Vitamins (30 minutes)

For **big-O proof**, if there exists constants c, and n_0 such that f(n) ≤ c*g(n) for every n ≥ n_0, then f(n) = O(g(n)).

For **big-Θproof**, if there exists constants c1, c2, and n_0 such that c1*g(n) ≤ f(n) ≤ c2*g(n) for every n ≥ n_0, then f(n) = Θ(g(n)).

1. Use the **formal proof of** O **and** Θ in order to show the following (10 minutes):

a) $n^2 + 5n - 2$ is $O(n^2)$

$n^2 + 5n - 2 < n^2 + 5n^2 + 2n^2 < c * n^2$ for all $n > n_0$

To find c, simply find an upper bound for each value

$n^2 + 5n - 2 < n^2 + 5n^2 + 2n^2$ for all $n > n_0$

$c = 1 + 5 + 2 = 8$

$n^2 + 5n - 2 < 8n^2$ for all $n > 1$

$c = 1 + 5 + 2 = 8$

$n^2 + 5n - 2 < 16n^2$

$c = 8,\ n_0 = 1$

b) $\frac{n^2-1}{n+1}$ is $O(n)$

$\frac{n^2-1}{n+1} < c * n$ for all $n > n_0$

$n - 1 < c * n$ for all $n > n_0$ * (basic algebra)

$n < 2 * n + 1$ for all $n > 1$

$c = 2,\ n_0 = 1$

*this is okay because the zero division issue only occurs when n = -1, n will never be -1 because n represents the number of operators

c) $\sqrt{5n^2 - 3n + 2}$ is $\Theta(n)$

$c * n < \sqrt{5n^2 - 3n + 2} < d * n$ for all $n > n_0$

$c^2 * n^2 < 5n^2 - 3n + 2 < d^2 * n^2$ for all $n > n_0$

(basic algebra; inequalities hold true when squaring guaranteed positive numbers)

$c^2 * n^2 < 5n^2 - 3n + 2 < d^2 * n^2$ for all $n > n_0$

definition of big O $\rightarrow d = \sqrt{10}$, $n_0 = 1$ ($\sqrt{10}$ ~ 3.16)

$5n^2 - 3n + 2 < d^2 * n^2 \rightarrow$
$5n^2 - 3n + 2 < 5n^2 + 3n^2 + 2n^2 \rightarrow 5n^2 - 3n + 2 < 10n^2$
$\sqrt{5n^2 - 3n + 2} < \sqrt{10}\, n$

definition of big Ω $\rightarrow c = 1$, $n_0 = 1$

$c^2 * n^2 < 5n^2 - 3n + 2$
$\rightarrow 5n^2 - 3n^2 - n^2 < 5n^2 - 3n + 2$
$\rightarrow 1n^2 < 5n^2 - 3n + 2$
$\rightarrow 1n < \sqrt{5n^2 - 3n + 2}$

$c = 1, d = \sqrt{10}$, $n_0 = 1$

2. State **True** or **False** and explain why for the following (10 minutes):

a) $8n^2(\sqrt{n})$ is $O(n^3)$ TRUE

$8n^2(\sqrt{n}) < c * n^3$ for all $n > n_0$

$8n^2(\sqrt{n}) < 8n^3$ for all $n > 1$
$8n^{2.5} < 8n^3$ for all $n > 1$

$c = 8,\ n_0 = 1$

b) $8n^2(\sqrt{n})$ is $\Theta(n^3)$ FALSE

$c * n^3 < 8n^2(\sqrt{n}) < d * n^3$ for all $n > n_0$

as n gets infinitely large, we can't find a tight bound

3. For each of the following code snippets, find $f(n)$for which the algorithm's time complexity is $\Theta(f(n))$in its **worst case** run and explain why. (10 minutes)

a)
```
def func(lst):
    for i in range(len(lst)):
        if (len(lst) % 2 == 0):
            return
```

O(n) because the len(lst) can be odd numbers, doesn't change

b)
```
def func(lst):
    for i in range(len(lst)):
        if (lst[i] % 2 == 0):
            print("i =", i)
        else:
            return
```

O(n) because the list can be all even numbers

c)
```
def func(lst):
    for i in range(len(lst)):
        for j in range(len(lst)):
            if (i+j) in lst:
                print("i+j = ", i+j)
```

O(n^3) because the if (i+j) in lst is linear, which is repeated n times, which is repeated n more times, n*n*n; however students should not think that the cost of nested loops is simply multiplied like so.

d)
```
def func(n):
    for i in range(int(n**(0.5))): #sqrt(n)
        for j in range(n): #n
            if (i*j) > n*n:
                print("i*j = ", i*j)
```

O(n*sqrt(n)) or O(n^3/2) because the inner loop runs n times, and the outer loop repeats the inner loop sqrt(n) times. So, n + n + n … + n = n*sqrt(n). Also that inner if statement would never run because i*j can never be > n**n if i < n and j < n for positive numbers.

e)
```
def func(n):
    for i in range(n//2): #n/2
        for j in range(n): #n
            print("i+j = ", i+j)
```

O(n^2) because the inner list is O(n)and is repeated n//2 times so n + n + n + n … + n = n*n//2 = n^2/2 = O(n^2)

Coding (See CS1134 Lab3 Solutions.py)

In this section, it is strongly recommended that you solve the problem on paper before writing code.

