Before we start the assignment, a quote by the mathematician John Lee:

“It is exactly at that stage of being stuck that the deepest learning occurs. It is all too easy for students to read someone else’s solution and immediately think “Oh, now I understand that,” when in fact they do not understand it nearly as well as they would have if they had struggled through it for themselves.

A much more effective strategy for getting unstuck is to talk the problem over with an instructor or a fellow student. Getting suggestions from other people and turning them into an argument of your own are much more useful than reading someone else’s complete and polished proof.”

**Problem 1.**

Twin primes are pairs of prime numbers that are at distance two from each other. For example, 5 and 7 are twin primes, and so are 11 and 13. The following program looks for twin primes between 2 and an input number $n$. Analyze the running time of the following program. Use $O(\cdot)$-notation like we did in class. The running time should be with respect to $n$.

```python
def isPrime(num):
    for i in range(2, num):
        if num % i == 0:
            return False
    return True

n = int(input("please enter a positive integer: "))
for j in range(2, n):
    j2 = j + 2
    if isPrime(j) and isPrime(j2):
        print("Twin primes: " + str(j) + " and " + str(j2))
```

**Problem 2.**

Consider the equation $a^x + b^x = c^x$. The Pythagorean theorem provides many solutions to this equation when $x = 2$. For example, $3^2 + 4^2 = 5^2$. Fermat’s Last Theorem states that there are no solutions for this equation when all four variables are positive integers and $x > 2$. The following program checks if there are such solutions when $a, b, c, x \leq n$. Analyze the running time of the program using $O(\cdot)$-notation. The running time should be with respect to $n$.

```python
n = int(input("please enter a positive integer: "))
for x in range(3, n+1):
    for a in range(1, n+1):
        for b in range(1, n+1):
            for c in range(1, n+1):
                if a**x + b**x == c**x:
                    print("Fermats Last Theorem is False!")
```
**Problem 3.**

Sort out the following expressions according to their asymptotic size. That is, \( f(n) \) should appear before \( g(n) \) if \( f(n) = O(g(n)) \).

- \( n^{12} + n^{10} \)
- \( n^5 + 2^n \)
- \( 10^{10} \)
- \( n^{11} \cdot 2^{\log_2 n} \)
- \( n^{10} + 3^n \)
- \( n^{12} \cdot \log_2 n \)

**Problem 4.**

Write an algorithm that receives a list of distinct integers (that is, no integer repeats twice) and finds the 10th largest number in the list. Your algorithm should have a running time of \( O(n) \). It can be inefficient in the sense that the constant in the \( O(\cdot) \)-notation may be large.

**Problem 5.**

Let \( L \) be a list of distinct integers. We say that an element of \( L \) is a top if it is larger than the element to its left and larger than the element to its right. For example, the tops in \( L = [1,5,7, -10,30,50,51,12] \) are 7 and 51.

The first element of \( L \) is a top if it is larger than the element to its right. The last element of \( L \) is a top if it is larger than the element to its left.

(a) Let \( L \) be a list of \( n \) distinct integers. Assume that \( n \) is even and larger than two. We claim that if \( L[n/2] < L \left[ \frac{n}{2} - 1 \right] \) then one of the first \( \frac{n}{2} - 1 \) elements of \( L \) is a top. Explain why this claim is true.

(b) Write an algorithm that receives a list of \( n \) distinct integers and returns the index of one top. If there are many tops, the algorithm returns only one of those. The running time should be \( O(\log n) \). Hint 1: Use part (a). Hint 2: think about binary search.