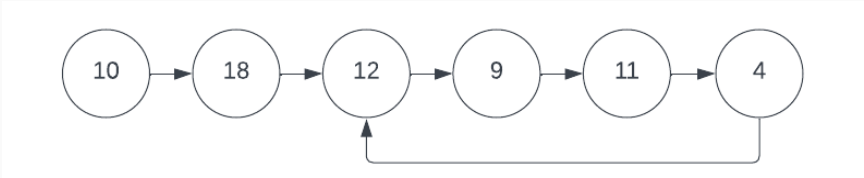

Output: False

## Question 9: DisconnectCycle

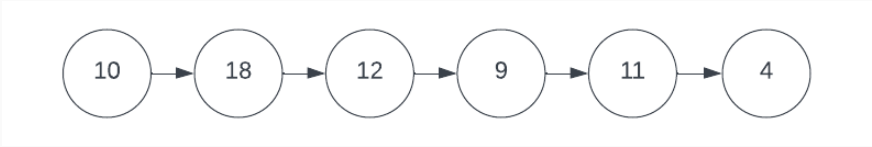
Given a singly linked list, disconnect the cycle, if one exists.

Examples:

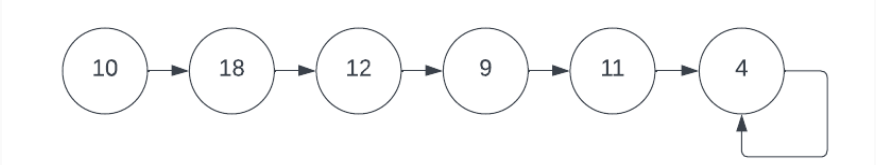
Input:



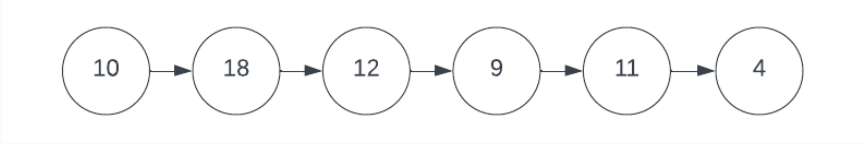

Output:




Input:




Output:

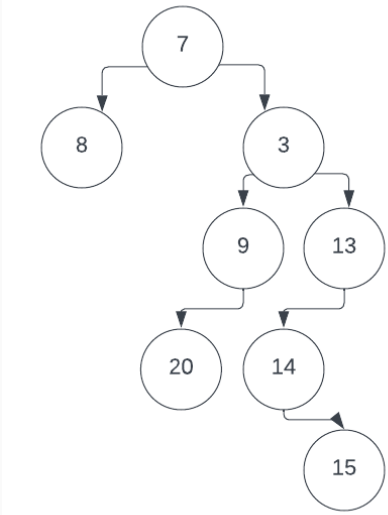



## Question 10: LeftView

Given a binary tree, create an array of the left view (leftmost elements in each level) of the tree.

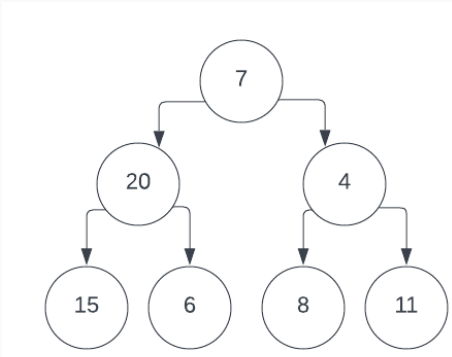
Examples:

Input:




Output: [7, 8, 9, 20, 15]

Input:



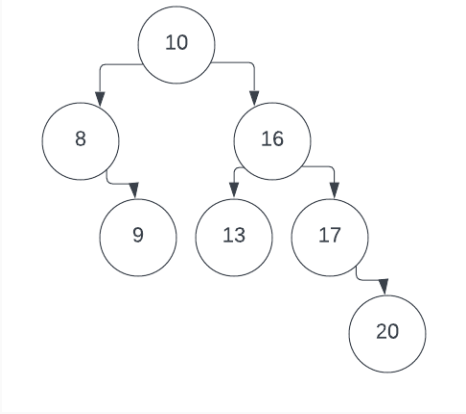

Output: [7, 20, 15]

## Question 11: FloorInBST

Given a target numeric value and a binary search tree, return the floor (greatest element less than or equal to the target) in the BST.

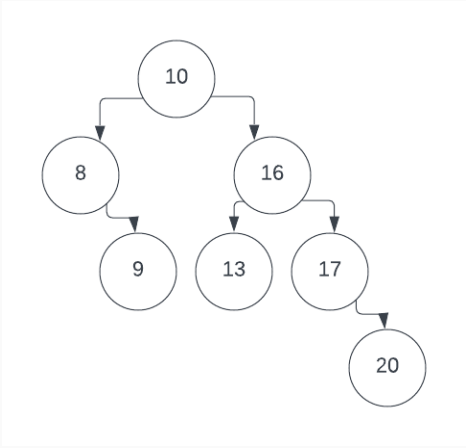
Examples:

Input: target=13




Output: 13

Input: target=15




Output: 13