Addis Coder Quiz 3

Welcome to Quiz 3! You will have 2 hours to complete this written quiz.

This quiz will cover material from weeks 1 through 3, including:

- Time Complexity
- Search Algorithms
- Sorting Algorithms
- Graph Traversal

`Name: _____`

`Lab: _____`

Question	Grade
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	

Problem 1: Algorithm Comparison

1.1 Which of the following time complexities uses the *fewest* number of operations as n increases?

a) $O(n)$

b) $O(n \log n)$

c) $O(n^2)$

d) $O(2^n)$

1.2 You are comparing two sorting algorithms: one with time complexity $O(n \log n)$ and another with $O(n^2)$. For large enough inputs, which uses fewer operations?

a) $O(n \log n)$

b) $O(n^2)$

c) They are the same

d) Depends on the values in the list

Problem 2: Time Complexity

What is the time compexity of the following functions? **Circle the correct choice** for each.

2.1

```
def count_pairs(n):
    for i in range(n):
        for j in range(n):
            print(i, j)
```

a) $O(n)$

b) $O(n \log n)$

c) $O(n^2)$

d) $O(2^n)$

2.2

```
def half_count(n):
    i = n
    while i > 1:
        i = i // 2
        print(i)
```

a) $O(\log n)$

b) $O(n)$

c) $O(n \log n)$

d) $O(n^2)$

2.3

```
def mystery(n):
    for i in range(n):
        for j in range(10):
            print(i, j)
```

a) $O(n)$

b) $O(n^2)$

c) $O(1)$

d) $O(n \log n)$

2.4

```
def run_something(n):
    for i in range(n):
        if i % 3 == 0:
            for j in range(n):
                print(i + j)
```

a) $O(n)$

b) $O(n \log n)$

c) $O(n^2)$

d) $O(2^n)$

Problem 3: More Big-O

3.1 Below are the number of operations performed by 4 different algorithms. Circle the option with the most operations for large n.

a) $50n + n \log n$

b) $n^2 + 100$

c) n^3

d) $\log(n) + 2n$

3.2 Write out the **Big-O time complexity** for all the algorithms in **Problem 3.1**.

a) ___

b) ___

c) ___

d) ___

Problem 4: True or False

State whether the following statements are True or False.

4.1 Binary search has time complexity O(n) in the worst case.

Answer: _____

4.2 Selection sort always makes the same number of comparisons regardless of input.

Answer: _____

4.3 In an undirected graph, every edge connects exactly two vertices.

Answer: _____

Problem 5: Fixing Recursive Selection Sort

You are given the following Python function that is supposed to sort a list in ascending order using recursive selection sort:

```
def RecSelectionSort(lst):
    if len(lst) <= 1:
        return lst
    min_index = 0
    for i in range(1, len(lst)):
        if lst[i] < lst[min_index]:
            min_index = i
    lst[i], lst[min_index] = lst[min_index], lst[i]
    return [lst[0]] + RecSelectionSort(lst[1:])
```

5.1 Write down the line of code that contains the bug.

Answer: _____

5.2 Write down a line of code to replace the line above and fix the function.

Answer: _____

5.3 On a list of length n, how many times will `RecSelectionSort` get called?

Answer: _____

Problem 6: Find 99

6.1 Write a function `find_99(L)` that takes as input a sorted list of integers `L` and uses **binary search** to find and return the position (index) of the number `99`. If `99` is not in the list, `find_99(L)` should return -1.

For example, if `L = [88, 53, 90, 99]`, then `find_99(L)` would return 3 and if `L = [-30, 8, 23, 43, 89]`, then `find_99(L)` would return -1.

6.2 What is the time complexity of your `find_99` algorithm, if n is the length of `L`?

`Answer: _____`

Problem 7: Improving Bubble Sort

Below is a working Bubble Sort implementation in Python:

```
def BubbleSort(lst):
    n = len(lst)
    for i in range(n - 1):
        for j in range(n - 1 - i):
            if lst[j] > lst[j + 1]:
                lst[j], lst[j + 1] = lst[j + 1], lst[j]
    return lst
```

Suppose we have a sorted list `sorted_lst` of length n. Then `BubbleSort(sorted_lst)` still uses $O(n^2)$ operations. Write a function `BetterBubbleSort` based on `BubbleSort` that uses $O(n)$ operations if the input list is already sorted.

Hint: How can you can detect that no swaps happened during a full pass through the list?

Problem 8: Mountain Climbing

You are a new mountain climber and want to count the number of peaks along the trail. You are given a sequence of heights along the trail. The trail has a "peak" at index `i` if `heights[i-1] < heights[i] > heights[i+1]`. (There are no peaks at the beginning or end of the trail.)

For example, `heights = [3, 5, 2, 9, 4]` has one peak at index `1` and one peak at index `3`.

Write a function `count_peaks(heights)` that takes in a list of heights `heights` and returns the number of peaks.

