Homework 6: Efficiency

This homework does not have a starting file. You should make a file called hw6.py and put all your work there. The descriptions of what should be done in that file are below.

This homework also requires a meaningful amount of work that is done outside the hw6.py file in which you are coding. You can type it if you want, or just handwrite it separately and attach it to the code when you submit it.

Exercise 1

The following is a series of run times that represent the time it takes various algorithms to finish. You should sort these run times from fastest to slowest, considering only the asymptotic relationship between the run times. (Here "fastest" refers to the algorithm, so log(n) is "faster" than 2^n.) If there are sets of run times that are asymptotically equal, put them next to each other in arbitrary order and circle them to show that they're equivalent.

n^2, $\log_3(n)$, $\log_{10}(n)^2$, 2^{n+1}, $\sqrt{n}$, 3^n, $7n^3 + 10n$, $5n$, $\log_5(n)$, 1000, 2^n, $2^{n/2}$, $n\log_2(n)$, $4n^3$

Instructions for the following exercises

In the following exercises you are given a function. You should write code for the function in hw6.py. You should then also write (outside hw6.py) an analysis of the running time of the function you have written. That is, you should say what its asymptotic runtime is, along with a short description of how you know that. Any correct solution that is correctly analyzed will get full credit. However, there are multiple ways to write these functions, and they don't all have the same (asymptotic) runtime. There will be bonus points if you find the faster ways of doing them.

Exercise 2: `longPalSub`

The function `longPalSub` should return the longest palindromic substring of its input string. For example, if the input is `"cbcbaabaca"` the output should be `"baab"` because that is the longest substring of the input that is a palindrome. (A palindrome is a string that reeds the same forward and backwards. `"cbc"`, for example, is also a palindrome but is too short.) If there is a tie for the longest palindromic substring in the input, the function can output any of the tied substrings arbitrarily.

Exercise 3: `twoSum`

The function `twoSum` is given as input a list of integers and a target value. The function should return `True` if there are two numbers in the list that add up to the target value and `False` otherwise. For example, `twoSum([-4, 7, -2, 1, 3], -1)` should return `True` because -2 and 1 (or -4 and 3) sum to -1. If the target was 6 instead of -1, it should return `False`.

Exercise 4: `threeSum`

The function `threeSum` is identical to `twoSum`, except that it checks for a set of three numbers that sum to the target in question, rather than two.

Exercise 5: `maxSubarray`

The function `maxSublist` takes as input a list of integers and returns the sublist that has the maximum total sum (again, breaking ties arbitrarily). For example `maxSublist([2,1,-3,4,-1,2,1,-5,4])` returns `[4,-1,2,1]` because the sum of that sublist (6) is the largest possible.