Problem Set 2

All parts are due on September 20, 2019 at 6PM. 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 2-1. [10 points] **Famy k-sort**

Merge sort is a divide-and-conquer algorithm based on the idea that merging two sorted arrays into one larger sorted array can be done in linear time. Recall that merge sort divides unsorted input into two equally-sized subarrays, recursively sorts them, and then merges them together.

Famy Owler is a 6.006 student this semester and is curious how modifications to merge sort would affect its running time. Famy has found a way to generalize the merge step of merge sort to combine k size-m sorted arrays into one sorted array of size km in $\Theta(k^2m)$ time, for any positive integer k. **Famy k-sort** is a generalization of merge sort which splits an input array into k equal chunks, recursively sorts them, and then combines them using Famy's merge.

(a) [5 points] Write a recurrence relation for Famy k-sort in terms of k and n, the number of items to sort.

(b) [5 points] If $k = n^a + b$ for some constants $a \geq 0$ and $b \geq 1$, where n is the size of the input array, for what values of a and b will Famy k-sort run in $\Theta(n \log n)$ time?

Problem 2-2. [20 points] **Solving recurrences**

Derive solutions to the following recurrences in two ways: draw a recursion tree **and** apply the Master Theorem. Assume $T(1) \in \Theta(1)$.

(a) [5 points] $T(n) = 2\,T(\frac{n}{2}) + O(\sqrt{n})$

(b) [5 points] $T(n) = 8\,T(\frac{n}{4}) + O(n\sqrt{n})$

(c) [5 points] $T(n) = T(\frac{n}{3}) + T(\frac{n}{4}) + \Theta(n)$ assuming $T(a) < T(b)$ for all $a < b$

(d) [5 points] $T(n) = T(n-1) + 2\,T(n-2) + \Theta(n)$ (only solve via recursion tree)

Problem 2-3. [10 points] **Stone Searching**

Sanos is a supervillain on an intergalactic quest in search of an ancient and powerful artifact called the Thoul Stone. Unfortunately she has no idea what planet the stone is on. The universe is composed of an infinite number of planets, each identified by a unique positive integer. On each planet is an oracle who, after some persuasion, will tell Sanos whether or not the Thoul Stone is on a planet having a strictly higher planet identifier than their own. Interviewing every oracle in the universe would take forever, and Sanos wants to find the Thoul Stone quickly. Supposing the Thoul Stone resides on planet k, describe an algorithm to help Sanos find the Thoul Stone by interviewing at most $O(\log k)$ oracles.

Problem 2-4. [15 points] **Collage Collating**

Fodoby is a company that makes customized software tools for creative people. Their newest software, Ottoshop, helps users make collages by allowing them to overlay images on top of each other in a single document. Describe a database to keep track of the images in a given document which supports the following operations:

1. `make_document()`: construct an empty document containing no images
2. `import_image(x)`: add an image with unique integer ID `x` to the top of the document
3. `display()`: return an array of the document's image IDs in order from bottom to top
4. `move_below(x, y)`: move the image with ID `x` directly below the image with ID `y`

Operation (1) should run in worst-case $O(1)$ time, operations (2) and (3) should each run in worst-case $O(n)$ time, while operation (4) should run in worst-case $O(\log n)$ time, where n is the number of images contained in a document at the time of the operation.

Problem 2-5. [45 points] **Brick Blowing**

Porkland is a community of pigs who live in n houses lined up along one side of a long, straight street running east to west. Every house in Porkland was built from straw and bricks, but some houses were built with more bricks than others. One day, a wolf arrives in Porkland and all the pigs run inside their homes to hide. Unfortunately for the pigs, this wolf is extremely skilled at blowing down pig houses, aided by a strong wind already blowing from west to east. If the wolf blows in an easterly direction on a house containing b bricks, that house will fall down, along with every house east of it containing strictly fewer than b bricks. For every house in Porkland, the wolf wants to know its **damage**, i.e., the number of houses that would fall were he to blow on it in an easterly direction.

(a) [4 points] Suppose $n = 10$ and the number of bricks in each house in Porkland from west to east is `[34, 57, 70, 19, 48, 2, 94, 7, 63, 75]`. Compute for this instance the damage for every house in Porkland.

(b) [8 points] A house in Porkland is **special** if it either (1) has no easterly neighbor or (2) its adjacent neighbor to the east contains at least as many bricks as it does. Given an array containing the number of bricks in each house of Porkland, describe an $O(n)$-time algorithm to return the damage for every house in Porkland **when all but one house** in Porkland is special.

(c) [8 points] Given an array containing the number of bricks in each house of Porkland, describe an $O(n \log n)$-time algorithm to return the damage for every house in Porkland.

(d) [25 points] Write a Python function `get_damages` 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`.

```python
def get_damages(H):
    '''
    Input:  H | list of bricks per house from west to east
    Output: D | list of damage per house from west to east
    '''
    D = [1 for _ in H]
    ##################
    # YOUR CODE HERE #
    ##################
    return D
```