Note: please finish the following tasks in Python, unless otherwise specified.

Problem 1

There is a deck of 54 cards, all facing down. Given a series of integers $x_0, x_1, x_2, \ldots, x_{n-1}$, we are going to conduct the following random experiment:

For each x_i:
1) pick the first x_i cards and turn them over;
2) completely shuffle all the cards by random.

Let U denote the number of cards that are facing up by the end of the experiment. Due to the randomness of the experiment, we don't always end up with the same U. Can you implement a function that calculates the *expected value* of U? (Don't use random library; simulation is *not* required.)

Signature:

```
def facing_up(x)
```

Input:
A list of integers x, each integer x_i between 0 and 54, inclusive.

Return:
A single float of the expected number of face-up cards.

Example

Input: x = [1]
Return: 1.0
Explanation: in the first round, we turn the first card over, and then shuffle the deck by random. Because only the turned card is facing up eventually, $E[U] = 1$.

Input: x = [27, 1]
Return: 27.0
Explanation: in the first round, we turn the first 27 cards over, and then shuffle the deck by random. Each card then has a 50% chance of facing up. In the second round, we turn the first card over. This operation does not affect the expected number of up-facing

cards, because we have equal chance turning over an up-facing or down-facing card. So $E[U] = 27$.

Problem 2

It is often useful to apply a weighted sum over a series of data, in order to create a single scalar feature. For example, if we have a series

$$x = [1, 5, 2, 4],$$

define the weight of the ith element to be $w_i = c^i$ for some scalar c, so that the weighted sum S can be presented as

$$S = \sum_{i=0}^{n-1} w_i \cdot x_i = \sum_{i=0}^{n-1} c^i \cdot x_i$$

Now we want to expand the weighted sum to all possible rotations of x, so for the aforementioned case, we would also consider $[5, 2, 4, 1], [2, 4, 1, 5]$ and $[4, 1, 5, 2]$ in addition to x itself. Can you help implement a function to find the lower and upper bounds of S for all rotations of a series x?

Signature:

```
def bounds_of_rotations(x, c)
```

Input:

1) x: a list of integers
2) c: a float that satisfies $0.2 \leq c \leq 0.8$.

Return:

A tuple with its first element being the lower bound, and the second the upper bound.

Example

Input: x = [1, 1, 1, 1, 1], c = 0.5
Return: (1.9375, 1.9375)

Input: x = [2, 5, 6, 8], c = 0.8
Return: (13.936, 16.24)

Bonus tasks:

1) Can you implement the function with $O(n)$ time complexity? If yes, give your implementation. If not, can you describe the time complexity of your current implementation?
2) If we define $w_i = i$, can you still implement the function with $O(n)$ time complexity?

Problem 3:

We have following database

application

```
|-- id: string (nullable = true)
|-- user_id: string (nullable = true)
|-- status: string (nullable = true)
|-- create_timestamp: long (nullable = true)
```

user

```
|-- id: string (nullable = true)
|-- mobile_number: string (nullable = true)
```

In table **application**, id is the unique id of each application, user_id is the unique id in table **user**. status is the status of the application, it can be 'Approved', 'Rejected', 'Pending'. create_timestamp is the unix timestamp of the application's create time. A user_id may make multiple applications.

In table **user**, id is the unique id of each user. Note that different user_id may have the same mobile_number in this table.

Now, we want to extract the following content for each application in **application**:

1) Find the last approved application with the same mobile_number as the current application and create time is before current application
2) The approval ratio of the historical applications with the same mobile_number as current application (historical applications means the applications whose create

time is before current application). If the mobile_number has no historical applications, set the value to be null.

The resulting schema should look like this:

```
|-- id: string (nullable = true)
|-- user_id: string (nullable = true)
|-- status: string (nullable = true)
|-- create_timestamp: long (nullable = true)
|-- last_approved_application: struct (nullable = true)
|    |-- id: string (nullable = true)
|    |-- user_id: string (nullable = true)
|    |-- create_timestamp: long (nullable = true)
|-- historical_approval_ratio: double (nullable = true)
```

The **application** and **user** tables are very big, so we cannot collect all data first and then do it in memory.

Please give a solution to implement this through SQL (MySQL, spark SQL, Hive SQL etc., select the one you are familiar with). Please also specify the SQL flavour you are using in your answer.