Problem Set 4

All parts are due on March 8, 2018 at 11PM. 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 `py.mit.edu/6.006`.

Problem 4-1. [30 points] **Consulting**

(a) [10 points] Some students in 6.006 want to find out if classmates in their other classes are also taking 6.006 so they can find study partners. Each MIT class is identified by a sequence of no more than 8 numbers and/or characters, for example 21W.789 or CMS.355. Assume that students have a credit limit allowing them to take no more than 8 classes per semester. Given a list of n 6.006 students and their semester schedules, describe a worst-case linear time algorithm to generate a list of MIT classes containing more than m 6.006 students.

(b) [20 points] Beverly Jezos is starting a new grocery delivery service with a **constantly-changing** product selection, and wants to create a mobile app for customers to search and purchase products from her inventory. Products have the following four properties: price (up to \$100.00), name, category (one of Produce, Meat, Dairy, Bakery, Frozen, Drinks, Snacks, or Misc), and stock (the number remaining before sold out). The mobile app should store a local copy of the current inventory in a database, dynamically displayed to a user as a table, with one product per row and one property per column. The columns may be reordered by the user. When a user changes the order of columns, the table should dynamically update to display products in sorted order according to their properties, with decreasing priority according to their new column order from left to right. Help Bev implement a system to display these dynamically sorted tables efficiently.

Problem 4-2. [20 points] **Sorting Sorts**

For each of the following scenarios, choose a sorting algorithm that best applies, and justify your choice. **Don't forget this! Your justification will be worth points more than your choice.** You may pick any sort we have covered in this class: insertion sort, selection sort, merge sort, heap sort, AVL sort, counting sort, or radix sort. Each sort may be used more than once. If you find that multiple sorts could be appropriate for a scenario, identify their pros and cons, and choose the one that best suits the application. State and justify any assumptions you make. "Best" should be evaluated, primarily by asymptotic running time, but also in terms of stability, space, and implementation.

(a) [5 points] Your TA just discovered a new technique for wide learning called "Vector Comprehensions" and is going to present their work in a far-away conference. Unfortunately, your TA waited too long to book flights, which have become expensive. There is a very large number of flights to choose from. Help your TA sort the flights by price.

(b) [5 points] Despite your best efforts, your TA's flights are still very expensive. To help pay for them, your TA takes on a high-paying minimum wage job as a junior quesadilla engineer at the local burrito place. At various times through out the day, the manager wants to know statistics about the orders that have been fulfilled or canceled during the day (e.g. mean, median, minimum, maximum). Help your TA maintain a sorted list of transactions, in order to respond to the manager quickly.

(c) [5 points] Your TA rushes to the airport with a stack of problem sets to grade, ordered by student's last name. While going through security, your TA trips, spilling the problem sets onto the ground. Picking them up, your TA notices that the assignments are still mostly sorted, though unfortunately a couple of assignments (far fewer than the total number of problem sets) are out of place. Help your poor TA get the problem sets back in order.

(d) [5 points] While at the conference, your TA wants to attend interesting talks. Unfortunately, your TA also has to grade problem sets and do other work, which limits the number of talks that can be attended. Some talks are preferred over others, and given two talks your TA can tell you which of the two would be preferred, though it would be difficult to put a numerical value on it. Help your TA choose the best set of talks according to their preferences.

Problem 4-3. [50 points] **Anagram Sort**

You are given a long list of words in some random order. Your task is to sort the words such that sets of anagrams are next to each other in the sorted list, minimizing worst-case complexity. Your sort should order words according to their histogram, where the counts of letters earlier in the alphabet are considered more significant than the counts of letters later in the alphabet, and each count is sorted in ascending order. Words that are anagrams of each other should appear in the order in which they were provided in the original list. For example, given the list $S = [\text{eat, dog, ate, good, tea}]$, we want to output $[\text{dog, good, eat, ate, tea}]$. The words with the letter a in them end up at the end of the list, since the count of a is the most significant.

To accomplish this task, we propose the following procedure:

1. Create a histogram of the letters present in each word, i.e. populate an array A of size σ, where $\sigma = 26$ is the size of the alphabet, and $A[i]$ is the frequency of letter i.
2. Once each word's histogram has been computed, apply radix sort (with a counting sort subroutine) on the histograms to sort the anagrams as desired.

(a) [10 points] Provide a histogram for each word in $S = [\text{ate, dog, eat, good, tea}]$ according to the first step of the algorithm described above.

(b) [10 points] Provide the orderings of words S after applying each step of radix sort, to the histograms generated in part (a). You may skip any radix step sorting the letter count of any letter that is not contained in any word of S.

(c) [30 points] Write the Python function `sort_anagrams(words)` that implements this algorithm. You may assume that all words in the input list contain only lower-case letters. You can download a code template containing some test cases from the website. Submit your code online at `py.mit.edu/6.006`, and **also in your PDF submission**. You should use either the verbatim or lstlisting LaTeX environments to include your code.

Note: For this problem, we expect you to implement a version of radix sort in your implementation. Any implementation that utilizes the built-in sort or sorted methods rather than implementing the radix sort will receive little or no credit. Submit your code in your PDF submission, in addition to the online code checker.

```python
def sort_anagrams(words):
    '''
    Sort anagrams to appear adjacent to each other in a returned list.
    Input:  list of strings which contain only lower-case letters
    Output: sorted list, with anagrams appearing consecutively
    '''
```