COL100 Assignment 4

Due date: TBA, 2021

General assignment guidelines

1. **All imperative programs should have relevant assertions given as comments at important points.** For example, for each loop you should state the invariant at the top of the loop, and the post-condition below the end of the loop. These can be stated in words instead of mathematically if you prefer.
2. If an analysis of efficiency is asked, it must be carried out for all functions that use recursion or loops, including helper functions.
3. Big O notation should be as tight as possible. For example, reporting $O(n^2)$ complexity for an $O(n)$ algorithm could lose you marks, even though it is technically correct.
4. For the programming component of each question, the name and the type of the main function must be exactly as specified in the question. You can take help of the test cases provided to validate your code.
5. Any function that has integer input and output must not perform any intermediate computation using floating-point numbers.

1 Making a calculator

In this problem, you have to write a function `evaluate` that takes a string representation of a simple mathematical expression, e.g. `"(1+2)*3"`, and evaluates it to obtain the resulting number.

The input string may include any of the following: positive numbers, `+`, `-`, `*`, `/`, `(`, `)`. The numbers may or may not contain a decimal part, e.g. the number 3 could be given as `"3"` or as `"3.0"` or with any number of zeroes. All expressions will be fully parenthesized, e.g. you will only be given `"(1+(2*3))-4"`, not `"1+2*3-4"`. Your implementation should return the correct result for any possible expression of this form.

This is a complex task that may require multiple helper functions. Here is one suggested

approach, though you may do it in a different way as long as it is correct:

1. Write a function `readNumber(s,i)` that takes a string *s* and an integer *i*, and returns a pair consisting of (i) the number starting at the *i*th character of the string, and (ii) the index after the last character of the number. For example, `readNumber("1+(2.4/3)",3)` should return `(2.4,6)` because `"2.4"` is formed from characters 3 to 5 of the given string.
2. Write a similar function `evalParen(s,i)` that evaluates a single parenthesized subexpression. For example, `evalParen("1+(2.4/3)",2)` should return `(0.8,9)` because the parenthesized subexpression `"(2.4/3)"` starts at character 2 and ends just before character 9.
3. Write `evaluate(s)` in terms of `readNumber` and `evalParen`.
4. Prove the correctness of these functions using the assertions in your code.

Hint: Use recursion to deal with nested parentheses. You may use the built-in `float` function to convert strings to floating-point numbers where needed.

Examples:

- `evaluate("1.0+(3/8))")` should give `1.375`.
- `evaluate("(((2*3)*4)*5)+99")` should give `219.0`.
- `evaluate("1+(((123*3)-69)/100)")` should give `4.0`.

2 A sequence of unique sums

Consider the following sequence of integers:

$$1, 2, 3, 4, 6, 8, 11, 13, 16, 18, 26, 28, 36, 38, 47, 48, 53, \ldots$$

This sequence is constructed as follows. The first two elements are defined to be 1 and 2. The next number is the smallest positive number greater than the last one, which is equal to the *sum of two distinct numbers* (that are already in the sequence) *in exactly one way.*

So the next number after 1 and 2 is trivially 3 = 1 + 2, as there are only two numbers in the starting sequence. The next number after 3 is 4 = 3 + 1. It is also true that 4 = 2 + 2, but this equation does not count, as the two addends have to be distinct.

The next number cannot be 5, as 5 = 1 + 4 but also 5 = 2 + 3: there should only be one way to make a number in the sequence by adding two distinct numbers found in the sequence. The next number is 6 (2 + 4). There are two ways to make 7 (1 + 6 or 3 + 4), so the next is 8 (2 + 6). And so on.

1. Create a function `sumSequence` such that `sumSequence(n)` returns a list of the first n numbers in the above sequence. Your code should use only loops and not recursion. You can use helper functions if you wish, but they should not be recursive either.
2. Prove the correctness of your algorithm.
3. Analyze the time complexity of your algorithm in terms of n and m, where m is the last number in the resulting list.

Examples:

- `sumSequence(4)` should give `[1, 2, 3, 4]`.
- `sumSequence(9)` should give `[1, 2, 3, 4, 6, 8, 11, 13, 16]`.
- `sumSequence(206)[-1]`, i.e. the last number in the list for $n = 206$, should give `1856`.

3 Shortest sublist with sufficient sum

Suppose I have a list of numbers, and I want to select as few of them as possible such that their sum strictly exceeds a given number n. However, the numbers I select must be consecutive, forming a contiguous sublist. So for example if the list is `A = [9,-4,7,5,2,-3,8]` and I want a sum that exceeds 15, I can't simply take the 9 and the 8. I have to select a contiguous sublist of numbers, and the shortest one with large enough sum is `[9,-4,7,5]`.

1. Write a function `minLength` such that `minLength(A,n)` returns the length of the shortest contiguous sublist of A whose sum strictly exceeds n. If no such sublist exists, return `-1`. Your code should use only loops (and helper functions if you wish) but not recursion.
2. Prove the correctness of your algorithm.
3. Analyze the time complexity of your algorithm in terms of the length of the input list.

Examples:

- `minLength([4,5,8,2,-1,3], 9)` should return 2.
- `minLength([3,-1,4,-2,-7,2], 4)` should return 3, because the shortest sublist with sum *strictly* greater than 4 is `[3,-1,4]`.
- `minLength([1,0,0,0,1], 1)` should return 5.
- `minLength([0,1,1,0], 2)` should return `-1`.

4 Merging an contact list (bonus)

Suppose you have a contact list as a list of name & email address pairs, for example:

```
[("RN","narain@cse"), ("HS","saran@cse"), ("RN","Rahul.Narain@iitd")]
```

You want to clean this up to gather all the email addresses of each person together. This will result in a list of pairs where the first component is the name and the second component is a *list* of email addresses, for example

```
[("HS",["saran@cse"]), ("RN",["narain@cse","Rahul.Narain@iitd"])]
```

Note that the order of the pairs, and the order of the email addresses in the inner lists, are not important. It is only required that the email addresses corresponding to the same name are correctly merged.

1. Create a function `mergeContacts` that performs this operation and returns a new list with the merged contacts.
2. Prove the correctness of your algorithm.
3. Analyze the time complexity of your algorithm.

For full marks, your algorithm should take only $O(n \log n)$ time, where n is the number of contacts in the input list. Can you figure out how to do this using sorting?

5 Submission and other logistics

As usual, you should submit two files to the Moodle assignment page.

1. One file should be a PDF file containing all your written work, i.e. correctness and efficiency analysis, etc. This can be handwritten with pen and paper and then scanned with your mobile, or it can be typed on your device using MS Word or LaTeX or any other software. The only requirement is that it should be clearly legible.
2. The other file should be a Python file containing all your code. Put your solutions to all four programming problems into a single file, along with any helper functions they require. We should be able to run any of the required functions by running your code in a Python console and then typing in a function call.

The filenames of your submitted files should be exactly the same as your entry number. For example, if your entry number is 2017CS10389, you should name your Python file as `2017CS10389.py` and your PDF file as `2017CS10389.pdf`. Your submission should consist of only these two files, uploaded individually. There should be no sub-folders or zip files in the submission on Moodle.

Failure to comply with these submission instructions will lead to a penalty.

Please ask any queries related to the assignment on the COL100 Piazza page (`https://piazza.com/iit_delhi/fall2020/col100`).