- Homework solutions should be neatly written or typed and turned in through **Gradescope** by 11:59pm on the due date. No late homeworks will be accepted for any reason. You will be able to look at your scanned work before submitting it. Please ensure that your submission is legible (neatly written and not too faint) or your homework may not be graded.
- Students should consult their textbook, class notes, lecture slides, instructors, TAs, and tutors when they need help with homework. Students should not look for answers to homework problems in other texts or sources, including the internet. Only post about graded homework questions on Piazza if you suspect a typo in the assignment, or if you don't understand what the question is asking you to do. Other questions are best addressed in office hours.
- Your assignments in this class will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should always explain how you arrived at your conclusions, using mathematically sound reasoning. Whether you use formal proof techniques or write a more informal argument for why something is true, your answers should always be well-supported. Your goal should be to convince the reader that your results and methods are sound.
- For questions that require pseudocode, you can follow the same format as the textbook, or you can write pseudocode in your own style, as long as you specify what your notation means. For example, are you using "=" to mean assignment or to check equality? You are welcome to use any algorithm from class as a subroutine in your pseudocode. For example, if you want to sort list A using `InsertionSort`, you can call `InsertionSort(`A`)` instead of writing out the pseudocode for `InsertionSort`.

There are 5 questions for a total of 55 points.

---

1. (10 points) Let $f(n)$ and $g(n)$ be functions from the nonnegative integers to the positive real numbers. Prove the following transitive property from the definition of big-O:

$$\text{If } f(n) \in O(g(n)) \text{ and } g(n) \in O(h(n)) \text{ then } f(n) \in O(h(n)).$$

Solution: Assume $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$. Then by definition of O, there is a c_1 and N_1, so that, for $n \geq N_1$, $f(n) \leq c_1 \cdot g(n)$; and a c_2 and N_2 so that, for $n \geq N_2$, $g(n) \leq c_2 \cdot h(n))$. Then for $N = \max(N_1, N_2)$ and $c = (c_1 c_2)$, for any $n \geq N$, $f(n) \leq c_1 \cdot g(n) \leq (c_1 c_2) \cdot h(n) = c \cdot h(n)$. So $f(n) \in O(h(n))$ by the definition of O.

2. State True or False:

(a) (2 points) $2 \cdot (3^n) \in \Theta(3 \cdot (2^n))$

(a) **False**

Reason/Method: (*You were not expected to write this*)
The constants 2 and 3 do not change order. $3^n/2^n = (1.5)^n$ which goes to infinity as n gets large. So $2 \cdot 3^n \in \omega(3 \cdot 2^n)$, and they are not the same order.

(b) (2 points) $(n^6 + 2n + 1)^2 \in \Omega((3n^3 + 4n^2)^4)$

(b) **True**

Reason/Method: (*You were not expected to write this*)
We don't need to compute ratios exactly. For each expression, the highest degree part, which determines the order, is something times n^{12}. So they have the same order.

(c) (2 points) $\log n \in \Omega((\log n) + n)$

(c) **False**

Reason/Method: (*You were not expected to write this*)
Since $\log n$ is not $\Omega(n)$.

(d) (2 points) $n \log n + n \in O(n \log n)$

(d) **True**

Reason/Method: (*You were not expected to write this*)
A sum of a fixed number of functions has the same order as its maximum term, in this case, $n \log n$.

(e) (2 points) $\log(n^{10}) \in \Theta(\log(n))$

(e) **True**

Reason/Method: (*You were not expected to write this*)
$\log(n^{10}) = 10 \log n \in \Theta(\log n))$

(f) (2 points) $\sum_{i=1}^{n} i^k \in \Theta(n^{k+1})$

(f) **True**

Reason/Method: (*You were not expected to write this*)
We don't need to give an exact expression. The largest term in the sum is n^k and there are n terms, so the sum is upper bounded by n^{k+1}, showing it is $O(n^{k+1})$. To see the other direction, we can't use the smallest term, because that is just 1, but we can use the middle term: $i = n/2$ and all $n/2$ greater terms. Each of those is at least $(n/2)^k = n^k/2^k$ and there are $n/2$ of them.

So the sum is at least $n^{k+1}/2^{k+1}$ which is $\Omega(n^{k+1})$ for any fixed k.

(g) (2 points) $(\log(n))^{\log(n)} \in O(n/\log(n))$

(g) **False**

Reason/Method: (*You were not expected to write this*)
This one is a bit tricky. The left is an exponential, but an exponential of a log, which are inverses of each other. For any fixed b, $b^{\log n}$ is a polynomial. But as b increases, the degree of that polynomial grows, and we are comparing on the right to a sub-linear function. So we could argue: For $n \geq 16$, $\log n \geq 4$, So $(logn)^{\log n} > 4^{\log n} = 2^{logn}2^{\log n} = n * n = n^2 \in \omega(n/logn)$.

(h) (2 points) $n! \in O(2^n)$

(h) **False**

Reason/Method: (*You were not expected to write this*)
If we look at the ratio, $n!/2^n = 1/2 * 2/2 * 3/2 * 4/2 * ...n/2$. All but the first three terms are ≥ 2. So the ratio is at least $(3/4)(2^{n-3}) = (3/32)2^n$ which goes to infinity quickly as n increases.

