This is a *warm-up/review* problem set covering topics from prerequisites you will need for CompSci 161. The purpose of this problem set is for you to establish how well prepared for the course you are. While you might not be able to answer each of these offhand, you should be able to with a little bit of work and review. If you have trouble with these problems, that should indicate to you that you have some prerequisite material to review. If I were to assign and collect a problem set 0 rather than providing this, it would probably be 4-5 problems similar to the ones here.

How to use this document: Read through the questions so you can get an idea for what sorts of things you would ideally be able to answer. Note that I am not asking you to answer them in any very constrained period of time – for that matter, some of these are of too high a difficulty level to give as a question during a test of prerequisites, which I assign some quarters but I do not plan to do in Winter 2023. Rather, choose a few problems, attempt them, and then discuss them with your study group, who will ideally have already attempted them as well. *Note that looking up an answer, or me providing one for you, would defeat the point of this exercise. You should never attempt to look up the solution to something in this class beyond what is provided by the professor during the quarter.* Then, repeat with other questions, until you have completed the set.

You should think about whether you were able to start a problem effectively, whether the terms used made sense to you, and whether you would be able to explain your technique to a friend in your study group who has also attempted the problem. *For most of these exercises, the best feedback is not whether or not you solved it correctly, but the ease or lack thereof with which you were able to attempt them.*

We will very likely discuss a subset of these problems, or problems similar to them, in lecture and discussion during week one of the quarter.

1. Recall the linear search algorithm:

```
int linearSearch(const std::vector<int> & numbers, int target)
{
    int i;
    int n = numbers.size();
    for(i=0; i < n; i++)
    {
        if( numbers[i] == target )
        {
            return i;
        }
    }
    throw ElementNotFoundException("Element not found by linear search.");
}
```

   Would it be correct to describe this algorithm as having a running time of $\Omega(n)$, where n is the size of the input vector? In 1-3 sentences, justify your answer.

2. Recall the following algorithm, which determines if a given input positive integer is prime:

```
bool isPrime?(positive integer $n$)
  **if** $n = 1$ **then**
    **return** false
  **else if** $n$ is 2 **then**
    **return** true
  **else if** $n$ is even **then**
    **return** false
  **for** $i$ from 3 to $\lceil\sqrt{n}\rceil$ by twos **do**
    **if** $i$ divides $n$ leaving no remainder **then**
      **return** false
  **return** true
```

   (a) Express the running time of this algorithm in $\mathcal{O}$ notation in terms of n, the value of the input number provided. Assume that each call to the divisibility test inside the for loop takes constant time.

   (b) We use the phrase "polynomial time" to refer to running time that is a polynomial function *of the input size* to the problem. Does this algorithm have a polynomial running time or not?

3. We say a positive integer n is "resolute" if 3 evenly divides (that is, leaves no remainder when used as a divisor) $n^2 + 2n$. I put forth a claim that all odd positive integers are resolute. Demonstrate that this claim is incorrect.

4. Multiple students have reported having been asked a version of the following question in an interview. I liked it so much I wanted to put it on this problem set.

   Suppose you have a vector V of n distinct numbers. The vector is sorted least to greatest. You want to create a vector V', such that each value in V has its square somewhere in V'. If x and $-x$ are both in V, it is okay to have x^2 once or twice in V'. Regardless, you want V' to be in sorted order.

   (a) Give an algorithm to find V' in $\mathcal{O}(n \log n)$ time.

   (b) Can you find an algorithm with running time $\mathcal{O}(n)$?

   (c) Can you find an algorithm with running time better than $\mathcal{O}(n)$? Why or why not?

5. Suppose we have an array A of n professors; each teacher has two characteristics: difficulty (a real number in the range $[0, 10]$, where a higher number indicates a more difficult teacher) and humor (a real number in the range $[0, 100]$, where a higher number indicates a funnier teacher). You may assume that each value is distinct (no two professors are exactly as difficult or exactly as funny), but not that the precision of real numbers is limited (as floats and doubles are in C++ and Java). Our goal is to determine a set of professors that might satisfy an answer to the question "who is the easiest teacher that is the funniest?" We wish to avoid professors with high difficulty and low humor. We would like to find the largest subset of the input data such that no professor in the chosen subset is **both** less funny **and** more difficult than another professor in the original input set. Note that if professor A is easier than professor B, it *does not follow* that professor A is also funnier than professor B.

   For example, if our dataset is:

Professor	Difficulty	Humor
Venabili	4	65
Jones Jr	8	15
Jones Sr	8.5	2
Grant	2	35
Moriarty	10	0
Plum	9	85
Walsh	8.2	90

   Then we want to return Venabili, Grant, and Walsh.

   Note that none of these are meant to refer to anyone at UC Irvine and any similarity is unintentional and a coincidence.

   (a) Devise an algorithm which takes as input A and n, and outputs the resulting set. Your algorithm does not need to be particularly efficient, but you may not give an algorithm that enumerates every subset.

