DS 4300 - Spring 2025

Homework 01

Extra Credit Deadline: Sunday, January 12 2024 @ 11:59 pm
Regular Credit Deadline: Tuesday, January 14 2024 @ 11:59 pm

Directions:
1. From the File Menu, make a copy of this document in your own Google Drive account OR download an MS Word version.
2. Provide **professional-quality, type-set** solutions for each of the stated questions.
   - Do not delete the actual question statement. However, you may add additional space between the questions to accommodate your answers.
   - ***Your solutions must be typeset***. The solutions you incorporate into this document MAY NOT be handwritten. This includes not writing them on an iPad and then copying them (or a screen shot) into this document. I suggest you use some type of equation editor in your writing tool of choice or use LaTeX to typeset your responses where appropriate.
   - Any source code or pseudocode should be typed in a *monospaced font* like Courier New or Fira Mono, both available in Google Docs. **DO NOT paste screenshots of your code.**
   - To reiterate, handwritten solutions in any form will receive NO CREDIT.
3. **Generate a PDF of your document with your solutions**. DO NOT upload screenshots, images, or pictures. Reminder, DO NOT delete the question statements. If you compose your solutions in LaTeX, please retype the questions.
4. Upload your solutions document to GradeScope by the due date. **It is your responsibility to associate each question with your solution in GradeScope and click the submit button afterwards.** Failure to do so will result in no credit for any questions not linked with your solution *and will not be a valid reason to request a re-grade*.

Academic Collaboration Reminder:
Remember that you may not look at, copy, capture, screenshot, or otherwise take possession of any other students' solutions to any of these questions. Further, you may not provide your solutions in part or in whole to any other student. Doing any of the above constitutes a violation of academic honesty which could result in an F in this class and a referral to OSCCR.

What is permissible? You are free and encouraged to talk to your peers about the conceptual material from the lectures or the conceptual material that is part of this assignment. I'm confident you know where the line between collaborative learning and cheating sits. Please don't cross that line.

Question 1:

Linear search and Binary search aim to find the location of a specific value within a list of values (sorted list for binary search). The next level of complexity is to find two values from a list that together satisfy some requirement.

Propose an algorithm to search for a pair of values in an unsorted array of *n* integers that are closest to one another. Closeness is defined as the absolute value of the difference between the two integers. Your algorithm should not first sort the list. [10 points]

```
def search_pair_unsorted(arr):
    # initialize the difference as highest possible value
    diff = float('inf')
    pair = None
    # through the nested loop, calculate difference between each possible pair
    for i in range(len(arr)):
        for j in range(i+1, len(arr)):
            pair_diff = abs(arr[i] - arr[j])
            # smallest difference is set as the value of diff
            if pair_diff < diff:
                diff = pair_diff
                pair = (arr[i], arr[j])
    return pair
```

Next, propose a separate algorithm for a sorted list of integers to achieve the same goal. [10 points]

```
def search_pair_sorted(arr):
    diff = float('inf')
    pair = None

    # Iterate through the sorted array and calculate difference of consecutive pairs
    for i in range(len(arr) - 1):
        pair_diff = arr[i + 1] - arr[i]

        # update diff and pair with the smallest difference and return it
        if pair_diff < diff:
            diff = pair_diff
            pair = (arr[i], arr[i + 1])
    return pair
```

Briefly discuss which algorithm is more efficient in terms of the number of comparisons performed. A formal analysis is not necessary. You can simply state your choice and then justify it. [5 points]

The sorted list algorithm is more efficient as it has to go through less comparisons than the unsorted list. The unsorted algorithm has a nested loop to compare every possible pair which grows quadratically. On the other hand, the sorted list algorithm has to only compare adjacent pairs, significantly decreasing the number of calculations.

Question 2:

Implement in Python an algorithm for a level order traversal of a binary tree. The algorithm should print each level of the binary tree to the screen starting with the lowest/deepest level on the first line. The last line of output should be the root of the tree. Assume your algorithm is passed the root node of an existing binary tree whose structure is based on the following BinTreeNode class. You may use other data structures in the Python foundation library in your implementation, but you may not use an existing implementation of a Binary Tree or an existing level order traversal algorithm from any source.

Construct a non-complete binary tree[1] of at least 5 levels. Call your level order traversal algorithm and show that the output is correct. [20 points]

```
class BinTreeNode:
    def __init__(self, value=0, left=None, right=None):
        self.value = value
        self.left  = left
        self.right = right


def rev_traversal(root):
    #declare the queue
    queue = deque([])

    # Add the root node to the queue
    queue.append(root)

    # declare list to hold nodes of each level
    traversal = []

    # while the queue is not null
    while queue:

        # Iterate through the nodes at each level
        level = []
        for x in range(len(queue)):
            # for each node, remove it from the queue and add it to the level list and if
            # it has a child, add it to the queue at appropriate position
            node = queue.popleft()
            level.append(node.value)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
        traversal.append(level)
```

[1] A complete binary tree is a binary tree where every level except possibly the last level is full, meaning no additional nodes could fit on that level.

```
    # Reverse the traversal and print
    traversal.reverse()
    for level in traversal:
        for value in level:
            print(value, end=" ")
        print()
```

Test:

```
root = BinTreeNode(6)

#level 2
root.left = BinTreeNode(7)
root.right = BinTreeNode(8)

#level 3
root.left.left = BinTreeNode(9)
root.left.right = BinTreeNode(10)
root.right.left = BinTreeNode(11)

#level 4
root.left.left.left = BinTreeNode(12)
root.right.left.right = BinTreeNode(13)

#level 5
root.left.left.left.left = BinTreeNode(14)
root.left.left.left.right = BinTreeNode(15)
root.right.left.right.left = BinTreeNode(16)

rev_traversal(root)
```

Output:

```
14 15 16
12 13
9 10 11
7 8
6
```

Question 3:

Fill out your profile on Slack including a clear head shot. This will help the TAs and I, as well as you all, associate names with faces quicker. Paste a screen shot of your completed Slack profile below. [5 points]