- This lab will cover more asymptotic analysis, problem solving, and searching.
- It is assumed that you have reviewed chapter 3 of the textbook. You may want to refer to the text and your lecture notes during the lab as you solve the problems.
- When approaching the problems, think before you code. Doing so is good practice and can help you lay out possible solutions.
- Think of any possible test cases that can potentially cause your solution to fail!
- If you finish early, you may leave early after checking out. If you don't finish by the end of the lab, we recommend you complete it on your own time. Ideally, you should not spend more time than suggested for each problem.
- Your TAs are available to answer questions in the lab, during office hours, and on Edstem.

Vitamins (35 minutes)

For **big-O proof**, show that 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**, show that 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)).

For **big-Ω proof**, show that 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)).

1. State **True** or **False** for the following (5 minutes):

   a) $n^3 \quad = \quad O(nlog(n^n))$

   b) $\sqrt{n} \quad = \quad O(log(n))$

   c) $\sqrt{n} \quad = \quad O(\frac{n}{log(n)})$

   d) $n! \quad = \quad O(100^n)$

2. For each of the following $f(n)$, write out the summation results, and provide a tight bound $\Theta(f(n))$, using the $\Theta\ notation$ (5 minutes).

   Given n numbers:

   $1 + 1 + 1 + 1 + 1 \ldots + 1 =$ ____________ $= \Theta($ ____________ $)$

   $n + n + n + n + n \ldots + n =$ ____________ $= \Theta($ ____________ $)$

   $1 + 2 + 3 + 4 + 5 \ldots + n =$ ____________ $= \Theta($ ____________ $)$

   Given $log(n)$ numbers, where *n* is a power of *2*:

   $1 + 2 + 3 + 4 \ldots + log(n) =$ ____________ $= \Theta($ ____________ $)$

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
```

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

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

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

```
e) 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)
```

4. 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. (15 minutes)

```
a) def func(lst):
         for i in range(len(lst)):
               for j in range(i):
                     print(lst[j], end = " ")
```

```
b) def func(lst):
         for i in range(0, len(lst), 2):
               for j in range(i):
                     print(lst[j], end = " ")
```

```
c) def func(n):
        for i in range(n):
             j = 1
             while j <= 80:
                  print("i = ",i,", j =", j)
                  j *= 2
```

```
d) def func(n):
        for i in range(n):
             j = 1
             while j <= n:
                  print("i = ",i,", j =", j)
                  j *= 2
```

Coding

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

1. Given a string, write a function that returns True, if it is a palindrome, and False, if not. A string is a palindrome if its characters are read the same backwards and forwards.

   For example, `"1racecar1"` is a palindrome but `"1racecar2"` is not. You are not allowed to create a new list or use any str methods. Your solution must run in $\Theta(n)$, where n is the length of the input string. (5 minutes)
   `PATTERN: TWO POINTERS MOVING INWARD -> <-`

   ```
   def is_palindrome(s):
        """
        : s type: str
        : return type: bool
        """
   ```

2. Write a function that takes in a string as input and returns a new string with its vowels reversed. For example, an input of "tandon" would return "tondan". Another example, mousse should return meusso. Your function must run in $\Theta(n)$ and you may assume all strings will only contain lowercase characters. Two lines of code have already been given to you.
   (10 minutes)
   `PATTERN: TWO POINTERS MOVING INWARD -> <-`

   You should use:
   1. The **list constructor** converts the string into a list in linear time.
   2. The **.join() string method**, which is guaranteed to run in linear time.
      The join() method is a string method that can take in a list of string values and returns a string concatenation of the list elements joined by a str separator.
      For example: `",".join(["a", "b", "c"])` will return `"a,b,c"`.
   3. low and high variables as pointers to traverse through the list.

   ```
   def reverse_vowels(input_str):
   """
   : input_str type: string
   : return type: string
   """
   list_str = list(input_str) #list constructor guarantees
   Theta(n)
   # Your code implementation goes here
   ```

```
    return "".join(list_str)
```

3. **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 k that has the maximum sum value and *return this value*.

   Your solution must run in Θ(n), where n is the length of the list. (30 minutes)

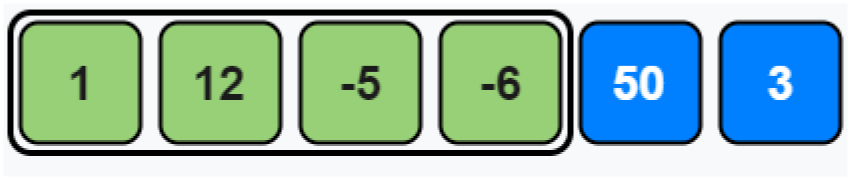
   For example,

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

   Hint: Use two pointers with a fixed distance and increase both the start and end points to update your sums. This is a notion known as a "sliding window".

   Contiguous Subarray = a continuous 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.



4. Complete the following (35 minutes):

   a. The function below takes in a **sorted** list with n numbers, all taken from the range 0 to n, with one of the numbers removed. Also, none of the numbers in the list is repeated. The function searches through the list and returns the missing number. This list contains no duplicate integers.

      For instance, lst = [0, 1, 2, 3, 4, 5, 6, 8] is a list of 8 numbers, from the range 0 to 8, with the number 7 missing. Therefore, the function below will return 7.

Analyze the worst case run-time of the function:

```
def find_missing(lst):
      for num in range(len(lst) + 1):
            if num not in lst:
                  return num
```

b. Rewrite the function so that it finds the missing number with a better run-time: **Hint:** the list is sorted – this algorithm should be in log(n). Also, make sure to consider the edge cases.

```
def find_missing(lst):
      """
      : nums type: list[int] (sorted)
      : return type: int
      """
```

c. