Problem Set 3

All parts are due on September 27, 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 3-1. [5 points] **Hash It Out**

(a) [2 points] Insert integer keys `A = [67, 13, 49, 24, 40, 33, 58]` in order into a hash table of size 9 using the hash function $h(k) = (11k + 4) \bmod 9$. Collisions should be resolved via chaining, where collisions are stored at the end of a chain. Draw a picture of the hash table after all keys have been inserted.

(b) [1 points] What is the maximum chain length of this hash table?

(c) [2 points] Suppose the hash function were instead $h(k) = ((11k + 4) \bmod c) \bmod 9$ for some positive integer c. Find the smallest value of c such that no collisions occur when inserting the keys from `A`.

Problem 3-2. [10 points] **Hash Sequence**

Hash tables are not only useful for implementing Set operations; they can be used to implement Sequences as well! (Recall the Set and Sequence interfaces were defined in Lecture and Recitation 02.) Given a hash table, describe how to use it as a black-box[1] (using only its Set interface operations) to implement the Sequence interface such that:

- `build(A)` runs in expected $O(n)$ time,
- `get_at` and `set_at` each run in expected $O(1)$ time,
- `insert_at` and `delete_at` each run in expected $O(n)$ time, and
- the four dynamic first/last operations each run in amortized expected $O(1)$ time.

Problem 3-3. [15 points] **Critter sort**

Ashley Getem collects and trains Pocket Critters to fight other Pocket Critters in battle. She has collected n Critters in total, and she keeps track of a variety of statistics for each Critter C_i. Describe **efficient**[2] algorithms to sort Ashley's Critters based on each of the following keys:

(a) [3 points] Species ID: an integer x_i between $-n$ and n (negative IDs are grumpy)

(b) [4 points] Name: a unique string s_i containing at most $10\lceil \lg n \rceil$ English letters

(c) [3 points] Number of fights fought: a positive integer f_i under n^2

(d) [5 points] Win fraction: ratio w_i/f_i where $w_i \leq f_i$ is the number of fights won

[1]By black-box, we mean you should not modify the inner workings of the data structure or algorithm.
[2]By "efficient", we mean that faster correct algorithms will receive more points than slower ones.

Problem 3-4. [20 points] **College Construction**

MIT has employed Gank Frehry to build a new wing of the Stata Center to house the new College of Computing. MIT wants the new building be as tall as possible, but Cambridge zoning laws limit all new buildings to be no higher than positive integer height h. As an innovative architect, Frehry has decided to build the new wing by stacking two giant aluminum cubes on top of each other, into which rooms will be carved. However, Frehry's supplier of aluminum cubes can only make cubes with a limited set of positive integer side lengths $S = \{s_0, \dots, s_{n-1}\}$. Help Frehry purchase cubes for the new building.

- **(a)** [10 points] Assuming the input S fits within $\Theta(n)$ machine words, describe an **expected** $O(n)$-time algorithm to determine whether there exist a pair of side lengths in S that exactly sum to h.
- **(b)** [10 points] Unfortunately for Frehry, there is no pair of side lengths in S that sum exactly to h. Assuming that $h = 600n^6$, describe a **worst-case** $O(n)$-time algorithm to return a pair of side lengths in S whose sum is closest to h without going over.

Problem 3-5. [50 points] **Copy Detection**

Plagiarism can be difficult to detect, especially when a copied text has been modified from its original form. It can be easier to detect plagiarism when a copied text appears without changes inside another text. Given a document string D and a query string Q, the **copy detection problem** asks whether Q appears as a contiguous substring of D. (For this problem, we assume that a string D is a sequence $(d_0, \dots, d_{|D|-1})$ of $|D|$ **ASCII characters**[3].) As an example, if $D =$ `'algorithm'`, the copy detection problem would answer True if $Q =$ `'rith'`, but answer False if $Q =$ `'log'`. A simple brute force algorithm solves the copy detection problem in $O(|D||Q|)$ time: there are $|D| - |Q| + 1$ contiguous substrings of D that have length $|Q|$:

$$\mathcal{D} = \{D_i = (d_i, \dots, d_{|Q|+i}) \mid i \in \{0, \dots, |D| - |Q|\}\},$$

so for each substring $D_i \in \mathcal{D}$, check whether $D_i = Q$, character-by-character in $O(|Q|)$ time. If we could determine whether $D_i = Q$ in $O(1)$ time instead of $O(|Q|)$ time, then the above algorithm would run much faster, in $O(|D|)$ time. Let's try to accomplish this goal using hashing!

- **(a)** [3 points] Given a string $S = (s_0, \dots, s_{|S|-1})$, let's think of it as the $|S|$-digit base-128 number $R(S) = \sum_{i=0}^{|S|-1} s_i \cdot 128^{(|S|-1-i)}$, where s_0 is the most significant digit. Note that $R(S_1) = R(S_2)$ if and only if $S_1 = S_2$. Describe an algorithm to compute $R(S)$ using $O(|S|)$ arithmetic operations, i.e., `(-, +, *, //, %)`. For parts (a), (b), and (c), assume that one arithmetic operation can operate on integers of arbitrary size.
- **(b)** [3 points] Given $R(D_i)$, i.e., the numerical representation of the i^{th} contiguous length-$|Q|$ substring of D for $i < |D| - |Q|$, and the value $f = 128^{|Q|}$, describe an algorithm to compute $R(D_{i+1})$ using $O(1)$ arithmetic operations.

[3]An ASCII character is represented by an integer between 0 and 127 inclusive. The built-in Python function `ord(c)` returns the ASCII integer corresponding to ASCII character `c`. `https://en.wikipedia.org/wiki/ASCII`

(c) [8 points] Describe an algorithm to solve the copy detection problem that uses at most $O(|D|)$ arithmetic operations.

The above algorithms pose a problem: if string $|S|$ is large, then $R(|S|)$ is roughly $\Omega(128^{|S|})$, likely much too large to fit in a single register on your CPU, so we won't be able to perform arithmetic operations on such numbers in $O(1)$ time. Instead of computing $R(|S|)$ directly, let's instead hash it down to a smaller range, specifically $R'(S) = R(S) \bmod p$, where p is a randomly chosen large prime number that fits into a machine word so it can be operated on in a CPU register in $O(1)$ time. If $Q = D_i$, then certainly $R'(Q) = R'(D_i)$, and we will be able to identify the match quickly! However, sometimes $R'(Q) = R'(D_i)$ when $Q \neq D_i$, and we will get a false match. Assume that p can be chosen sufficiently randomly such that false matches are unlikely to occur, i.e., the probability that $R'(Q) = R'(D_i)$ when $Q \neq D_i$ is $\leq \frac{1}{|Q|}$, for any $D_i \in \mathcal{D}$.

(d) [3 points] Given $R'(D_i)$ and the value $f' = (128^{|Q|}) \bmod p$, describe an $O(1)$-time algorithm to compute $R'(D_{i+1})$.

(e) [8 points] Describe an expected $O(|D|)$-time algorithm to solve the copy detection problem.

(f) [25 points] Write a Python function `detect_copy` that implements your algorithm. Instead of choosing p randomly, please use the Mersenne prime $p = 2^{31} - 1$. You can download a code template containing some test cases from the website. Submit your code online at `alg.mit.edu`.

```python
def detect_copy(D, Q):
    '''
    Input:  D | an ASCII string
    Output: Q | an ASCII string where |Q| < |D|
    '''
    p = 2**31 - 1
    ##################
    # YOUR CODE HERE #
    ##################
    return False
```