1. **Matrix Multiplication**

   Given two \( n \times n \) matrices \( A \) and \( B \), write an algorithm to compute the product of \( A \) and \( B \). Then, analyze your algorithm to find out how many additions and multiplications your algorithm will take. Then analyze the running time of this algorithm using big-O notation.

2. **Time analysis**

   What is the running time of the following algorithm (using big-O notation)? Show your analysis.

   ```
   1: result ← 1
   2: for \( i = 1 \) to \( n \) do
   3:     for \( j = 1 \) to \( i \) do
   4:         result ← result + 1
   5:     end for
   6: end for
   ```

3. **Stable Matching: A Generalization**

   This problem is taken from the book chapter posted on the class web page (Exercise 4 on p.23). One generalization of stable matching discussed in class is to consider the situation of National Resident Matching Program, which matches medical students to hospitals. Here, there are \( m \) hospitals and \( n \) students. Each hospitals has one or more openings for residents, and each student can only work for one hospital. We assume there are more students than the number of openings. That is, there will be students who can not find positions. Like before, each student has a ranking of (all) hospitals and each hospital has a ranking of (all) students. The problem is to find a stable assignment of students to hospitals, so that all openings are filled. We say an assignment is stable if neither of the following occurs:

   1. First type of instability. There are two students, \( s, s' \), and a hospital \( h \), where \( s \) is assigned with \( h \), \( s' \) is free, and \( h \) prefers \( s' \) over \( s \).

   2. Second type of instability. There are two students, \( s \) who is assigned to hospital \( h \), and \( s' \) who is assigned to \( h' \). But \( h \) prefers \( s' \) over \( s \) and \( s' \) prefers \( h \) over \( h' \).

   Now show that there always exists a stable assignment by giving an efficient algorithm for this problem.

4. **The Pancake Problem**

   A stack of \( n \) pancakes is placed in front of you. You have a spatula which you can insert anywhere into the stack and flip over all the pancakes above the spatula. You want to arrange the pancakes in order of their diameter (they are perfectly round), and you want to use as few flips as possible. As an example suppose \( n = 6 \), and the pancakes are numbered 1 through 6 in order of their diameter with 1 the smallest and 6 the largest. Suppose the original order is 346215, and the left end of the sequence represents the top of the stack. In one flip I can get 643215 (by flipping the first three pancakes: 346), then in the next flip 512346, then 432156, then 123456, so four flips are enough in this case. Let \( F(n) \) be the worst case number of flips needed to arrange a stack of \( n \) pancakes. Find an efficient (in a worst case sense) algorithm for this problem, where efficiency is measured by the worst case number of flips.

   Remember that your algorithm should work for pancakes with any order. To start you off you should easily be able to show that \( F(n) \) is at most \( 2n-2 \). Next, reduce that bound a little more if you can.