# Stat 766 Homework 1

1. Given two strings, needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

   Constraint:

   haystack and needle consist of only ASCii characters, digits, and punctuations.

   $1 \leq$ haystack.length, needle.length $\leq 10000$

   For example,

   ```
   input:
   haystack = 'copydestination';  needle = 'des'

   output from strStr:
   4

   needle = 'esd'
   output:
   -1
   ```

   Complete the definition of the function **strStr** in the Python class **Solution**.

   ```
   class Solution:

       def strStr(self, haystack: str, needle: str) -> int:
   ```

2. You are given an array of two sorted lists, each list in ascending order. Merge the two lists into one sorted list (in increasing order) and return it.

   For example:

   ```
   Input: lists = [[1,4,5],[1,3,4,6]]

   Output: [1,1,3,4,4,5,6]
   ```

   Constraints:

lists.length =2

$0 \leq$ lists[i].length $\leq 500$

-104 $\leq$ lists[i][j] $\leq$ 104

lists[i] is sorted in ascending order.

The sum of lists[i].length will not exceed 700.

A simple yet inefficient way to solve this problem is to combine the two lists into one list and then use the .sort or sorted function on the combined list. This solution is simple but computationally slow because it ignored the fact that each of the two lists were already in increasing order. Comparing each number with other numbers to find the position will lead to total of $O(n^2)$ comparisons, where $n$ is the length of the combined list. (We say the computational complexity is of order $O(n^2)$.) Implement this simple solution by completing the definition of the function **slowMerge** in the Python class **Solution**.

3. A more efficient way to solve the problem in Question 2 is to use the Merge-sort algorithm described below. This algorithm's computation complexity is $O(n)$.

   {Comment: Begin with two presorted sets of numbers $X$ and $Y$, of length $m_x$ and $m_y$ respectively}

   **Merge-sort algorithm**:

```
Initialize $Z$ as an empty list. The final $Z$ will have length $n = m_x + m_y$
Initialize $j = 0$, $k = 0$, $\ell = 0$
{Comment: Just to avoid referring to something that does not exist.}
Initialize $X_{m+1} = \infty$, $Y_{r+1} = \infty$
while $\ell \leq n = m_x + m_y$ do
  {Comment: Since $X$ and $Y$ are already sorted, the smallest entry must be towards the
  beginning of one of the arrays. Start there and move forward.}
  if $X_j < Y_k$ then
    Set $Z_\ell = X_j$. Set $j = j + 1$
  else
    Set $Z_\ell = Y_k$. Set $k = k + 1$
  end if
  Set $\ell = \ell + 1$
end while
return $Z$
```

Implement this algorithm by completing the definition of the function **MergeSort** in the Python class **Solution**.

4. Two linked tables are stored in csv files.

```
Table: Customers

+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| name        | varchar |
+-------------+---------+
```

id is the primary key (column with unique values) for this table. Each row of this table indicates the ID and name of a customer.

```
Table: Orders

+-------------+------+
| Column Name | Type |
+-------------+------+
| id          | int  |
| customerId  | int  |
+-------------+------+
```

id is the primary key (column with unique values) for this table. customerId is a foreign key (reference columns) of the ID from the Customers table. Each row of this table indicates the ID of an order and the ID of the customer who ordered it.

Complete the function **find_customers** so that the function returns all customers who never order.

Return the result table as a data frame in any order.

The result format is in the following example. Pay attention to the column name of the result table.

Note: While you are testing your code, you can read in a csv table to Python with pd.read_csv command. However, don't use pd.read_csv in your function. Your function will be evaluated on different data sets.

Example input and output:

```
Input:
Customers table:   Orders table:         Output:
+----+-------+     +----+------------+   +-----------+
| id | name  |     | id | customerId |   | Customers |
+----+-------+     +----+------------+   +-----------+
| 1  | Joe   |     | 1  | 3          |   | Henry     |
| 2  | Henry |     | 2  | 1          |   | Max       |
| 3  | Sam   |     +----+------------+   +-----------+
| 4  | Max   |
+----+-------+
```

5. Find the name of managers with at least five direct reports of employee entries.

   Table: Employee

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| name        | varchar |
| department  | varchar |
| managerId   | int     |
+-------------+---------+
```

   id is the primary key (column with unique values) for this table. Each row of this table indicates the name of an employee, their department, and the id of their manager. If managerId is **missing**, then the employee does not have a manager. No employee will be the manager of themselves.

   Complete the function **find_managers** in the class **solution** so that the function can be used to find managers with at least five direct reports.

Return the result table in any order.

The result format is in the following example.

Example input and output:

```
Input:
Employee table:
+-----+-------+------------+-----------+
| id  | name  | department | managerId |
+-----+-------+------------+-----------+
| 101 | John  | A          |           |
| 102 | Dan   | A          | 101       |
| 103 | James | A          | 101       |
| 104 | Amy   | A          | 101       |
| 105 | Anne  | A          | 101       |
| 106 | Ron   | B          | 101       |
+-----+-------+------------+-----------+
Output:
+------+
| name |
+------+
| John |
+------+
```

## Submission Instruction

Fill in the definition of functions in the file **student_answer_template.py**. Then submit the .py file on gradescope. You will be be able to see your score in a few minutes if your coding is successful. If there are errors when you submit it, correct errors and resubmit.