DSC 40B - Homework 03

Due: Wednesday, April 24

Write your solutions to the following problems by either typing them up or handwriting them on another piece of paper. Unless otherwise noted by the problem's instructions, show your work or provide some justification for your answer. Homeworks are due via Gradescope at 11:59 p.m.

Problem 1.

Determine the worst case time complexity of each of the recursive algorithms below. In each case, state the recurrence relation describing the runtime. Solve the recurrence relation, either by unrolling it or showing that it is the same as a recurrence we have encountered in lecture.

a)

```
import math
def find_max(numbers):
    """Given a list, returns the largest number in the list.

    Remember: slicing a list performs a copy, and so takes linear time.
    """
    n = len(numbers)
    if n == 0:
        return 0
    if n == 1:
        return numbers[0]
    mid_left = math.floor(n / 3)
    mid_right = math.floor(2n / 3)

    return max(
        find_max(numbers[:mid_left]),
        find_max(numbers[mid_left:mid_right])
        find_max(numbers[mid_right:])
    )
```

b)

```
import math
def find_max_again(numbers, start, stop):
    """Returns the max of numbers[start:stop]"""
    if stop <= start:
        return 0
    if stop - start == 1:
        return numbers[start]

    middle = math.floor((start + stop) / 2)

    left_max = find_max_again(numbers, start, middle)
    right_max = find_max_again(numbers, middle, stop)

    return max(left_max, right_max)
```

c) In this problem, remember that `//` performs *flooring division*, so the result is always an integer. For example, `1//2` is zero. `random.randint(a,b)` returns a random integer in $[a, b)$ in constant time. Note that you are asked to determine the **worst-case** time complexity of the following algorithm.

```
import random
```

```
def foo(n):
    """This doesn't do anything meaningful."""
    if n == 0:
        return 1

    # generate n random integers in the range [0, n)
    numbers = []
    while i < n:
        i = i + 2
        number = random.randint(1, n)
        numbers.append(number)

    x = sum(numbers)

    if x is even:
        return foo(n//2) / x**.5
    else
        return foo(n//2) * x
```

Problem 2.

A *rotated sorted array* is an array that is the result of taking a sorted array and moving a contiguous section from the front of the array to the back of the array. For example, the array `[5,6,7,1,2,3,4]` is a rotated sorted array: it is the result of taking the sorted array `[1,2,3,4,5,6,7]` and moving the first 4 elements, `[1,2,3,4]`, to the back of the array. Sorted arrays are also rotated sorted arrays, technically speaking, since you can think of a sorted array as the result of taking the sorted array and moving the first 0 elements to the back.

For example, all rotated version of `[1,2,3,4,5]` is the following:

- `[1,2,3,4,5]`
- `[5,1,2,3,4]`
- `[4,5,1,2,3]`
- `[3,4,5,1,2]`
- `[2,3,4,5,1]`

The function below attempts to find the value of the minimum element in a rotated sorted array. It is given the array `arr` and the indices `start` and `stop` which indicate the range of the array that should be searched. You may assume the numbers in `arr` are **unique**.
Fill in the blanks to make the function work correctly. Your function should have time complexity $\Theta(\log n)$.

```
import math
def find_min(arr, start, stop):
    """Searches arr[start:stop] for the samllest element.

    Assumes arr is a rotated sorted array.
    """
    if ------: # write doum appropriate base case;
        return -----

    # you may include more than one base case if needed

    mid = math.floor((stop + start) / 2)
```

```
if ----------------:

    return arr[---------]

elif ------------------ :

    return find_min(-----------------)

else:

    return find_min(-----------------)
```

Programming Problem 1.

In a file named `swap_sum.py`, write a function named `swap_sum(A, B)` which, given two **sorted** integer arrays `A` and `B`, returns a pair of indices `(A_i, B_i)` – one from A and one from B – such that after swapping these indices, `sum(B) == sum(A) + 10`. If more than one pair is found, return any one of them. If such a pair does not exist, return `None`.

For example, suppose `A = [1, 6, 50]` and `B = [4, 24, 35]`. Swapping 6 and 4 results in arrays $(1, 4, 50)$ and $(6, 24, 35)$; the elements of each list sum to 55 and 65. Thus, you must return `(1, 0)` as you are expected to return the indices.

Your algorithm should run in time $\Theta(n)$, where n is the size of the larger of the two lists. Your code should not modify `A` and `B`.

This is a coding problem, and you'll submit your `swap_sum.py` file to the Gradescope autograder assignment named "Homework 03 - Programming Problem 01". The public autograder will test to make sure your code runs without error on a simple test, so be sure to check its output! After the deadline a more thorough set of tests will be used to grade your submission.