Assignment 1

(Due: Oct 24, 2019) 11:59PM

Total points: 100

Introduction

The goal of this assignment is to refamiliarize your Python skills and learn to use it to write some sorting algorithms and in the process develop your problem-solving skills. For each of the problems you will formulate or lookup an algorithm, write the corresponding source code, and thoroughly test it before handing it in. Be sure to carefully read each of the problems, as every detail will play a role in the solution.

This assignment is to be completed **individually**. You may work with other students on understanding the problems and discussing the algorithms, however, you are not allowed to share nor acquire source code from other students in the class, current or previous. You must start the assignment early in order to get proper help from the TAs and the instructor. You can also post general questions or ask for clarifications on Piazza, however, **do not post your own source code on Piazza**.

Please read the entire assignment carefully, before starting

There is an extra credit question at the end. It is of course optional.

Methodology

I expect you to create a source file `assignment1.py` that will contain the solutions to the questions below. For each question you will create a method using the provided defintions. You may add any number of helper methods.

PLEASE USE the method definitions exactly as given for each questions. If you do not, then the autograder on Gradescope will not be able to recognize your answer and will give you a zero.

Additionally, you may want to create another file, for example *test-assign.py*, that will contain code for testing your functions. You are encouraged to test each of your algorithms with various edge and corner cases. *You will NOT hand in this file.*

Abstain from looking up solutions on the web or textbooks. I am sure you will find partial or even complete

solutions online, but following this approach is detrimental to your development and success in this class.

All you need to submit for this work is ONE `assignment1.py` file on Gradescope.

Before you proceed, please read each question carefully, several times if needed, work out a plan of action on paper, and then write your code.

Question 1 (25 points)

Write a Python function (method) that displays a one-year calendar for ANY year. The program should prompt the user for a year (past, present or future). Then the program should calculate, the weekday for Jan 1 for that year. It turns out there is an equation for calculating this value, which is given as:

```
day = R(1+5R(Y-1,4)+4(Y-1,100)+6R(Y-1,400),7)
```

Here *R(x,y)* is remainder after the division of *x* by *y*, and *Y* is the year. This equation, credited to Carl Friedrich Gauss, produces a value between 0 and 6, where 0 is Sunday, 1 is Monday,….6 is Saturday. Once you know which day of the week the January 1 was for the year, you should print the calendar for the entire year.

The calendar should be formatted as shown in the sample execution below. Note that numbers the days must be right-justified under the names of the days and that two spaces separate the names of the days from each other.

Please take care to make sure if the year your entered is a leap year or not. snippet of the execution of the code is given below.

```
Enter a year: 2004 <<user input>>

<<SAMPLE OUTPUT>>
In the Year 2004, January 1 was/will be on a Thursday.

It is a leap year.


     ***    CALENDAR FOR YEAR 2004    ***

January

Sun  Mon  Tue  Wed  Thu  Fri  Sat
                    1    2    3
  4    5    6    7    8    9   10
 11   12   13   14   15   16   17
 18   19   20   21   22   23   24
 25   26   27   28   29   30   31

February

Sun  Mon  Tue  Wed  Thu  Fri  Sat
  1    2    3    4    5    6    7
  8    9   10   11   12   13   14
                 .
                 .
                 .
(output continues for all 12 months)
```

Use loops to make sure you print the calendar for the whole year automatically.

We will consider only years from the *Current Era (C.E.)* for this question.

Grading

Your method should return a list as it's output. The first value of the list should be the first day of the year, the second the second day, and so on until you have all 365 (366 for a leap year) days of the year in one list.

> For example, if the year starts on say a Tuesday, then the first value of the list should be 2. Thus the full list would be `[2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6,..... n]` with *0<=n<=6* being the last day of the year. Note that the values in the above list lie between 0 and 6.