1.

a. Given a list of values (`int, float, str, …`), write a function that reverses its order in-place. You are not allowed to create a new list. Your solution must run in Θ(n), where n is the length of the list (10 minutes together in lab).

```
def reverse_list(lst):
    """
    : lst type: list[]
    : return type: None
    """
```

b. Modify the function to include low and high parameters that represent the positive indices to consider. Your function should reverse the list from index low to index high, inclusively. By default, low and high will be None so these parameters are optional. If they're both None (no parameters passed), set low to 0 and high to len(lst) - 1 just like in the previous function above.

You are not allowed to create a new list. Your solution must run in Θ(n), where n is the length of the list (5 minutes).

```
def reverse_list(lst, low = None, high = None):
    """
    : lst type: list[]
    : low, high type: int
    : return type: None
    """
```

Example:

lst = [1, 2, 3, 4, 5, 6], low = 0, high = 5

```
reverse_list(lst) #default, no parameters passed
print(lst) → [6, 5, 4, 3, 2, 1]
```

lst = [1, 2, 3, 4, 5, 6], low = 1, high = 3

```
reverse_list(lst, 1, 3)
```

```
print(lst) → [1, 4, 3, 2, 5, 6]
```

2. Given a **sorted** list of positive integers with zeros mixed in, write a function to move all zeros to the end of the list while maintaining the order of the non-zero numbers. For example, given the list [0, 1, 0, 3, 13, 0], the function will modify the list to become [1, 3, 13, 0, 0, 0]. Your solution must be in-place and run in $\Theta(n)$, where n is the length of the list. (25 minutes)

```
def move_zeros(nums):
    """
    : nums type: list[int]
    : return type: None
    """
```

Hint: You should traverse the list with 2 pointers, both starting from the beginning. One pointer will traverse through the entire list but when should the other pointer move?

3. Recall the following question from Homework 1 Question 2:

```
def shift(lst, k):
      """
      : lst type: list[]
      : k type: int
      : return type: None
      """
```

The function takes in a list and shifts it to the left by k steps.

ex) shift([1, 2, 3, 4, 5, 6], **2**) → [3, 4, 5, 6, **1, 2**]

In the homework, you probably solved it using the list methods, pop() and insert() to shift the list or manually shifted the list each time using an extra loop. Know that your homework solution was not linear if you used either of these methods. Since the run-time of the list methods have not been discussed at this point, do not use any of the methods for this question.

This time, you will attempt to solve this with run-time in mind. That is, your solution must run in $\Theta(n)$, where n is the length of the list (15 minutes).

The direction will also change so that the **shift function will shift to the right instead.**

ex) shift([1, 2, 3, 4, 5, 6], **2**) → [**5, 6,** 1, 2, 3, 4]

Hint: You should use the function reverse_list(lst, low, high) function from part 1b to solve this problem

4. Maximum Sum Subarray:

You are given an integer array `nums` consisting of `n` elements, and an integer `k`.

Find a contiguous subarray whose length is equal to $\frac{n}{k}$ that has the maximum average value and *return this value*.

For example,

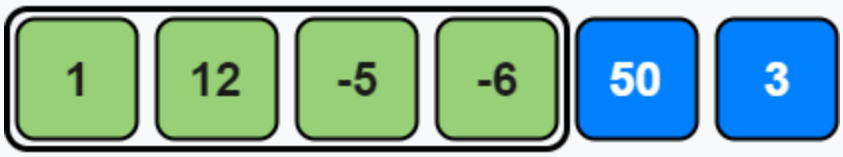
```
Input: nums = [1,12,-5,-6,50,3], k = 2
Output: 47
Explanation: Maximum sum is (-6 + 50 + 3) = 47. The window size is 3
```

since the size is $\frac{n}{k}$, or $\frac{6}{2}$

Hint: Use two pointers with a fixed distance and increase both the start and end points to update your sums

Contiguous Subarray = a smaller array nested within an array

In this array, `[1,12,-5,-6,50,3]`, the array `[1,12,-5,-6]` is a contiguous subarray because each number is adjacent to the next. `[1,12,50]` is not a contiguous subarray.



OPTIONAL

Sort the following 18 functions in an increasing asymptotic order and write <, <=, between each two subsequent functions to indicate if the first is asymptotically less than, asymptotically greater than or asymptotically equivalent to the second function respectively.

For example, if you were to sort: $f_1(n) = n$, $f_2(n) = log(n)$, $f_3(n) = 3n$, $f_4(n) = n^2$, your answer could be $log(n) < (n <= 3n) < n^2$

Hint: Try grouping the functions like so: linear, quadratic, cubic, exponential … etc

$$f_1(n) = n$$

$f_2(n) = 500n$

$f_3(n) = \sqrt{n}$

$f_4(n) = log(\sqrt{n})$

$f_5(n) = \sqrt{log(n)}$

$f_6(n) = 1$

$f_7(n) = 3^n$

$f_8(n) = n \cdot log(n)$

$f_9(n) = \frac{n}{log(n)}$

$f_{10}(n) = 700$

$f_{11}(n) = log(n)$

$f_{12}(n) = \sqrt{9n}$

$f_{13}(n) = 2^n$

$f_{14}(n) = n^2$

$f_{15}(n) = n^3$

$f_{16}(n) = \frac{n}{3}$

$f_{17}(n) = \sqrt{n^3}$

$f_{18}(n) = n!$

$(1 \leq 700) < \sqrt{log(n)} < (log(\sqrt{n}) \leq log(n)) < (\sqrt{n} \leq \sqrt{9n}) < \frac{n}{log(n)} < (\frac{n}{3} \leq n \leq 500n) < nlog(n) < \sqrt{n^3} < n^2 < n^3 < 2^n < 3^n < n!$