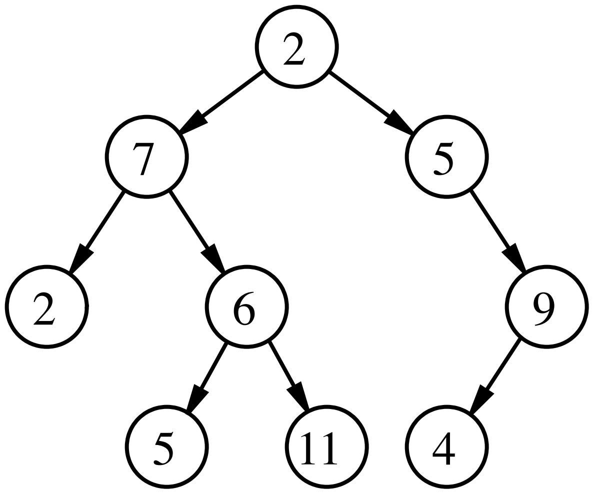
Interview Prep Course

Unit 3 - Trees, Binary Trees, Searches, Heaps



Topics

- Trees
- Binary Trees
- Searches
- Heaps

Before the Weekly Practice Session

Check out the tasks before session tab to review the required tasks **before attending the weekly practice session**. Due before Self-paced.

After the Session

Check out the tasks after session tab to review the required tasks **after attending the weekly practice session.** Due before Self-paced.

Interview Prep Course

Tasks Before Session 3

Complete the following tasks **before the session** this week:

- Carefully review core concept videos and topic links for the weekly topics
- Complete 1 problem **from at least 2 category buckets** for any 2 of the following: Trees and Binary Search and Heaps problem sets. This task requires solving a minimum of 4 problems (2 for each topic).
- Prepare for your mock interview session by reviewing one of the questions and being prepared to interview another person by asking them this question. **Note:** If your birthday **day is an odd number**, prepare to ask someone question #1, otherwise if your **birthday day is an even number** then prepare to ask someone question #2.

Note: InterviewBit does not always allow for problems to be solved in every language and as a result **not all questions can be solved in Objective-C or Javascript**. If you are encountering problems, we recommend you either take questions from interviewbit and **push the solutions in your language of choice to your git repo** or you can switch to solving problems in Java. If you are switching to Java, you can use this Java syntax cheatsheet as a reference.

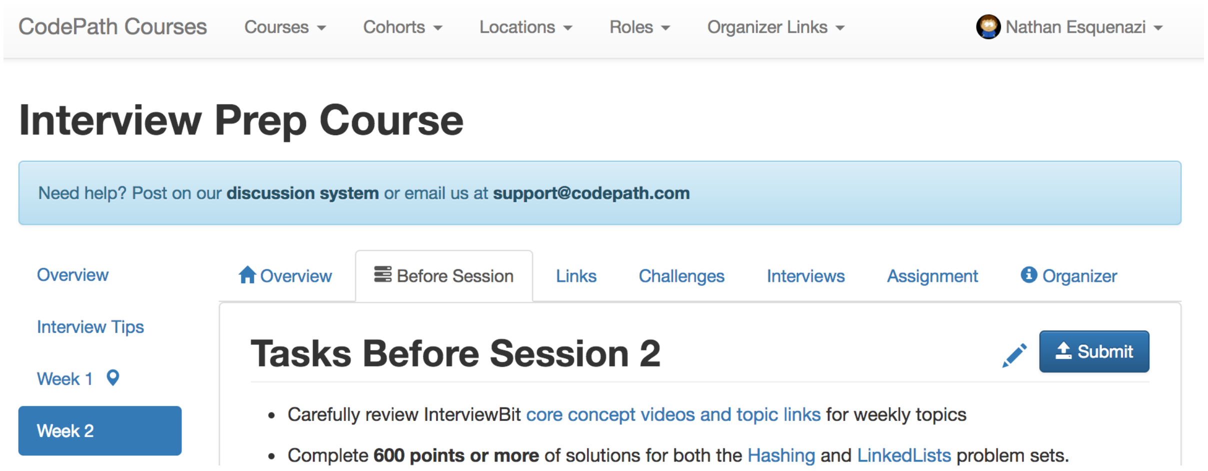
Submitting the Tasks

Once you've completed these exercises, submission involves recording a GIF of your profile / topic pages after completion.

First, go to your profile on interviewbit and record a GIF of the profile and/or topic pages that **indicates you've completed each subcategory**.

For the repository, link to your **github project repo for code solutions**. You are not required to put all your solutions into this repo, but you are encouraged to put ones you think will be most helpful to review in the future.

