Problem Set 1

All parts are due on September 13, 2018 at 11PM. Please write your solutions in the LaTeX and Python templates provided. Aim for concise solutions; convoluted and obtuse descriptions might receive low marks, even when they are correct. Solutions should be submitted on the course website, and any code should be submitted for automated checking on `alg.mit.edu`.

Problem 1-1. [20 points] **Asymptotic behavior of functions**

For each of the following sets of five functions, order them so that if f_a appears before f_b in your sequence, then $f_a = O(f_b)$. If $f_a = O(f_b)$ and $f_b = O(f_a)$ (meaning f_a and f_b could appear in either order), indicate this by enclosing f_a and f_b in a set with curly braces. For example, if the functions are:

$$f_1 = n, \qquad f_2 = \sqrt{n}, \qquad f_3 = n + \sqrt{n},$$

the correct answers are $(f_2, \{f_1, f_3\})$ or $(f_2, \{f_3, f_1\})$.

Note: Recall that a^{b^c} means $a^{(b^c)}$, not $(a^b)^c$, and that $\log$ means $\log_2$ unless a different base is specified explicitly. Sterling's approximation may help for part **d)**.

a)	**b)**	**c)**	**d)**
$f_1 = 20n + 18$	$f_1 = n^{2 \log n}$	$f_1 = 2^{n^3}$	$f_1 = (2n)!$
$f_2 = 20n \cdot 18$	$f_2 = 2^{2^{\log n}}$	$f_2 = 2^{(n+1)^3}$	$f_2 = \binom{2n}{n}$
$f_3 = 20n^{18}$	$f_3 = 2^{(\log n)^2}$	$f_3 = n^{n^2}$	$f_3 = 2^{2n}$
$f_4 = \log_{20}(n^{18})$	$f_4 = n^{\log n}$	$f_4 = 2^{2^{n+1}}$	$f_4 = (2n)^n$
$f_5 = (\log_{18}(n))^{20}$	$f_5 = 2^{\log(\log n)}$	$f_5 = 3^{2^n}$	$f_5 = n^{2n}$

Problem 1-2. [30 points] **Recurrences**

(a) 3-Way Max Subarray Sum: In Lecture 2 we saw Divide-and-Conquer algorithm for the Max Subarray Sum problem based on the insight that, for an input array A of length n, the best (**consecutive**) subarray will either:

- lie within A's first half,
- lie within A's second half, or
- use at least one element from each half.

Recall that we have an $O(n)$ strategy for the last case: scan forwards to find the largest subarray beginning in the middle, and scan backward to find the largest subarray ending in the middle.

Derek Amayn thinks that splitting A into three pieces instead of two will speed things up! Derek's modified Divide-and-Conquer algorithm observes correctly that the best subarray will either:

- lie within the first two-thirds of A,
- lie within the last two-thirds of A, or
- use at least one element from each third.

You may assume n is a power of 3, for simplicity.

i. [5 points] **Write** a recurrence for the amount of work Derek's strategy will take.
ii. [5 points] **Solve** the recurrence using the Master Theorem. Is it $O(n \log n)$?

(b) Solving recurrences: Derive solutions to the following recurrences in two ways: draw a recursion tree **and** apply the master theorem. Assume $T(1) = O(1)$.

i. [5 points] $T(n) = 2\,T(\frac{n}{4}) + O(\log n)$
ii. [5 points] $T(n) = 17\,T(\frac{n}{3}) + \Theta(n!)$
iii. [5 points] $T(n) = 25\,T(\frac{n}{5}) + n^2 + 2n + \log n$
iv. [5 points] $T(n) = T(\sqrt{n}) + O(1)$ (Tree only)

Problem 1-3. [50 points] **Searching a Sorted 2D Array**

A two-dimensional $n \times m$ matrix A is **two-dimensionally sorted** if every row and column is sorted in increasing order rightward and downward: in other words,

$$A[y][x] \geq A[y][x-1] \qquad \text{and} \qquad A[y][x] \geq A[y-1][x]$$

whenever those indices are in range. For example, the matrix below is two-dimensionally sorted.

```
A = [[  1,  3,  7,  8, 16 ],
     [  2,  3,  8, 13, 17 ],
     [  4,  5, 14, 15, 21 ],
     [  8, 17, 17, 21, 23 ],
     [ 11, 17, 24, 26, 30 ]]
```

The SORTED2DSEARCH problem asks, given a two-dimensionally sorted array A and integer v, does the value v appear as an element of A?

(a) [5 points] Checking every element of the array against v solves this problem in $O(nm)$ time. Describe an algorithm that uses binary search to solve SORTED2DSEARCH in $O(n \log m)$ time. (In the following parts, you'll design and implement a linear time algorithm.)

(b) [5 points] Consider element $s = A[n-1][0]$ in the bottom left corner of A. If $v > s$, how does that limit possible locations of v? What if $v < s$? Show that if $v \neq s$, the number of elements from A that could be equal to v reduces by at least $\min(n, m)$.

(c) [15 points] Describe an $O(n+m)$ time algorithm to solve SORTED2DSEARCH. (For any algorithm you describe in this class, you should **argue that it is correct**, and **argue its running time**.)

(d) [25 points] Write a Python function `search_sorted_2D_array` that implements your algorithm. You can download a code template containing some test cases from the website. Submit your code online at `alg.mit.edu`.

```
1  def search_sorted_2D_array(A, v):
2      '''
3      Return tuple (x, y) such that A[y][x] == v, or None.
4      Input:  A | Array of equal length arrays of integers
5                |     representing the rows of a 2D array
6                |     where A[y][x] >= A[y - 1][x] and
7                |           A[y][x] >= A[y][x - 1]
8                |     for all (x, y) in range.
9              v | An integer to search for in A.
10     '''
11     ##################
12     # YOUR CODE HERE #
13     ##################
14     return None
```