Homework 4

1 Without Coding

Problem 1 (15 points)

Give the Big O run-time of the following algorithms. Explain/show your work.

Binary Search:

```python
def binary_search(arr, low, high, x):
    # Check base case
    if low > high:
        return None
    else:
        mid = (high + low) // 2
        element = arr[mid]

        if element == x:
            return mid
        elif element > x:
            return binary_search(arr, low, mid - 1, x)
        else:
            return binary_search(arr, mid + 1, high, x)
```

Selection Sort:

```python
def selection_sort(arr):
    for i in range(len(arr)):
        smallest_index = i
        smallest_value = arr[i]

        # Find smallest element
        for j in range(i, len(arr)):
            candidate = arr[j]
            if candidate < smallest_value:
                smallest_index = j
                smallest_value = candidate

        # Swap front with smallest value
        temp = arr[i]
        arr[i] = smallest_value
        arr[smallest_index] = temp

    return arr
```

Insertion Sort:

```
def insertion_sort(arr):

    for i in range(len(arr)):
        current_element = arr[i]
        # Find correct place in list
        j = 0
        while j < i and arr[j] <= current_element:
            j += 1

        # Move all larger elements over 1
        for x in range(i, j-1, -1):
            arr[x] = arr[x-1]

        # Insert current element
        arr[j] = current_element
```

Problem 2 (10 points)

Simplify the following Big O expressions. Show work for partial credit.

1. $O(2 * N + Log(N) + N^3 + 10)$
2. $O(1000)$
3. $O(10 + 20 * N + 10 * N * Log(N) + 10 * N^2)$
4. $O(N * Log(N) + 20 * N + N * Log(N)^2)$
5. $O(2 * N^2 * Log(N))$

Problem 3 (15 points)

What are the 4 tenants of Object-Oriented Programming with brief explanations (<2 sentences) of each?

2 Coding Problems

Problem 4 (20 points)

In class, we've studied Singly-Linked Lists which are made of nodes that point at subsequent nodes. One of the biggest annoyances with Linked Lists is the difficulty of going backwards through a list (such as getting the previous node or traversing the list backwards).

An intuitive solution to this inefficiency is the doubly-linked list, which adds pointers to previous nodes. Doubly-Linked Lists are not very different from Singly-Linked Lists, but are far more common because they are easier to use.

In this problem, you are to implement a Doubly-Linked List from scratch (you may use the Singly-Linked List code from class). You will have to create 2 classes (Node and DoublyLinkedList).

The Node class will have the following methods:

- __init__(value, prev, next)
- get_prev()
- get_next()
- get_value()
- set_prev(node)
- set_next(node)
- set_value(val)

The DoublyLinkedList class must have the following methods:

- __init__(): Note that there are no extra parameters here
- add_to_end(val): adds element as last
- add_to_front(val): adds element to first
- delete(val): deletes first occurrence of val
- reverse(): reverses the list
- compare(lst): check if regular Python list has the same values in the same order as the DLL
- find(val): return the index(as it would be in a list) of the first occurrence of val

Problem 5 (15 points)

Write a function that will sort a given list using merge sort. You must implement and use the merge sort algorithm (but may be recursive or iterative). The function will take a list as an input and return a sorted version of the list (you may assume it will be a list of integers).

The method signature must be merge_sort(lst).

Efficiency Challenge!

Problem 6 (10 points)

List to Dictionary
Write a function that has three parameters: a list of unsorted numbers with no duplicates, a start number, and an end number. This function should return a dictionary with all integers between the start and end number (inclusive) as the keys and their respective indices in the list as the value. If the integer is not in the list, the corresponding value would be None.

Example

```
unsorted list: [2,1,10,0,4,3]
two numbers: 3, 10
returned output: {3:5, 4:4, 5:None, 6:None, 7:None, 8:None, 9:None, 10:2}
```
[1]

The function signature must be: lst_to_dict(lst, start, end)

Problem 7 (15 points)

Write a function that, when given a list, it returns the indices of two numbers that add up to a specific number. You may assume that any input will have exactly one solution. You must return the indices in numerical order (smaller, larger)

Example:

```
Given lst = [1, 2, 3, 5, 9, 15], target = 7
Because lst[1] + lst[3] = 2 + 5 = 7,
return (1, 3)
```
[2]

The function signature must be: target_sum(lst, target)

Extra Credit (Up to 5 points)

The top 5 submissions that have the most efficient programs will get extra credit. The autograder will test and run the efficiency and create a leader-board. You are evaluated as the cumulative time for both functions

1st place on the leader-board gets +5 points
2nd place gets + 4
...
5th place gets + 1

Submission Instructions

Submit 2 files:
hw4.py - For the coding part
hw4.pdf - For the written part

Submit both parts on **Gradescope.**

At the top of hw4.py and hw4.pdf, include the following information:

- Your name
- The name of any classmates you discussed the assignment with, or the words "no collaborators"
- A list of sources you used (textbooks, wikipedia, research papers, etc.) to solve the assignment, or the words "no sources"
- Number of late days used for the current assignment, and the total number of late days used up thus far in the semester (including with the current assignment)

Grading Methodology

The **written part** of the assignment will be graded for correctness. In the case that your answer is incorrect, partial credit may be awarded based on the provided work.

The **code part** of the assignment will be graded by an automatic grading program. It will use the method signatures specified in the assignment and use multiple test cases.

Up to 5 points can be deducted for improper comments and styling. I expect:

- Descriptive variable names
- Comments for what functions do
- Comments for complex parts of code
- Proper naming conventions for any Variables, Functions, and Classes