Next, go to the "Before Session" tab for the project and click the "Submit" button on the right-hand side:



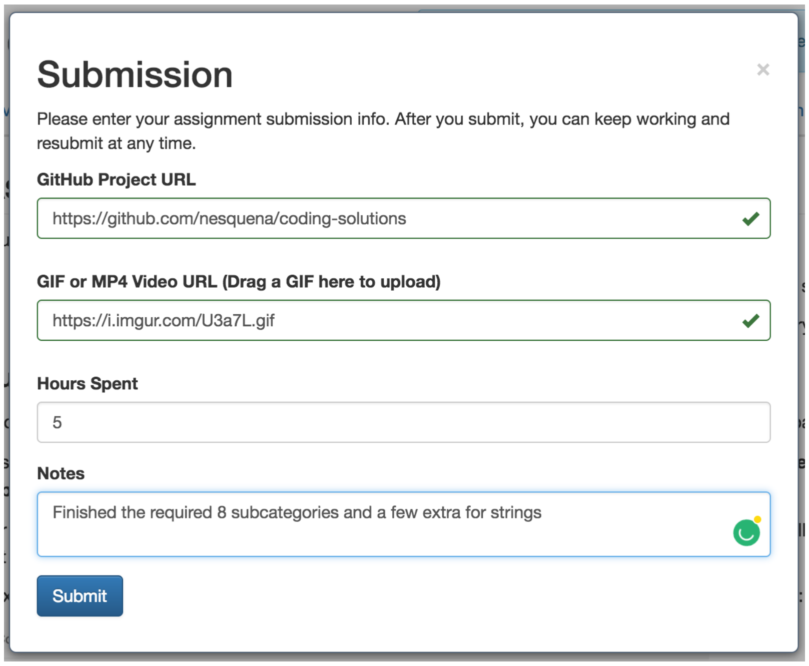
Interview Prep Course

Need help? Post on our **discussion system** or email us at **support@codepath.com**

Tasks Before Session 2

- Carefully review InterviewBit core concept videos and topic links for weekly topics
- Complete **600 points or more** of solutions for both the Hashing and LinkedLists problem sets.

In the dialog that appears, enter the required information about your project:



Submission

Please enter your assignment submission info. After you submit, you can keep working and resubmit at any time.

GitHub Project URL

https://github.com/nesquena/coding-solutions

GIF or MP4 Video URL (Drag a GIF here to upload)

https://i.imgur.com/U3a7L.gif

Hours Spent

5

Notes

Finished the required 8 subcategories and a few extra for strings

Then press "Submit" at the bottom to finalize your submission. Note that **you can always update your submission at any time** in case you want to re-upload the GIF or update the hours spent.

Interview Prep Course

Links - Trees, Binary Trees, Searches, Heaps

Trees and Binary Trees

- InterviewCake Binary Trees

Core Concept Videos:

- HackerRank Video: Trees
- HackerRank: Tree is a BST

Problem Sets:

- Problem Set: Trees

Additional Videos:

- Intro to Trees
- Binary Tree
- Binary Search Tree
- Implementing a Binary Search Tree
- Inorder successor in BST

Searches

- InterviewCake Binary Search
- Binary Search Succinctly Guide

Core Concept Videos:

- HackerRank Video: Binary Search
- Count Occurences in Sorted Array

Problem Sets:

- Problem Set: Binary Search

Additional Videos:

- Intro Binary Search
- Implementing Binary Search
- Count Rotations in Array

- Ice Cream Parlor Problem

Heaps

- Understanding Heaps

Core Concept Videos:

- HackerRank Video: Heaps
- HackerRank: Continuous Median with Heaps

Problem Sets:

- InterviewBit Heaps

Additional Videos:

- Intro to Heaps
- Removing an Item
- Traversing a Heap
- MIT Course: Heaps (Long video)

Interview Prep Course

Challenges

The challenge session will **be around 45 minutes** and involves collaboratively solving the suggested problems as a group. Participants are split into groups of 3 and should do the following:

1. **Introductions** - Each person should introduce themselves to the other in the group including name, where you work, and which bootcamp you took (unless everyone has already met).
2. **Create a codepad** - Create a new collabedit document and share the URL with the other members. You'll be using this to collaboratively solve the problem.
3. **Talk through problems** - Pick any challenge and then begin to work through this problem together. Not only solving the problem but also whiteboarding the solution and discussing the solution. This isn't just about getting a correct solution but also effective communication.
    - **Tip:** Use **pseudocode rather than a particular language** when solving problems so that others in your group that prefer a different language can still understand and participate.
4. **Review solution** - After completing a problem, scroll down to the bottom of this tab to find the [interviewbit solutions repo] and discuss / compare your solution with the solution given. Is the one you've developed less efficient or less concise?

