Review of the first 18 lectures

The first midterm exam will be closed book and closed notes. Calculators or laptop computers are not allowed. Good examples of questions are quizzes and homework exercises.

The material is focused on the first 18 lectures. Of those 18 lectures, the first seven are devoted to general Python programming, while the theme of the last eleven lectures is recursion.

This sheet contains some preliminary examples of questions which may help you prepare for the first midterm exam. This list is by no means exhaustive.

1. Consider an n-by-m matrix A represented as a list of n arrays, where each array holds m integer numbers.

   (a) Write a script that prompts the user for the dimensions n and m of the matrix A and that then generates random numbers uniformly distributed, in the range from 10 to 99.

   (b) Make one long list L of $n \times m$ numbers stored in A.

   (c) Reshape the matrix A from dimensions n-by-m to m-by-n, distributing the $n \times m$ numbers in the list L among the m rows of n integer numbers. The i-th number in the list L is placed at row $i//n$ and column $i\%n$ of the reshaped matrix.

2. For $n > 1$, the n-th hailstone $h(n)$ is defined as $h(n) = h(n/2)$, if n is even, and $h(n) = h(3n+1)$ if n is odd. If $n = 1$, then this recursion stops. Write a recursive Python function to compute these hailstones. Use an extra accumulating parameter to trace the execution.

3. Give a ***recursive*** definition of a Python function (call it `strlen`) to determine the length of a string given on input. The function returns the number of characters in the string.

4. Write a ***recursive*** Python function `CharCount` which takes on input a string `s` and a character `c`. The function returns the number of times the character `c` occurs in the string `s`.

5. Describe a ***recursive*** algorithm to revert the order of the elements in a list. The elements of the input list will appear in reverse order in the output list returned by the Python function.

6. The trapezoidal rule to approximate the integral of a function $f(x)$ over an interval $[a, b]$ is defined by $\frac{b-a}{2}(f(a)+f(b))$. Give a Python function that applies this rule recursively k times, each time splitting the interval in two equal halves and applying the rule to each subinterval. Why is this recursive solution not so efficient? What could you do to make it more efficient?

7. The value of the n-th Chebyshev polynomial $T(n, x)$ is recursively computed as follows

$$T(0, x) = 1, \quad T(1, x) = x, \quad T(n, x) = 2xT(n-1, x) - T(n-2, x), \text{for } n \geq 2.$$

   Define a function to compute the value for $T(n, x)$ ***recursively*** and ***efficiently***.

8. Consider again the definition of the Chebyshev polynomials $T(n, x)$. Define a function to compute the value for $T(n, x)$ ***iteratively***, using a stack of function calls.

9. Consider a list `L` of positive numbers. Give an algorithm to find that triplet of elements in `L` whose sum is maximal. Given the list `L` on input, the Python function returns the triplet `(i,j,k)` so that `L[i] + L[j] + L[k]` is maximal over all other choices of three numbers in `L`.

10. Files on disk are organized as a tree. Describe an algorithm to enumerate all directories and files on disk using an infix traversal. Use the functions in the `os` module to implement this traversal in Python.

11. For repeated efficient searches for file names on disk, it might be convenient to build a binary tree of file names to enable binary search for a name of a file on disk. Give a file name, the search would return the complete path name if the file exists on disk. Describe a class `FileTree`, its design and content to enable such searches.

12. Write a Python script that takes on input a list L and a number k on input. The script prints all different selections of k elements of the list L. For example, if $L = [1, 2, 3, 4]$ and $k = 3$, then the script prints the lists $[1, 2, 3]$, $[1, 2, 4]$, $[1, 3, 4]$, and $[2, 3, 4]$.

    You may not make assumptions on the length of the list on input, but you may assume that all elements of the given list are different.

    Specify first all parameters of the recursive function. For each parameter, write one sentence to define its role in the recursive solution.

13. A proper partition of a number `n` is a list of strictly increasing positive numbers which all add up to `n`. For example, 5 has two proper partitions: `[1, 4]` and `[2, 3]`.

    To print all proper partitions of `n`, use a recursive function `enum`. Describe below the parameters of `enum`. For every parameter write a sentence about its meaning.

14. Consider any list of objects. Do not assume all objects in the list are numbers (but they could be) or of any particular type. All permutations of the list `['a','+',1]` are `['a', '+', 1]`, `['a', 1, '+']`, `['+', 'a', 1]`, `['+', 1, 'a']`, `[1, 'a', '+']`, and `[1, '+', 'a']`. Develop Python code to write all permutations of a given list.

    Describe the algorithm to enumerate all permutations. What are the parameters of the recursive function you will use?

15. Write a recursive function that enumerates all ways to write a given number `n` as a sum of squares, where every square is strictly larger than one and the squares are weakly increasing. For example, for `n = 25`, the program writes the following three lines:

```
25 = 2^2 + 2^2 + 2^2 + 2^2 + 3^2
25 = 3^2 + 4^2
25 = 5^2
```

    Call the recursive function `SumOfSquares`. Describe its parameters: for every parameter write a sentence about its meaning.

Please note the policy on skipping exams: If an exam is missed, then greater weight will be placed on the final exam, especially on the material covered on the missing exam. Please be prepared when you show up for the exam. Skipping this midterm exam will make an application for an incomplete at the end of the semester very difficult.