   Do not have your answer depend on limitations of any programming language. For example, while difficulty or humor are numeric types, do not use that, say, a **double** in C++ is "not really" a real number.

   (b) Analyze the worst-case runtime of your algorithm using Θ-notation.

   (c) Do you believe your algorithm has obtained the best possible asymptotic runtime? Explain your reasoning. Note that finding the best possible asymptotic runtime **is not** required to get full credit on this question.

6. Suppose you have two max-heaps, A and B, with a total of n elements between them. You want to discover if A and B have a key in common. Give a solution to this problem that takes time $\mathcal{O}(n \log n)$. You need only return a boolean value (true or false), not the key in common. Note that the problem does not ask for a specific key in common, but rather asks if there exists a key that they do have in common. For this problem, do not use the fact that heaps are arrays. Rather, use the API of a heap; that is, you may call the following functions:

   - `max()`, which returns the maximum element in the heap at the moment in time $\mathcal{O}(1)$.
   - `extractMax()`, which removes the maximum element in the heap, returning nothing, and takes $\mathcal{O}(\log n)$ time.
   - `insert(e)`, which inserts the parameter into the heap, taking $\mathcal{O}(\log n)$ time.

   Each of these functions takes the time that they did when you learned them in a course like ICS 46. The heaps are implemented as binary heaps via an array (or equivalent).

   Give a brief explanation for why your algorithm has the required running time.

   You may allocate $\mathcal{O}(1)$ additional memory in solving this problem.

7. Consider the following problem that humanity might face when we make first contact with an alien race. We know that the aliens' written language operates in a manner similar to English; their alphabet contains n characters, which we will denote $c_1, c_2, \ldots c_n$. We also know that they have a concept of *alphabetical order*, also similar to English, where some character (not necessarily c_1) is first (like 'A' in English), some other character is second, and so on. Comparison of two words for alphabetical order works the same way in their language that it does in English.
   We are given a list of m words in the aliens' language, written in an order that follows their alphabetical order. Our goal is to determine a *feasible* alphabetical order for the aliens' alphabet that is consistent with the word list given.
   Give an efficient algorithm to solve this problem and analyze its runtime. Your algorithm should output the feasible alphabetical order for the aliens' alphabet if one is consistent with the input; alternately, if we have evidence that the aliens lied to us and the list *is not* in alphabetical order, your algorithm should output the evidence of this.

8. (a) Use the Euclidean algorithm to find $\gcd(698, 277)$
   (b) Find integers s and t such that $698s + 277t = 1$.
   (c) You have two bottles: one can hold exactly 698 fluid ounces of water and another that can hold exactly 277 fluid ounces. Neither of the bottles has any markings (although you do know their capacities). Describe a way to measure *exactly one* fluid ounce of water.

9. What is 11^{152} **mod** 77? **Show your work.** The following values should help you to compute this value.

$$\begin{aligned} 11^1 &\equiv 11 \bmod 77 \\ 11^2 &\equiv 44 \bmod 77 \\ 11^4 &\equiv 11 \bmod 77 \end{aligned}$$

10. Prove that the difference of any two odd perfect squares is always a multiple of 4.

11. Prove that if a number is 2 more than a multiple of 4, it is not a perfect square.

12. Recall the Fibonacci sequence, where $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$. Prove that, for all non-negative integers n, F_n is even if and only if n is divisible by 3.

13. Suppose we seat 25 computer science majors and 25 philosophy majors (no double majors in this problem) around a circular table. Show that there is always a person seated with two philosophy majors adjacent to them.

14. Suppose we have a fair die that has twelve (12) sides. That is, if we roll it, each of the first 12 positive integers are equally likely to be the result of the roll.

    For this problem (and ones like it), you are encouraged to leave your answer in terms of "n choose k" (often written as $\binom{n}{k}$) and other mathematical expressions such as summations.

    (a) If we roll the die, what is the probability the result is prime? As a reminder, one is **not** a prime number.

    (b) Suppose we roll this die 1000 times. What is the probability we get a prime number exactly 200 times?

    (c) Suppose we roll this die 1000 times. What is the probability we get a prime number at least 200 times?

    (d) What is the expected value of a single roll of this die?

15. A "fun sized" (Halloween) bag of Skittles candy has 20 Skittles (pieces of candy) in it. Each piece of candy is one of five flavors (grape, lemon, orange, strawberry, or green).

    Two bags of skittles are considered identical if the mix of pieces is the same in each. For example, a bag containing ten grape and ten lemon candies is the same as every other bag with ten grape and ten lemon, regardless of the order the grape or lemon candies were added to the bag.

    (a) How many different fun-sized bags of Skittles are possible?

    (b) How many different fun-sized bags of Skittles are possible if each must contain at least one of each flavor?

    (c) How many different fun-sized bags of skittles are possible if (i) we no longer have the restriction that at least one of each flavor must be chosen and (ii) a bag contains *at most* 20 candies (and at least one piece of candy), rather than it must contain exactly 20?