Repeat this and work through as many challenges as you can in the time alotted. At the end, if you believe you have a better solution than the one provided, submit a PR here to improve the solution set.

Problems

Here is the set of challenge problems for this week:

- Challenge 1 - Identify Binary Search Trees
- Challenge 2 - Count Leaf Nodes
- Challenge 3 - Compare Trees
- Challenge 4 - Path Sum with BFS
- Challenge 5 - Path Sums with DFS

Setup

For all following challenge questions, we'll consider a simple tree implementation as defined by a single class, `TreeNode`. Each instance of `TreeNode` represents a single node in the tree and encapsulates its value and its child nodes:

```java
public class TreeNode<E> {

    E value;
    TreeNode<E> left, right;

    public TreeNode(E value) {
        this.value = value;
        this.left = right = null;
    }

}
```

Challenge 1 - Identify Binary Search Trees

Write a program that takes as its input an instance of `TreeNode` representing the root of a tree and returns a `boolean` value which is true if the input is a binary search tree and false if it is not.

Challenge 2 - Count Leaf Nodes

Write a program that takes as its input an instance of `TreeNode` representing the root of a tree and returns a `int` value representing the number of nodes in the tree with no children.

Challenge 3 - Compare Trees

Write a program that takes as its input *two* instances of `TreeNode` and returns a `boolean` value which is true if both trees have an identical structure with the same node values.

Challenge 4 - Path Sum with BFS

Write a program that takes as its input an instance of `TreeNode<Integer>` representing the root of a BST and a target `int` value and returns a `boolean` value which is true if the tree contains a path between the root and any leaf such that the sum of all the values of the nodes on the path equal the target value.

Hint: Use a breadth-first search

Example: Given a target value of `22` and a BST with the following structure:

```
BST:
          5
         / \
        4   8
       /   / \
      11  13  4
     /  \      \
    7    2      1
```

The path `5→4→11→2` sums to `22`, so the output in this case would be `true`.

Challenge 5 - Path Sums with DFS

Write a program that takes as its input an instance of `TreeNode<Integer>` representing the root of a BST and a target `int` value and returns an `int[][]` containing the node values of all paths between the root and any leaf such that the sum of all the values of the nodes on the path equal the target value.

Hint: Use a depth-first search

Example: Given a target value of `22` and a BST with the following structure:

```
BST:
          5
         / \
        4   8
       /   / \
      11  13  4
     /  \    / \
    7    2  5   1
```

There exist two paths from root to leaf that sum to 22: `5→4→11→2` and `5→8→4→5`, so the output in this case would be an array of two arrays containing those values:

```
int[][] output = {
  { 5, 4, 11, 2 },
  { 5, 8, 4, 5 }
};
```

Solutions

- Solution for Challenge #1
- Solution for Challenge #2
- Solution for Challenge #3
- Solution for Challenge #4
- Solution for Challenge #5

Interview Prep Course

Interview Questions

The interview session will **be around 60 minutes** and involves taking turns as interviewer and interviewee. Participants are split into pairs and should do the following:

1. **Introductions** - Introduce themselves to the other person (unless they have already met).
2. **Create a Codepad** - Create a new collabedit document and share the URL with the other members. You'll be using this to solve the problem.
3. **Interview Question 1** - The person that has prepared for interview question 1 asks the question to their partner. The partner tries to understand and then solve the question for 30 minutes.
4. **Interview Question 2** - The person that has prepared for interview question 2 asks the question to their partner. The partner tries to understand and then solve the question for 30 minutes.

At the end, if you believe you have a better interview solution than the one provided (or if there is no provided solution), submit a PR here to improve our solution set.

Interview Question 1 - Convert a Linked List to BST

Imagine an interview in which you are presented with the following problem:

Write an algorithm which takes as its input a linked list of integer values and outputs a balanced binary search tree in *O(n)* time.

Before beginning, consider the above statement carefully, not just in terms of which details were specified but also in terms of which details were omitted.

There are really *two* problems being presented here: not just the algorithm, but also the interview question itself. In both cases you have to show your work: not just the code for the solution, but the process of engaging with the problem and designing the solution. Your first job as a programmer is to make sure the problem is clearly defined and understood. In interviews (and reality) this is a verbal/visual and often interactive process.