Please use this exact function prototype for this question as your work will be auto-graded in Gradescope.

```
def calendar(year: int) -> list
```

Question 2 (50 points)

You have seen BubbleSort in the class. You will remember that in BubbleSort, in every iteration, we find the maximum element in the FIRST *n-i* elements of the list and push it to the end of the list *(n-1-i)*th position in the list . Here, *i* is the current iteration of the algorithm starting at element 0, and *n* is the number of elements in the list. Eventually, this leads to the list being fully sorted.

PART 1: (25 points)

Please implement an alternative version called `Alt-BubbleSort` that does the opposite? That is, in every iteration, it finds smallest element in the LAST *n-i* elements of the list pushes it to the ith position in the list. Again, here, *i* is the current iteration of the algorithm starting at element 0, and *n* is the number of elements in the list.

Grading

After every round of the algorithm, your method should store the current state of the list in a *lists-of-lists*, starting with the orignal list (i.e., the input). Once the algorithm terminates, you should store the final sorted list in the list-of-lists and return the entire list-of-lists.

> For example, if you are sorting a list `[3,1,2]` then your algorithm may return the following list-of-lists `[[3,1,2][1,3,2][1,2,3]]`

Please use these exact function prototype for this question as your work will be auto-graded in Gradescope.

```
def alt_bubblesort(A: list, size: int) -> list
```

PART 2: (25 points)

Now implement yet another version called `Switch-BubbleSort` combines the tradition BubbleSort and Alt-BubbleSort. That is, it alternates between finding the largest element and pushing it to the end of the list is one iteration and then finding the smallest element and pushing it to the beginning of the list is the next iteration and goes back and forth until the entire list is sorted, which is then returned.

Grading

Again, after every round of the algorithm, your method should store the current state of the list in a *lists-of-lists*, starting with the orignal list and ending with the sorted list

Please use these exact function prototype for this question as your work will be auto-graded in Gradescope.

```
def switch_bubblesort(A: list, size: int) -> list
```

NOTE: The following link might be useful in understanding how to deal with list of lists: https://snakify.org/en/lessons/two*dimensional*lists_arrays/

Question 3 (25 points)

Implement `BucketSort`. This algorithm is described in *Section 8.4* of *Cormen et. Al. (3rd Edition)* textbook. Please look at the algorithm given and implement it. Please make sure to test your implementation with a list made entirely of fractional numbers (0.8, 0.65 etc). Your method should return a sorted list at the end.

Grading

To ensure that you're using bucket sort, we require that you include these steps within your algorithm:

- Write to a file called bucket1.txt your buckets after you filled them, but before you sort them
- Write to a file called bucket2.txt your buckets after each bucket has been sorted
- Return your sorted list at the end

Please make sure that for each of the print statements above, you separate each bucket with a newline. In regards to printing each element within the bucket, separate the elements with a space inbetween (ie. 0.55 0.67 0.34).

For example code of how to write to a file in python, please look at the bottom of this page.

Please use this exact function prototype for this question as your work will be auto-graded in Gradescope.

```
def bucketsort(A: list, size: int) -> list
```

Question 4 (30 points) -- EXTRA CREDIT

There is an algorithm called ColumnSort that works on a list of *N* elements arranged in a rectangle. The array has *R* rows and *S* columns such that *RxS = N*, subject to three restrictions:

- *R* must be even
- *S* must be divisor of R
- $R >= 2S^2$ [The example below violates this property, but still works. The point is for the rectangle to be tall and narrow]

When ColumnSort completes, the array is sorted in a column-major order. This means reading down the columns, from left to right, the elements increase monotonically (i.e., strictly increase).

