# COMP 550.002: Fall 2023
# Assignment 3

Announced: October 5, 2023

Due Date: October 18, 2023

All problems are collaborative. You can form a group of at most four students to collaborate. You MUST mention the names of your collaborators, and cite any material you took help from (including discussions on canvas by other students) except the textbook and provided slides.

CLRS refers Cormen et al. textbook.

**Note:** Except Problems 2 and 3(b), solving these problems (particularly, Problems 1 and 3(a)) prepares you for Midterm 1.

## Problem 1 (15 + 15 = 30 Points)

[**This problem is subset of CLRS Problem 4.4-1**] For each of the following recurrences, sketch its recursion tree, and guess a good (as tight as possible) asymptotic **upper bound** on its solution. Then use the substitution method to verify your answer.

**(a)** $T(n) = 4T(n/2) + n$.

**(b)** $T(n) = 3T(n-1) + 1$.

(Hint: to create a guess from recursion tree, you can use the following steps: (i) sketch it (assume $n$ as a power of a constant if needed for simplification), (ii) count the number of levels to reach 1 in your tree, (iii) count the number of leaves, (iv) count cost of each level, (v) use your per-level costs to guess the solution (you don't need to do the detailed math involving summing up all the costs per level).

## Problem 2 (7 + 7 + 7 = 21 Points)

[**This problem is subset of CLRS Problem 4-1**] Give asymptotically **tight bounds** for $T(n)$ in each of the following recurrences. Justify your answers. (Hint: Apply the Master Theorem whenever possible to justify your answer.)

**(a)** $T(n) = 7T(n/2) + n^2 \lg n$.

**(b)** $T(n) = 16T(n/4) + n^2$.

**(c)** $T(n) = 2T(n/2) + n^3$.

# Problem 3 (7 + 7 = 14 Points)

[**This problem is based on CLRS Problem 7-4**] "StoogeSort" is a sorting algorithm named after the comedy team "The Three Stooges". If the input size, $n$, is 1 or 2, then the algorithm sorts the input immediately. Otherwise, it recursively sorts the first $2n/3$ elements, then the last $2n/3$ elements, and then the first $2n/3$ elements again. The details are shown below.

```
Algorithm StoogeSort(A, i, j):
  Input: An array, A, and two indices, i and j, such that 1 ≤ i ≤ j ≤ n
  Output: Subarray, A[i..j], sorted in nondecreasing order
  n ← j − i + 1        // The size of the subarray we are sorting
  if n = 2 then
      if A[i] > A[j] then
          Swap A[i] and A[j]
  else if n > 2 then
      m ← ⌊n/3⌋
      StoogeSort(A, i, j − m)        // Sort the first part
      StoogeSort(A, i + m, j)        // Sort the last part
      StoogeSort(A, i, j − m)        // Sort the first part again
  return A
```

**(a)** Write a recurrence relation corresponding to the running time $T(n)$ of the StoogeSort algorithm.
**(b)** Determine an asymptotic bound for $T(n)$ using the Master Theorem.

# Problem 4 (15 Points)

Write the pseudo-code of the binary search algorithm that searches a key $x$ in a sorted array $A[1:n]$ in $O(\lg n)$ time and returns the index where $x$ is found and NIL otherwise.

# Problem 5 (20 Points)

Let $A[1:n]$ and $B[1:n]$ be two arrays, each containing $n$ numbers already in sorted order. Give an $O(\lg n)$-time algorithm to find the "lower" median of all $2n$ elements in arrays A and B. Since the total number of elements is even, there are two medians. The lower median is the smallest among these two medians.

(Hint: $A[k]$ is the lower median if $n-k$ elements in $B$ are smaller/equal than $A[k]$ and the remaining $k$ elements in $B$ are larger/equal than $A[k]$. Use this idea to give an algorithm that performs similarly as binary search.)

A generalized version of this problem can be found here: `https://leetcode.com/problems/median-of-two-sorted-arrays/`. To validate your algorithm, you can implement this generalized version and submit it there. This is optional and will not be part of the grading. Even if you choose to implement this, for ease of grading, please submit the pseudocode as requested in the original question as your answer.

# Problem 6 (Extra Credit: 20 Points)

Implement the divide-and-conquer algorithm to determine the closest pair of points. For full credit, your program should be correct and run in $O(n \lg n)$ time. For a correct $O(n \lg^2 n)$ implementation, you will receive 90% of total points of this problem.

You can collaborate with your collaborators (the same group as for Problem 1–5) to discuss about implementation ideas and to take debugging help. However, you need to write your own code. At the top of you submitted code, you need to mention your collaborators name. More details will be provided in a separate document.