One way to start is by restating the problem with more specifics and observations, especially with regard to any implicit requirements arising from the explicitly stated ones. For instance, here's an example of how one might restate the above problem:

*Since the input is in the form of a linked list, we won't have the benefit of a data structure that supports random access, such as an array. And given the time requirement of **O(n)**, it's likely the solution will involve a straightforward traversal of the data set--the amount of work we're doing needs to scale linearly with the size of the input list. Additionally, we'll have to consider the depth of the leaf nodes to ensure the output BST is balanced.*

But almost as much information is left unspecified. Rather than assume that any vagueness on the part of the interviewer is an oversight or intended as latitude, consider this may be intentional in order to reveal which questions you know to ask. Some examples:

- Is the linked list single-linked or doubly-linked?
- Are its values sorted? Are duplicate values ruled out?
- Is there a space complexity requirement in addition to the time requirement? (i.e., is using storage allowed)?
- Are there specific implementations of the data structures that should be used? (i.e., using a standard collection class vs. implementing a model of the structure)
- Is test case data available?

You should be able to think of questions such as these and others for almost any problem you are given. Even (especially) if you have no idea how to approach the problem.

For this exercise, let's assume the following further answers:

- Single-linked list of sorted, unique values
- No space complexity requirements
- The input / output data structures must be implemented, otherwise collections are allowed
- No example/test data, you're on your own

From the above, we know a few more things:

- We can use extra storage, which generally opens up more possible approaches to a solution, especially since we can use collections classes such as Stack or Queue for that part
- The linked list type points us towards straightforward a linear algorithm, probably in one or two passes
- The input won't require a lot of pre-processing, so we should focus on building the BST
- We'll need to implement the input list structure and output tree structure first to frame the algorithm

At this point, you have enough information to at least sketch the solution in code or pseudo-code. You should be able to implement a simple single-linked list and binary tree from memory to demonstrate your knowledge of the structure. You could also stub out the solution using something as simple as a static method that declares these in its signature.

Before you begin to write the algorithm code, think about the approach to the solution: will it be recursive or iterative? What's the space complexity of the solution structure? Any additional structures that would simplify the approach? Is there a way to ensure the output tree is balanced while constructing it?

Hint: there is a solution with a space complexity of *O(n)* using no helper collections

Bonus 1 - Use a Doubly-linked List

How would you improve the algorithm given a doubly-linked list as input?

Hint: think about the difference between a DLL and a BST

Interview Question 2 - Sort tree by columns

Write an algorithm which takes as its input a binary tree of chars and outputs a singly-linked list of its values ordered by column first and row second.

- The root node is in row `0`
- Any node that is a child of another node will be in the row immediately *following* that of its parent
- Any node that is a *left* child will be in the column immediately *preceding* that of its parent
- Any node that is a *right* child will be in the column immediately *following* that of its parent
- Any node that is a *left* child with a parent that is a *right* child will be in the same column as its grandparent
- Any node that is a *right* child with a parent that is a *left* child will be in the same column as its grandparent
- If two nodes share the same row and column, the node *whose parent is a left child* comes first

Example:

```
Input tree:
          5
         / \
        4   8
       /   / \
      9   6   4
     / \     / \
    7   2   5   1
```

In the above tree, there are 7 columns, three of which have two nodes: `[4, 2]`, `[5, 6]`, and `[8,5]`. Therefore the algorithm would return a linked list with the following characters: `7→9→4→2→5→6→8→5→4→1`.

Bonus 1 - Print the tree

Adapt the algorithm to print the tree to match the output above. If two nodes share the same row and column, print an `*` in place of the value:

```
Input tree:
          5
         / \
        4   8
       / \ / \
      9   *   4
     / \   \ / \
    7   2   *   1
```

Interview Prep Course

Assignment 3

You are required to complete a few additional items this week after the session:

- **Required:** Complete 1 problem **from at least 2 additional category buckets** for both the Trees and Binary Search problem sets. This requires you've completed a **total of 1 problem for 8 category buckets** (4 per topic).
- **Optional:** Complete 4 more problems of your choosing between the Trees, Binary Search, and Heaps problem sets.

Note: InterviewBit does not always allow for problems to be solved in every language and as a result **not all questions can be solved in Objective-C or Javascript**. If you are encountering problems, we recommend you either take questions from interviewbit and **push the solutions in your language of choice to your git repo** or you can switch to solving problems in Java. If you are switching to Java, you can use this Java syntax cheatsheet as a reference.