3. Given two lists A of size m and B of length n, our goal is to construct a list of all elements in list A that are also in list B. Consider the following two algorithms to solve this problem.

```
procedure Search1(List A of size m, List B of size n)
1.  Initialize an empty list L.
2.  SORT list B
3.  for each item a ∈ A,
4.      if (BinarySearch(a, B) ≠ 0), then
5.          Append a to list L.
6.  return L
```

Note: Assume that the `SORT` algorithm used in `Search1` algorithm above takes time proportional to $k \log k$ on an input list of size k.

```
procedure Search2(List A of size m, List B of size n)
1.  Initialize an empty list L
2.  for each item a ∈ A,
3.      if (LinearSearch(a, B) ≠ 0), then
4.          Append a to list L.
5.  return L
```

Answer the following questions:

(a) (2 points) Calculate the runtime of `Search1` in Θ notation, in terms of m and n.

(b) (2 points) Calculate the runtime of `Search2` in Θ notation, in terms of m and n.

(c) (2 points) When $m \in \Theta(1)$, which algorithm has faster runtime asymptotically?

(d) (2 points) When $m \in \Theta(n)$, which algorithm has faster runtime asymptotically?

(e) (2 points) Find a function $f(n)$ so that when $m \in \Theta(f(n))$, both algorithms have equal runtime asymptotically.

Solution: For the first algorithm, line 1 is constant time, line 2 takes $O(nlogn)$ time from the assumption, and we go through a loop m times, with each iteration taking $O(\log n)$ time. So the total time is $O((m+n)\log n)$. For the second algorithm, we do a linear search in an array of size n, which takes time $O(n)$ a total of m times. So the total time is $O(nm)$. (There actually is a whole research area called "database cracking" that addresses the question of whether, when, and how much to sort the data in a database, for precisely the reason that these two are incomparable). If m is constant, the first is still $O(n \log n)$ because we need to sort, but the second becomes $O(n)$, so the second is faster. If $m \in \Theta(n)$, the first is much faster, $O(n \log n)$ vs. $O(n^2)$ for the second. The break-even point is when $n \log n$ is of the same order as nm, or when $m \in \Theta(\log n)$, when both are $O(n \log n)$.

4. Show that if c is a positive real number, then $g(n) = 1 + c + c^2 + ... + c^n$ is:

   (a) (3 points) $\Theta(1)$ if $c < 1$.

   Solution: The trichotomy here will be very useful in analysing divide and conquer algorithms. By the formula for the sum of a geometric series, when $0 < c < 1$ is any constant,, $g(n) = (1 - c^{n+1})/(1 - c) \leq 1/(1 - c) \in \Theta(1)$.

   (b) (3 points) $\Theta(n)$ if $c = 1$.

   Solution: Here, we can't use that formula, because it involves dividing by 0. But each term is 1 and there are $n + 1$ terms, so $g(n) = n + 1 \in \Theta(n)$.

   (c) (3 points) $\Theta(c^n)$ if $c > 1$.

   Solution: The same formula works here, but now both terms are negative, so we'll reverse both to see $g(n) = (c^{n+1} - 1)/(c - 1)$ Since $1/(c - 1)$ is a fixed positive multiplicative constant, it doesn't change the order, and neither does subtracting 1 since c^{n+1} grows . We can simplify $c^{n+1} = cc^n \in \Theta(c^n)$, so $g(n) \in \Theta(c^n)$ in this case.

5. The Fibonacci numbers $F_0, F_1, ...,$ are defined by

$$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$$

(a) (4 points) Use induction to prove that $F_n \geq 2^{0.5n}$ for $n \geq 6$.

(b) (4 points) Use induction to prove that $F_n \leq 2^n$ for $n \geq 0$.

(c) (2 points) What can we conclude about the growth of F_n?

Solution: Instead of solving the stated problem, I'm going to use this framework to give the actual order of the Fibonacci sequence, and show where the golden ratio comes in. The claims above show that F_n is growing exponentially, although we haven't identified the base exactly. To use the same argument to identify the correct base, let's do the proofs in more general terms.

Let's see for which w we can prove $F_n \leq c_1 w^n$ for some c_1 and for which we can prove $F_n \geq c_2 w^n$ for some c_2. In each, we can pick c_1, c_2 to make the inequalities true for the base cases $n = 1, n = 2$. Then we can give a proof by strong induction: Assume that, for $k \geq 2$ and all $1 \leq j \leq k$, $F_j \leq c_1 w^j$. Then we want to conclude $F_{k+1} \leq c_1 w^{k+1}$. $F_{k+1} = F_{k-1} + F_k \leq c_1 w^{k-1} + c_1 w^k = c_1 w^{k+1}(1/w^2 + 1/w)$. So if $1/w^2 + 1/w \leq 1$, the proof goes through. The other direction has the same steps, except then we need $1/w^2 + 1/w \geq 1$. Thus, if we can find a $w > 1$ with $1/w^2 + 1/w = 1$, both the upper and lower bounds will hold, and we'll have identified the actual base. Multiplying through by w^2, the constraint becomes $1 + w = w^2$, which we can solve using the quardratic equation: $w = (1 + \sqrt{5})/2$. $((1 - \sqrt{5})/2$ is also a solution, but is negative.) This number is called "the golden ratio" and occurs both in art and nature.