# Problem Set 4

**All parts are due on March 8, 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 4-1.** [8 points] **Hash Practice**

Insert integer keys $(2, 17, 6, 20, 5, 25, 1)$ in order into a hash table using hash function $h(k) = 13k + 2 \bmod 8$. 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. What would happen if you tried to insert the key $25$ into the hash table again?

**Problem 4-2.** [8 points] **Hashing Insecurities**

In this problem, we call a hash family $\mathcal{H}$ **insecure** if there exists a pair of distinct keys $(k_1, k_2)$ for which $h(k_1) = h(k_2)$ for all $h \in \mathcal{H}$.

**(a)** [2 points] Can a universal hash family ever be insecure? Explain.

Prove that the following families of hash functions (mapping keys from $\{0, \ldots, u-1\}$ to the range $\{0, \ldots, m-1\}$) are each insecure.

**(b)** [2 points] $\mathcal{H}_b = \{h(k) = (k + a) \bmod m \mid a \in \{1, \ldots, u\}\}$

**(c)** [2 points] $\mathcal{H}_c = \{h(k) = ak \bmod m \mid a \in \{1, \ldots, u\}\}$

**(d)** [2 points] $\mathcal{H}_d = \{h(k) = (h_s(\lfloor k/s \rfloor) + h_s(k - s\lfloor k/s \rfloor)) \bmod m \mid h_s \in \mathcal{H}_s\}$ where $s = \sqrt{u}$ is an integer, and $\mathcal{H}_s$ is a universal family of hash functions $h_s : \{0, \ldots, s-1\} \to \{0, \ldots, m-1\}$.

**Problem 4-3.** [8 points] **Changeable Priority Queue**

When we discuss shortest-path algorithms in a few weeks, we will need a priority queue which supports **changing** the key of a stored item. In order to identify items, each item `x` stores a unique integer **identifier** `x.id` in addition to its key `x.key` (keys might not be unique). Describe a data structure that supports the following operations, each in $O(\log n)$ time:

- `insert(x)`: insert item `x`
- `delete_min()`: remove an item with smallest key
- `change_key(id, k)`: change key of stored item `x` with identifier `id` to `k` (if it exists)

**Problem 4-4.** [16 points] **Evenly Uneven**

Loapo Ohyeah and her team are competing in a speed skating relay race having $n$ race segments, where each race segment has integer length. Loapo will skate exactly two of the race segments, and she needs to choose which ones to skate. In the interest of pacing, Loapo wants her pair of race segments to be **evenly uneven**: specifically, she wants one of her two race segments to be exactly four times the length of the other.

**(a)** [8 points] Given the list of race segments in the relay, describe an **expected** $O(n)$-time algorithm to find an evenly uneven pair, or return that no such pair exists.

**(b)** [8 points] If the length of every race segment is less than $n^4$, describe a **worst-case** $O(n)$-time algorithm to find an evenly uneven pair, or return that no such pair exists.

**Problem 4-5.** [10 points] **Doggo Sort**

Kindella Angel has $101 \cdot n$ dalmation pups that she loves very much and would never make coats out of them. However, while most of these puppies are oh so very good pups, Kindella admits that some of them are better than others, and better pups deserve more treats. Help Kindella sort her puppies, so she can decide who gets the most treats.

**(a)** [5 points] Kindella has decided that the best pups have the most spots. She knows that the total number of spots on all of the puppies is exactly $n^{101}$. Given a list of the spots on each pup, describe an efficient[1] algorithm to sort these pupperinos by their number of spots.

**(b)** [5 points] Although spots are very important, Kindella has found that many puppies have the same number of spots. She concludes that the best pups actually have the highest **spot density**: each pupper has an integer representing the surface area of its body, and a pup's spot density is its number of spots divided by its surface area. Given a list containing each pup's spot number and surface area, describe an efficient[1] algorithm for Kindella to sort her puppies by spot density, once and fur all.

[1] By "efficient", we mean that faster correct algorithms will receive more points than slower ones.

**Problem 4-6.** [50 points] **Po-$k$-er Hands**

Meff Ja is a card shark who enjoys playing card games. He has found an unusual deck of cards, where each of the $n$ cards in the deck is marked with a lowercase letter from the 26-character English alphabet. We represent a deck of cards as a sequence of letters, where the first letter corresponds to the top of the deck. Meff wants to play a game of Po-$k$-er with you. To begin the game, he deals you a Po-$k$-er hand of $k$ cards in the following way:

1. The deck $D$ starts in a pile face down in a known order.
2. Meff **cuts** the deck uniformly at random at some location $i \in \{0, \ldots, n-1\}$, i.e., move the top $i$ cards in order to the bottom of the deck.
3. Meff then deals you the top $k$ cards from the top of the cut deck.
4. You **sort** your $k$ cards alphabetically, resulting in your Po-$k$-er **hand**.

Let $h(D, i, k)$ be the Po-$k$-er hand resulting from cutting a deck $D$ at location $i$. Then cutting deck $D =$ `'abcdbc'` at location 2 would result in the deck `'cdbcab'`, which would then yield the Po-4-er hand $h(D, 2, 4) =$ `'bccd'`. From a given starting deck, many hands are possible depending on where the deck is cut. Meff wants to know the **most likely** Po-$k$-er hand for a given deck. Given that the most likely Po-$k$-er hand is not necessarily unique, Meff always prefers the lexicographically smallest hand.

**(a)** [5 points] Given the deck $D =$ `'koperrepokpokeropkre'`, list the set of all possible Po-3-er hands $\{h(D, i, 3) \mid i \in \{0, \ldots, 19\}\}$, along with their probabilities of appearing given a cut location chosen uniformly at random.

**(b)** [10 points] Describe a data structure that can be built in $O(n)$ time from a deck $D$ of $n$ cards and integer $k$, after which it can support `same(i, j)`: a constant-time operation which returns True if $h(D, i, k) = h(D, j, k)$ and False otherwise.

**(c)** [10 points] Given a deck of $n$ cards, describe an $O(n)$-time algorithm to find the most likely Po-$k$-er hand, breaking ties lexicographically. State whether your algorithm's running time is worst-case, amortized, and/or expected.

**(d)** [25 points] Write a Python function `most_likely_hand(D, k)` 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`.

```
def most_likely_hand(D, k):
    '''
    Input:  D | string of length n representing the deck
            k | integer representing hand size
    Output: H | string of length k representing the most likely hand
    '''
    ##################
    # YOUR CODE HERE #
    ##################
    return ""
```