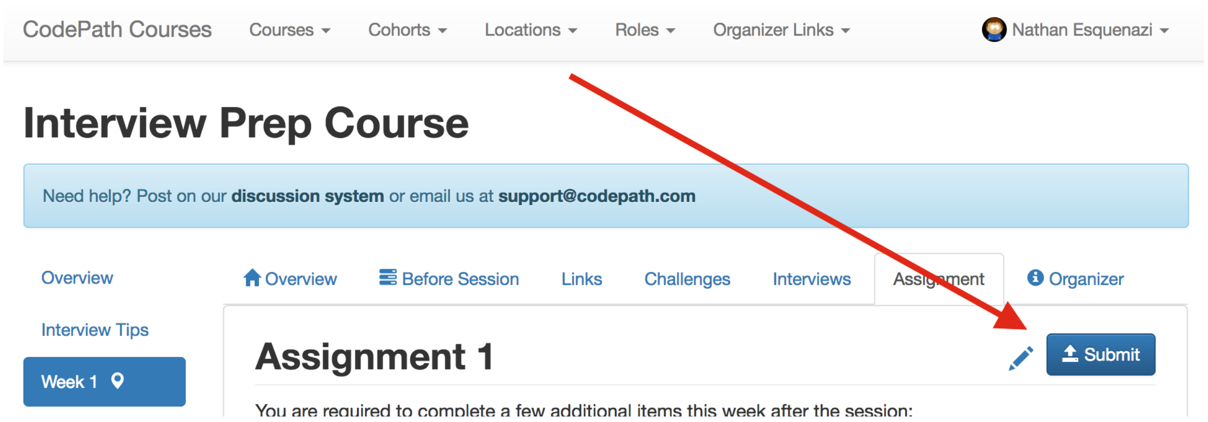
Submitting the Assignment

Once you've completed this assignment, submission involves recording a GIF of your profile / topic pages after completion.

First, go to your profile on interviewbit and record a GIF of the profile and/or topic pages that **indicates you've completed each subcategory**.

For the repository, link to your **github project repo for code solutions**. You are not required to put all your solutions into this repo, but you are encouraged to put ones you think will be most helpful to review in the future.

Next, go to the "Assignment" tab for the project and click the "Submit" button on the right-hand side:



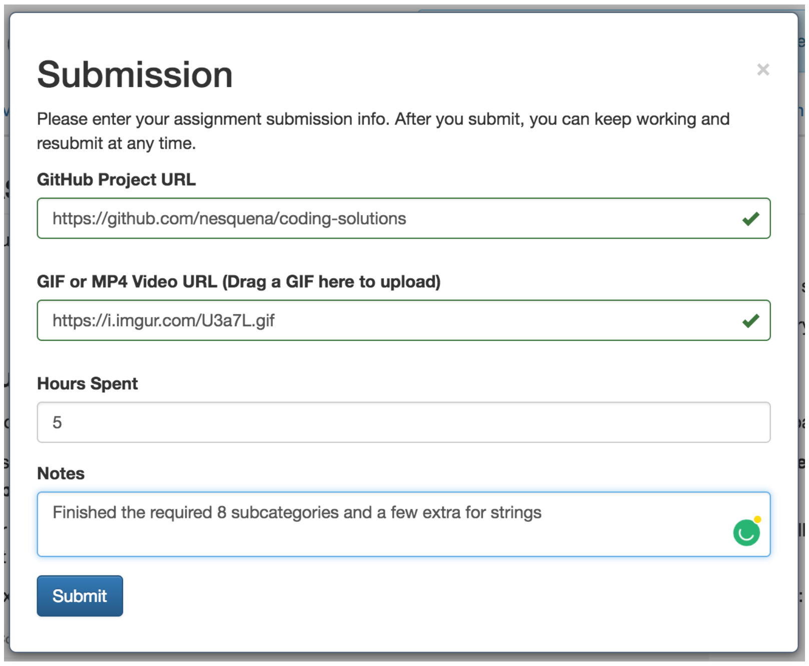
Interview Prep Course

Need help? Post on our **discussion system** or email us at **support@codepath.com**

Assignment 1

You are required to complete a few additional items this week after the session:

In the dialog that appears, enter the required information about your project:



Submission

Please enter your assignment submission info. After you submit, you can keep working and resubmit at any time.

GitHub Project URL

https://github.com/nesquena/coding-solutions

GIF or MP4 Video URL (Drag a GIF here to upload)

https://i.imgur.com/U3a7L.gif

Hours Spent

5

Notes

Finished the required 8 subcategories and a few extra for strings

Then press "Submit" at the bottom to finalize your submission. Note that **you can always update your submission at any time** in case you want to re-upload the GIF or update the hours spent.