# PRACTICE PROBLEMS FOR THE FIRST MIDTERM
# CSCI 417

The first midterm will cover everything that we have learned so far. There will be a mix of proof questions and coding questions.

## 1. Proof questions

**Problem 1.** Use the Euclidean algorithm to calculate $\gcd(120, 26)$.

**Problem 2.** Use the method of least remainder to calculate $\gcd(92, 24)$.

**Problem 3.** Show that the sum of an odd number and odd number is an even number.

**Problem 4.** Recall that a complex number $z$ is a number of the form $a+bi$ where $a, b \in \mathbb{R}$. The magnitude of $z$, denoted by $|z|$ is defined as

$$|z| = \sqrt{a^2 + b^2}.$$

Let $z_1, z_2$ be two complex numbers. Show that $|z_1||z_2| = |z_1 z_2|$.

**Problem 5.** Use Gauss's method to calculate the following sum

$$S_n = 1 + 4 + 7 + \ldots + (4n - 1).$$

**Problem 6.** Convert the following numbers into decimal

(1) $\overline{122}_4$
(2) $\overline{1124}_5$.
(3) $\overline{AB1}_{16}$.

**Problem 7.** In American Football, the possible scores are as follows

- Touchdown (TD): 6 points
- Extra Point (PAT): 1 point (after TD)
- Two-Point Conversion: 2 points (after TD)
- Field Goal (FG): 3 points
- Safety: 2 points

Let $F_n$ be the number of distinct combinations of these scores that total $n$ points.

(1) Compute $F_0, F_1, F_2, F_3$.
(2) Find a recursive formula for $F_n$.

**Problem 8.** Let $X = \{0, 1, 2\}$. An $X$-string of length $n$ is a sequence of the form $a_0 a_1 \ldots a_{n-1}$ such that $a_i \in X$ for each $0 \leq i \leq n - 1$. Let $p(n)$ be the number of $X$-strings of length n that contain no adjacent zeros.

(1) Find a recursive formula for $p(n)$.

(2) Use your recursive formula to calculate $p(5)$.

**Problem 9.** Give the Big-O performance of the following code fragments:

(1)
```
for i in range(n):
    for j in range(i):
        k = i + j
```

(2)
```
i = n
while i > 0:
    k = 2 + 2
    i = i // 2
```

(3)
```
i = 2
while i < n:
    k = 3+ 4
    i = i**2
```

**Problem 10.**

(1) **True/False**: The time complexity of accessing an element at a specific index in a list is $O(1)$.
(2) **True/False**: Inserting an element at the end of a list in Python has an average time complexity of $O(1)$.
(3) **True/False**: The time complexity for searching for an element in a dictionary (hash table) is $O(n)$.
(4) **True/False**: Removing an element from a set has an average time complexity of $O(1)$.
(5) **True/False**: The time complexity of a full scan to find an element in an unsorted list is $O(\log n)$.
(6) **True/False**: Inserting a new key-value pair into a dictionary has an average time complexity of $O(1)$.
(7) **True/False**: The time complexity for accessing a value using its key in a dictionary is $O(n)$.
(8) **True/False**: The time complexity for sorting a list using the built-in `sorted()` function in Python is $O(n \log n)$.
(9) **True/False**: Removing an element from the beginning of a Python list has a time complexity of $O(1)$.

## 2. Coding questions

**Problem 11.** Recall the following sum from Problem 5.

$$S_n = 1 + 4 + 7 + \ldots + (4n - 1).$$

(1) Write a recursive function to calculate $S_n$.

(2) Write an iterative fucntion to calculate $S_n$. What is the big $O$-performance of your algorithm?

**Problem 12.** Write a function to check if a number is a prime. Recall that a number $n$ is prime if the only divisors of $n$ are 1 and $n$. What is the time complexity of your algorithm. For this problem we are assuming that standard opeators such as modulo/multiplication is $O(1)$ (this assumption might not be true if $n$ is really big).

**Problem 13.** A number $n$ is called a perfect if the sum of its divisors is equal to $2n$. For example, 6 is perfect because all divisors of 6 are $\{1, 2, 3, 6\}$ and

$$1 + 2 + 3 + 6 = 2 \times 6.$$

On the other hand, 8 is not perfect because

$$1 + 2 + 4 + 8 = 15 \neq 2 \times 8.$$

Write a program to check whether a number is perfect. Can you do so with a $O(\sqrt{n})$ time complexity?

**Hint:** If $d$ is a divisor of $n$ then $n/d$ is also a divisor of $n$.

**Problem 14.** This problem is related to Problem 7.

(1) Write a recursive function to calculate $F_n$.
(2) Write an iterative solution to calculate $F_n$.

**Problem 15.** Given a list of strings, write a function named longest_string(alist) that returns the longest string. If there are multiple strings of the same maximum length, return the one that appears last in the list. For example

```
string_list = ["apple", "banana", "cherry", "kiwi", "banana"]
result = longest_string(string_list)
print(result)  # Output: "banana"
```

What is the big-O performance of your program?