Introduction to Algorithms: 6.006
Massachusetts Institute of Technology — February 7, 2018
Instructors: Srini Devadas, Jason Ku, and Nir Shavit — Problem Set 1

Problem Set 1

All parts are due on February 15, 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 `py.mit.edu/6.006`.

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

For each of the following sets of five functions, write down a permutation of the numbers 1 to 5 such that, if a occurs before b in the sequence, then $f_a = O(f_b)$. If $f_a = O(f_b)$ and $f_b = O(f_a)$, then a and b may appear in either order.

a)	**b)**	**c)**	**d)**
$f_1 = 2018n$	$f_1 = (\log\log n)^5$	$f_1 = 9^n$	$f_1 = 4^n$
$f_2 = n\log(n^{2018})$	$f_2 = \log(5^{n^5})$	$f_2 = 3^{3n}$	$f_2 = 4(n!)$
$f_3 = 2018^n$	$f_3 = (\log n)^{\log(n^5)}$	$f_3 = 2^{3^n}$	$f_3 = n^4$
$f_4 = (\log n)^{2018}$	$f_4 = \log((\log n)^5)$	$f_4 = 3^{3^{n+1}}$	$f_4 = \binom{n}{4}$
$f_5 = n^{2018}$	$f_5 = n^{5\log n}$	$f_5 = 3^{n^2}$	$f_5 = \binom{n}{n/4}$

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

(a) [5 points] **Unbalanced Search:** Normal binary search finds an element in a sorted array of n elements in $O(\log n)$ time. In one application, we decide to preferentially search near the beginning of the array, so we modify the algorithm to repeatedly divide the search space into ratio 1:c instead of 1:1, for some constant $c > 2$. Write down and solve a recurrence relation to upper bound the running time of this unbalanced binary search. Does the modified binary search still run in $O(\log n)$ time?

(b) [25 points] **Solving recurrences:** Derive solutions to the following recurrences in two ways: draw a recursion tree **and** apply the master theorem. Assume $T(1) = O(1)$.

1. $T(n) = 3T(\frac{n}{2}) + O(n)$
2. $T(n) = 2T(\frac{n}{3}) + O(n)$
3. $T(n) = 4T(\frac{n}{2}) + \Theta(n^2)$
4. $T(n) = 3T(\frac{n}{3}) + \Theta(n)$
5. $T(n) = T(n-5) + O(\log n)$ (Tree only)

Problem 1-3. [50 points] **2D Peak Finding**

A **peak** in a two-dimensional $n \times n$ array of integers is an element of the array that is greater than or equal to each of its eight adjacent neighbors (some of which may not exist if the element is on the boundary). In this problem, you will design and implement an algorithm to find such a peak.

(a) [5 points] Prove that every nonempty two-dimensional array contains a peak. **Please review the 'Writing Proofs' handout on Stellar for guidelines on writing proofs.**

(b) [10 points] Consider a two-dimensional array A and rectangular sub-array $B \subseteq A$. Prove that if sub-array B contains an element that is greater than or equal to every element of A directly outside and adjacent to B, then B contains a peak.

We call such an element a **witness**, as it certifies the sub-array contains a peak.

(c) [10 points] Given an $n \times m$ array with $n \geq m$, divide it by splitting the longer dimension, n, in half. Show how to identify a witness in one of the halves to certify that it contains a peak in $O(m)$ time.

(d) [5 points] Our algorithm will divide the array in half by splitting its longer dimension, identify a witness in one of the halves, and then recursively search within the half containing the witness. Write and solve a recurrence relation based on this algorithm. How does the running time of this algorithm relate to input size, i.e., is the algorithm sub-linear, linear, or super-linear?

(e) [20 points] Write a Python function `find_peak_2D` that implements your algorithm. You can download a code template containing some test cases from the website. Submit your code online at `py.mit.edu/6.006`.

```python
def find_peak_2D(A, r = None, w = None):
    '''
    Find a peak in a two dimensional array.
    Input: 2D integer array A, subarray indices r, witness w
    '''
    if r is None:
        r = (0, 0, len(A[0]) - 1, len(A) - 1)
    px, py, qx, qy = r  # A[py][px] upper left, A[qy][qx] lower right
    if w is None:
        w = (0, 0)
    wx, wy = w          # A[wy][wx] witness
    ##################
    # YOUR CODE HERE #
    ##################
    return (0, 0)
```