Introduction to Algorithms: 6.006
Massachusetts Institute of Technology — September 12, 2017
Instructors: Jason Ku, Muriel Medard, and Silvio Micali — Problem Set 1

Problem Set 1

All parts are due on September 19, 2017 at 11:59PM. 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.csail.mit.edu .

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

Arrange each set of functions in a sequence such that: if f_i occurs before f_j then $f_i = O(f_j)$.

Set 0:	**Set 1:**	**Set 2:**	**Set 3:**
$f_1 = n^{6006}$	$f_1 = 8^n$	$f_1 = \log((\log n)^4)$	$f_1 = 5(n!)$
$f_2 = 6006n$	$f_2 = 2^{n^3}$	$f_2 = (\log n)^{\log(n^4)}$	$f_2 = \binom{n}{5}$
$f_3 = n\log(n^{6006})$	$f_3 = 2^{2^{n+1}}$	$f_3 = n^{4\log n}$	$f_3 = n^5$
$f_4 = 6006^n$	$f_4 = 2^{2n}$	$f_4 = \log(4^{n^4})$	$f_4 = \binom{n}{n/5}$
$f_5 = (\log n)^{6006}$	$f_5 = 3^{2^n}$	$f_5 = (\log\log n)^4$	$f_5 = 5^n$

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

(a) [10 points] **Setting up recurrences:**

1. The binary search algorithm finds an element in a sorted list of n comparable elements in $O(\log n)$ time. Write down a recurrence relation for binary search.
2. The insertion sort algorithm sorts a list of n comparable elements in $O(n^2)$, and is a common way to sort a small number of items, like a hand of cards. Write down the recurrence relation for insertion sort.

(b) [20 points] **Solving recurrences:**

Derive solutions to the following recurrences (i.e. show your work!). You may assume that constant sized problems can be solved in constant time.

1. $T(n) = 8T(\frac{n}{2}) + n^2$
2. $T(n) = 8T(\frac{n}{4}) + n^2$
3. $T(n) = 9T(\frac{n}{3}) + n^2$
4. $T(n) = T(n/3) + O(n)$ by expanding out the recurrence

Problem 1-3. [50 points] **Odd Corners**

Let a two-dimensional array of integers be *corner-odd* if the values in the four corners of the array sum to an odd number. We want you to write efficient code to find the smallest corner-odd subarray (i.e. contains the fewest elements) contained in an input array that is also corner-odd.

(a) [10 points] **Invariant:** Given an $n \times m$ corner-odd array that is strictly larger than 2×2, show there always exists a strict subset of the array that is also corner-odd.

(b) [5 points] **Minimal:** Show that every corner-odd array contains a 2×2 corner-odd subarray.

(c) [10 points] **Algorithm:** Describe an efficient, recursive algorithm that returns a smallest corner-odd subarray from a given an input corner-odd array.

(d) [5 points] **Analysis:** Write and solve a recurrence relation for your algorithm. Your runtime should be asymptotically faster than $O(nm)$.

(e) [20 points] **Implement:** Write a Python function `smallest_corner_odd` that implements your algorithm. You may find it useful to write a helper function to test if a subarray is corner-odd. You can download a code template containing some test cases from the website. Please submit your code to alg.csail.mit.edu . You will need to make an account.

```
##
def smallest_corner_odd(A, r = None):
    '''
    Input: 2D array A, subarray indices ((px, py), (qx, qy))
    Output: a smallest corner-odd subarray if the input
        subarray is corner-odd, else return None.
        Your output should be a tuple of tuples
        representing the indices of the upper left
        and lower right corners of the subarray.
    '''
    n, m = len(A[0]), len(A)
    if r is None:
        r = ((0, 0), (n - 1, m - 1))
    (px, py), (qx, qy) = r # (upper left), (lower right)
    ##################
    # YOUR CODE HERE #
    ##################
    return None
##
```