CS5001 Silicon Valley

Spring 2019

HW8 (Binary Search (Divide and Conquer vs Brute Force) and Project Design)

Assigned: April 1, 2019

Deadline: April 8, 2019 11:59pm

Your solutions will be evaluated according to the following grading rubric.

Your completed work should be compressed and submitted to Bottlenose following this naming convention: `firstName_lastName_assignment8.zip`.

Writtten Component

The written component for this week's homework will be a design for your final project. At the end of this project, you will have implemented a working version of the game Connect Four. If you are unfamiliar with the rules, you can read them here. We will start by first designing the game. Design is a challenging aspect of Software Engineering, and this exercise is designed to expose you to a higher level of technical thinking.

Requirements

For this phase of the project, complete the following tasks:

1. Provide a list of functions needed to implement the game. Assume the game will be played by 2 human players, so there is no need to worry about an AI component. Consider using the Design Recipe:

```
# Purpose

# Signature

# Examples
```

2. Provide a list of classes needed to implement the game. For example, you might need a `board` object for the game.
3. Design and implement a flow-chart outlining the structure of your implementation.

Programming Component

Code Structure and Style

Follow the PEP 8 style guide. *Tip: Use the design recipe to write and document your functions.* Some reminders:

- Place each of the following programs in their own file
- Make sure that your functions are modular—one action per function
- Avoid "magic numbers" by using constants where appropriate
- Write Pytest test cases for every function (except main()). Write as many test cases as you need to feel confident that your code works as expected. We expect to see test cases even if the problem text doesn't explicitly mention test cases.

Programming #1

- Filename: brute_force_search.py

Implement a brute-force algorithm to find an Integer in a `List` of Integers. You may use whatever approach you like.

Use the following design recipe to write your function:

```
# Purpose
# This function takes a List and an Integer as input, and returns True if the
Integer is in the List and False otherwise.

# Signature
# brute_force_search :: (List, Integer) => Boolean

# Example
# brute_force_search :: ([1,2,3], 1) => True
# brute_force_search :: ([1,1,1,1], 2) => False
```

Requirements:

- Your function should expect a `List` and an `Integer` as input
- Your function should return a `Boolean` as a result
- Your function must include at least 3 test cases

Programming #2

- Filename: binary_search.py

We discussed Binary Search Trees and the Binary Search Algorithm in class. Implement a version of Binary Search that works on a `List`. You may assume the `List` is sorted in ascending order.

Use the following design recipe to write your function:

```
# Purpose
# This function takes a List and an Integer as input, and returns True if the
Integer is in the List and False otherwise.

# Signature
# binary_search :: (List, Integer) => Boolean

# Example
# binary_search :: ([1,2,3], 1) => True
# binary_search :: ([1,1,1,1], 2) => False
```

Requirements:

- Your function should expect a `List` and an `Integer` as input
- Your function should return a `Boolean` as a result
- Your function must include at least 3 test cases

Programming #3

- Filename: binary_search_tree.py

Implement a Binary Search Tree class in Python. Your class should support the `get` and `add` operations.

Requirements:

- Your class should support `get` and `add`. Do not implement any other operations.
- Your class methods should be well-tested.