Introduction to Algorithms: 6.006
Massachusetts Institute of Technology
Instructors: Jason Ku, Muriel Medard, and Silvio Micali

September 12, 2017
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:**
$f_1 = n^{6006}$	$f_1 = 8^n$	$f_1 = \log((\log n)^4)$	$f_1 = 5(n!)$
$f_2 = 6006n$	$f_2 = 2^{n^3}$	$f_2 = (\log n)^{\log(n^4)}$	$f_2 = \binom{n}{5}$
$f_3 = n\log(n^{6006})$	$f_3 = 2^{2^{n+1}}$	$f_3 = n^{4\log n}$	$f_3 = n^5$
$f_4 = 6006^n$	$f_4 = 2^{2n}$	$f_4 = \log(4^{n^4})$	$f_4 = \binom{n}{n/5}$
$f_5 = (\log n)^{6006}$	$f_5 = 3^{2^n}$	$f_5 = (\log\log n)^4$	$f_5 = 5^n$

Solution:
Set 0: $(f_5, f_2, f_3, f_1, f_4)$. This order follows from knowing that $\log$ grows slower than n^a for all $0 < a$, and n^a grows slower than n^b for $0 < a < b$.

Set 1: $(f_4, f_1, f_2, f_5, f_3)$. This order follows by converting all the exponent bases to 2, specifically $f_1 = 2^{3n}$ and $f_5 = 2^{(\log_2 3)2^n}$ (constant factors matter inside the exponent).

Set 2: $(f_1, f_5, f_4, f_2, f_3)$. This order follows from elementary exponentiation and logarithm rules.

Set 3: $(\{f_2, f_3\}, f_4, f_5, f_1)$. This order follows from the definition of the binomial coefficient and Stirling's approximation. The tricky one is f_4 which is $O((5/4^{4/5})^n/\sqrt{n})$ by repeated application of Stirling and algebra. A weaker bound could also be used with Stirling applied once: $\binom{n}{k} \leq n^k/k! < (ne/k)^k$ yielding $O(\sqrt[5]{5e}^n)$.

Rubric:

- 5 points for correct sequence
- 3 points for sequence with a single pair swapped incorrectly

- 0 points otherwise

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.

   Solution: $T(n) = T(n/2) + O(1)$

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.

   Solution: $T(n) = T(n-1) + O(n)$

Rubric:
5 points for each correct recurrence
0 points otherwise

(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. $T(n) = 8T(\frac{n}{2}) + n^2$

   Solution: $T(n) = \Theta(n^3)$ by the Master Theorem because $n^2 = O(n^{\log_2 8-\epsilon})$ for e.g. $\epsilon = 1$.

2. $T(n) = 8T(\frac{n}{4}) + n^2$

   Solution: $T(n) = \Theta(n^2)$ by the Master Theorem because $n^2 = \Omega(n^{\log_4 8+\epsilon})$ for e.g. $\epsilon = 1/2$, and the regularity condition $af(n/b) \leq cf(n)$ holds, e.g. $8(n/4)^2 \leq cn^2$ for $c = 1/2$.

3. $T(n) = 9T(\frac{n}{3}) + n^2$

   Solution: $T(n) = \Theta(n^2 \log n)$ by the Master Theorem because $n^2 = \Theta(n^{\log_3 9})$.

4. $T(n) = T(n/3) + O(n)$ by expanding out the recurrence

   Solution: $T(n) = O(n)$ by seeing that $1 + 1/3 + 1/9 + \ldots$ is a geometric series that is bounded by a constant for all $n \in \mathbb{N}$, specifically, $\sum_{i=0}^{n} 3^{-i} < 3/2$.

Rubric:
3 points per correct recurrence.
2 points for showing work i.e. Master Theorem, recursion tree, expansion, etc.
-2 points off total if they use $O(.)$ instead of $\Theta(.)$ for recurrence solution.

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.

Solution: Being strictly larger than 2×2 implies some dimension of the array is greater than 2. Without loss of generality, assume width is the larger dimension. Consider the top and bottom elements x and y of a middle column of the array. Their sum is either even or odd. Because the array is corner-odd, the sum of either the left two corners or the right two corners of the array is odd, with the sum of the other two being even. Thus, x and y together with either the two left or two right corners form a subarray that is also corner-odd. □

Rubric: 10 points for a complete proof. Partial credit should be awarded for partial proof.

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

Solution: Proof by contradition. Suppose there exists a corner-odd array that contains no 2×2 corner-odd subarray, and let A be a smallest such array (with the fewest elements). Since A is corner-odd it cannot itself be 2×2, so it must have a 2×2 array as a strict subset. By part (a), there exists a strictly smaller subset of A that is also corner-odd, contradicting the minimality of A. □

Rubric: 5 points for a complete proof. Partial credit should be awarded for partial proof.

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

Solution: Divide the input corner-odd array in half along its larger dimension and identify the subarray half which is also corner-odd, which exists by (a). Repeat on the corner-odd subarray until the subarray is 2×2, and return it.

Rubric: 5 points for a correct algorithm. 5 points for an efficient algorithm ($O(\log(nm))$). Partial credit should be awarded for partial proof.

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

Solution: Let $N = nm$. Then the recurrence relation is $T(N) = T(N/2) + O(1)$. The solution to this recurence is $T(N) = O(\log N) = O(\log(nm))$.

Rubric: 2 points for a correct recurrence relation. 3 points for a correct run time order.

(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
##
```

Solution:

```
##
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 #
    ###################
    def corner_odd(A, r):
        (px, py), (qx, qy) = r
        corners = ((px, py), (px, qy), (qx, py), (qx, qy))
        return (sum([A[y][x] for (x, y) in corners]) % 2 != 0)
    if corner_odd(A, r):
        if (qx - px <= 1) and (qy - py <= 1):      # Base Case
            return r
        if (qy - py) < (qx - px):                  # Width larger than height
            mx = (px + qx) // 2
            rs = (((px, py), (mx, qy)), ((mx, py), (qx, qy)))
            for r_ in rs:                          # Check which side is corner-odd
                if corner_odd(A, r_):
                    break
```

```
        else:                                       # Width not larger than height
            my = (py + qy) // 2
            rs = (((px, py), (qx, my)), ((px, my), (qx, qy)))
            for r_ in rs:                           # Check which side is corner-odd
                if corner_odd(A, r_):
                    break
        return smallest_corner_odd(A, r_)           # Recurse
    return None                                     # Input was not corner odd...
##
```

Rubric: This problem is graded by ALG.