Problem 9: Sort by Last Digit

Write a function `sort_by_last_digit(nums)` that sorts a list of integers in ascending order based on the last digit of each number.

Example: `nums = [34, 21, 45, 12]`

Last digits:

- 34 → last digit is 4
- 21 → last digit is 1
- 45 → last digit is 5
- 12 → last digit is 2

If we sort by these last digits in ascending order:

- 1 → 21
- 2 → 12
- 4 → 34
- 5 → 45

so `sort_by_last_digit([34, 21, 45, 12]) = [21, 12, 34, 45]`

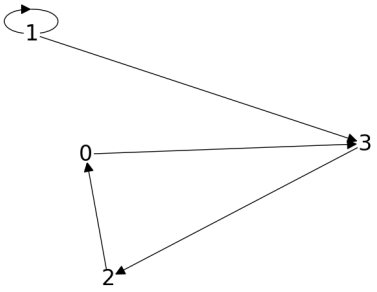
Problem 10: Graphs

10.1 Draw the following directed graph below: `{0: [3], 1: [1, 3], 2: [0, 3], 3: []}`

10.2 What is the **list of edges** of this graph?

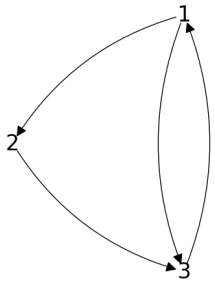
`Answer: _____`

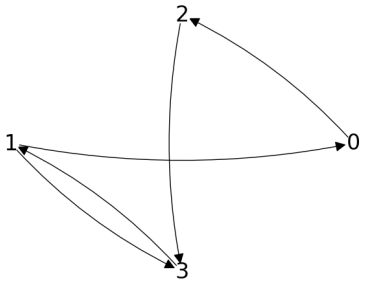
10.3 Is the graph above the same as the graph below? (Hint: two graphs are the same if their **list of edges** is the same.)

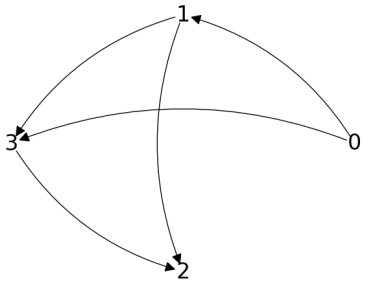


`Answer: _____`

10.4 For each graph below, write `True` if each node has the same in-degree and out-degree as in the graph above (in problem 10.3), and otherwise `False`.







a) _____

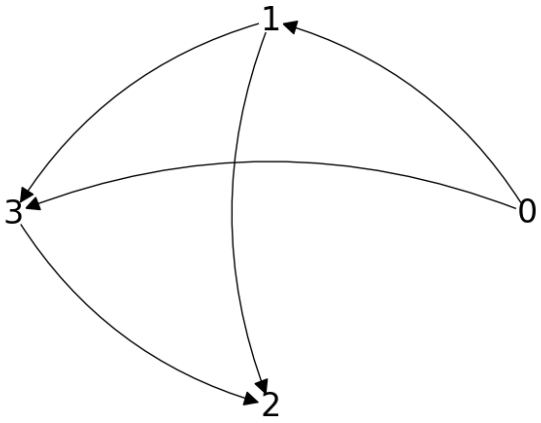
b) _____

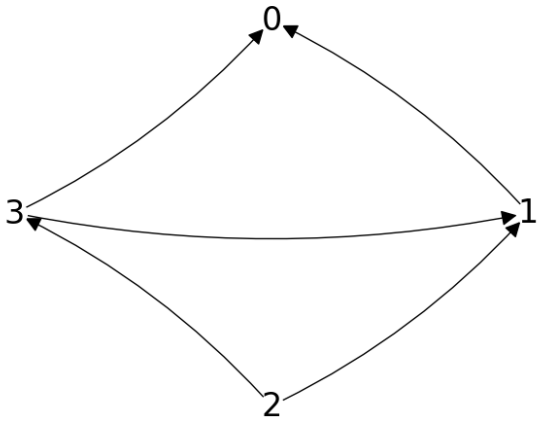
c) _____

Problem 11: Graph Traversal

11.1 Write a function `flip_edges(G)` that takes a directed graph `G` as an adjacency list and returns the graph with all the edges flipped.

An example of a graph and its flipped version is shown below.





11.2 Imagine you are an explorer looking for treasure. You are given a directed graph G. Each node u represents a position on a map. An edge from u to v means that you can travel from position u to position v. You are also given a set of "treasure nodes" T.

Write a function `can_reach_treasure(G, T)` that returns the list of nodes that can reach a treasure node.

For example, in the first graph above (problem 11.1), if node `3` is the only treasure node, then `can_reach_treasure(G, T)` should return `[0, 1]`.

Hint: you may use the `flip_edges` function you defined above.