# CSCI-SHU 210 Data Structures

## Assignment 2 Complexity

This assignment will be manually graded. Please submit on NYU Classes only.

**Question 1 (Count Primes):**

Implement function, count_primes(n), to count the number of prime numbers less than a non-negative number, n.

**Example:**

Input: count_primes(10)
Returns: 4
Explaination: There are four prime numbers less than 10: 2, 3, 5, 7

### Important:

- What is the runtime for your program? Mention your runtime in your .py file.

**Question 2 (Merge):**

Write a merge(l1,l2) function that takes two iterable objects and merges them alternately, once one runs out it continues from the other. Your algorithm should take O(n) time. For example, it should work as follows:

```
print([i for i in merge( range(5),range(100,105))])
print([i for i in merge( range(5),range(100,101))])
print([i for i in merge( range(1),range(100,105))])
```

should output:

```
[0, 100, 1, 101, 2, 102, 3, 103, 4, 104]
[0, 100, 1, 2, 3, 4]
[0, 100, 101, 102, 103, 104]
```

## <u>**Question 3 (Largest Ten):**</u>

Implement an efficient algorithm (Python code) for finding the **ten largest elements** in a list of size n.

Input: largest_ten([9,8,6,4,22,68,96,212,52,12,6,8,99,128])
Returns: [212, 128, 99, 96, 68, 52, 22, 12, 9, 8]   # Order doesn't matter.

### Important:

- You should avoid modifying original sequence.
- You can assume input list size is always greater than 10.
- What is the runtime for your program? <u>Mention your runtime in your .py file.</u>

## <u>**Question 4 (Min-Max):**</u>

Implement an algorithm (Python code) for finding both the minimum and maximum of n numbers using fewer than 3n/2 comparisons. Show that total number of comparisons is less than or equal to 3n/2.

(Hint1: First, construct a group of candidate minimums and a group of candidate maximums.).

(Hint2: For simplicity, list sizes are even only).

Input: min_max([6,2,5,8,1,-4,6,12,78,21,55,62,1,0])
Returns: (-4, 78) # Order matters. First term is min, second term is max.
# You can return a list, or a tuple to contain your results.

**<u>Question 5 (Three-Way Disjoint problem):</u>**

Given three sets of items, A, B, and C, they are **Three-Way Set Disjoint** if there is no element common to all three sets, i.e., there exists no x such that x is in A, B, and C. In the text book, two solutions of **Three-Way Set Disjointness** is described which run time complexity is O(n^3) and O(n^2).

Implement an algorithm (Python Code) that solves the **Three Way Set Disjoint** problem using O(nlogn) time. (Hint: Use O(nlogn) sorting algorithm).

```
l1 = [1,2,3,4,5]
l2 = [6,7,8,9,10,11,12]
l3 = [5,13,14,15,16]
l4 = [5,6,7,8,9,10,11]

Input1:three_way_disjoint(l1,l2,l3)
Returns: True

Input2:three_way_disjoint(l1,l4,l3))
Returns: False
```

**<u>Question 6 (Why is O(n^2) faster than O(nlogn) sometimes?):</u>**

Al and Bob are arguing about their algorithms. Al claims his O(nlogn)-time method is always faster than Bob's O(n^2)-time method. To settle the issue, they perform a set of experiments. To Al's dismay, they find that if n < 100, the O(n^2)-time algorithm runs faster, and only when n ≥ 100, O(n log n)-time one runs faster. Explain how this is possible.

### Important:

- You can submit a .txt file for this question.