ST444: Exercise Sheet 4

Part I: Python

1. Create a function that reads in a positive integer and then returns a list of all the divisors of that number. (If you don't know what a divisor is, it is a number that divides evenly into another number. For example, 13 is a divisor of 39 because 39 / 13 has no remainder.)

2. The greatest common divisor (GCD) of two positive integers m and n is the greatest integer that divides both m and n with no remainder. Write a function that returns the GCD of two positive integers using **recursion**.

   (Hint: use the "Euclid's algorithm" (google if you don't know); alternatively, note that if $m > n$, $\mathrm{GCD}(m, n) = \mathrm{GCD}(m - n, n)$)

3. Write a function that takes a positive integer N (in decimal) and return its binary representation. For example, $8 = 1000$ and $366 = 101101110$.

4. Write a program that reads in a positive integer n and prints out all $n!$ permutations of the n letters starting at a (assume that n is no greater than 26). A permutation of n elements is one of the $n!$ possible orderings of the elements. As an example, when $n = 3$ you should get the following output.

   ```
   bca cba cab acb bac abc
   ```

   Don't worry about the order in which you enumerate them.

   (Hint: use recursion.)

5. ($\star$) This is a really tricky question. Consider the following recursive function. Without running the code, could you figure out `f(0)`?

   ```
   def f(x):
       if (x > 1000):
           return (x - 4)
       else:
           return f(f(x+5))
   ```

6. Write a function that performs the bubble sort on a list of floating-point numbers. Try it out on a list of 10^4 elements. How does it compare with the default sorting method `list.sort()`? In Python, to create a list of 10^4 random elements from $U[0, 1]$, one can use the following commands (we will discuss Modules in Python shortly):

   ```
   import random
   # the following line allows us to get a pseudo-uniform[0,1]
   random.uniform(0, 1)
   # the following three lines gives us a list of iid U[0,1]
   xs = []
   for x in range(10000):
       xs.append(random.uniform(0, 1))
   ```

7. Implement the Gauss elimination.

   (Hint: in Python, a matrix can be represented as "a list of lists".)

8. Implement the Cholesky decomposition.

Part II: Matrix computation (theory)

1. Consider the system of linear equations

$$\begin{aligned} x_1 + 4x_2 + x_3 &= 12 \\ 2x_1 + 5x_2 + 3x_3 &= 19 \\ x_1 + 2x_2 + 2x_3 &= 9 \end{aligned}$$

   (a) Solve the system using Gaussian elimination with partial pivoting.

   (b) Solve the system using Gaussian elimination with full pivoting.

2. Verify the correctness of the Cholesky decomposition and Crout's LU decomposition.

3. A square matrix $\mathbf{A} = \{a_{ij}\}$ is *tridiagonal* if $a_{ij} = 0$ whenever $|i - j| > 1$, i.e.,

$$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ & a_{32} & \ddots & \ddots & \\ & & \ddots & \ddots & a_{n-1,n} \\ & & & a_{n,n-1} & a_{n,n} \end{pmatrix}$$

   For a given $n \times n$ tridiagonal matrix $\mathbf{A}$ and a $n \times 1$ vector $\mathbf{y}$, find an algorithm that solves the linear system $\mathbf{Ax} = \mathbf{y}$. What's the time complexity of your algorithm? Is it rate optimal?

   (Hint: find an $O(n)$ algorithm)

4. Given the sample covariance matrix of the daily returns of n different stocks, Σ. You will invest p_j proportion of the total asset into the j-th stock for $j = 1, \ldots, n$. Here short selling is allowed, so the only constraint on $p_1, \ldots, p_n$ is $p_1 + \cdots + p_n = 1$. How to construct a portfolio that is most volatile? And how to construct a portfolio that is least volatile?

5. ($\star$) Recall the matrix multiplication example from Lecture 1. Let n be an positive integer, and let $a_1, \ldots, a_{n+1}$ be $n+1$ positive integers. Suppose that you are given n matrices, where the i-th matrix $\mathbf{B}_i$ is $a_i \times a_{i+1}$. How to determine the optimal parenthesization of a product of these n matrices, $\mathbf{B}_1\mathbf{B}_2 \cdots \mathbf{B}_n$?

   (Hint: use dynamic programming; for more details, search for "matrix chain multiplication")