ColumnSort operates in eight steps, regardless of the value of N. The odd steps are all the same: sort the columns individually. Each of the even step moves around the numbers in the columns/rows in specific ways. The steps work as follows:

Given a list: `[10 14 5 8 7 17 12 1 6 16 9 11 4 15 2 18 3 13]`

Step 0:

Arrange the list in the *RxS* rectangle that follows the properties listed above. Here: R = 6, S = 3 (a divisor of R)

```
10  14  5
8   7   17
12  1   6
16  9   11
4   15  2
18  3   13
```

Step 1:

Sort each column

```
4   1   2
8   3   5
10  7   6
12  9   11
16  14  13
18  15  17
```

Step 2:

Transpose the list. Turn the left most column into the top R/S rows, in order. Turn the next column into the rest R/S rows in order and so on.

```
4   8   10
12  16  18
1   3   7
9   14  15
2   5   6
11  13  17
```

Step 3:

Sort the column

```
1   3   6
2   5   7
4   8   10
9   13  15
11  14  17
12  16  18
```

Step 4:

Perform the INVERSE of the permutation performed in Step 2

```
1   4   11
3   8   14
6   10  17
2   9   12
5   13  16
7   15  18
```

Step 5:

Sort the columns

```
1   4   11
2   8   12
3   9   14
5   10  16
6   13  17
7   15  18
```

Step 6:

Shift the top half of each column into the bottom half of the same column, and shift the bottom half of each column into the top half of the next column to the right. Leave the top half of the left most column empty. Shift the bottom half of the last column into the top half of a new rightmost column, and leave the bottom half of the new column empty.

```
    5   10  16
    6   13  17
    7   15  18
1   4   11
2   8   12
3   9   14
```

Step 7:

Sort each column

```
    4   10  16
    5   11  17
    6   12  18
1   7   13
2   8   14
3   9   15
```

Step 8:

Perform INVERSE of Step 6

```
1   7   13
2   8   14
3   9   15
4   10  16
5   11  17
6   12  18
```

Step 9:

Return the sorted list out the left most column to the right most

`[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18]`

Your code should work for any list of sizes: `18`, `36`, `48`, and `72` that we evaluate your work with. You may want to write a helper function that populates these lists with random numbers automatically when you test your code.

Grading ColumnSort

As with Question 3, you will need to include these steps within your algorithm:

- Write to a file called column1.txt your rectangle, filled with the initial values (step 0)
- Write to a file called column2.txt your rectangle after the first pass of column sorting (step 1)
- Write to a file called column3.txt your rectangle after the first transposition (step 2)
- Write to a file called column4.txt your rectangle after the vertical shift (step 6)
- Return the correctly sorted list

Please use this exact function prototype for this question as your work will be auto-graded in Gradescope.

```
def columnsort(A: list, N: int) -> list
```

SUBMISSION

You will submit ONE file called `assignment1.py` to Gradescope for this lab.

`assignment1.py` should contain your entire source code for this assignment.

The assignment is *autograded*, so be very careful and return appropriate intrmedidate states and final answer for each question as described in the assignment. **If you are not sure ask.**

Late submissions will receive a ZERO.

NOTE: Gradescope allows me to compare all submissions with each other and see how similar they are. If I find submissions which are too similar to each others, all the similar looking assignments will receive a ZERO. Disciplinary action might be taken depending upon the severity of the issue.

Notes on Redirecting Output to a File in Python

Question 3 and 4 require you to write to a file. Although there are many ways to output information to a file, one method is redirecting Python's `stdout`. To do this, first, we need to save the current location of standard output to a variable, so that we'll be able to point stdout back when we're done!

```
<var_name> = sys.stdout
```

When we want to print to a file, we're going to use the following template:

```
# Create and open a file of our choice, and save this path to our variable name
<variable_name_for_our_file> = open('<filename_to_write_to>','w')

# Make standard output point to our file
sys.stdout = <variable_name_for_our_file>

# Using print will now instead print to our file
print("words, and things")

# After writing everything we need to to the file,
<variable_name_for_our_file>.close()
```

Here's a short example with actual variable names that you can try to test things out:

```
# Save current standard output path
orig_stdout = sys.stdout

# Create (and open) a file called smiles.txt, that we can write to
myfile = open('smiles.txt','w')
# Redirect stdout to our file
sys.stdout = myfile

# Print smiles, which will now be written into the file
for i in range(3):
     print(":-)")

# Close our file
myfile.close()

# Set our output back to stdout
sys.stdout = orig_stdout
```

If you have any further questions about manipulating standard output, post on Piazza and we can add more as needed.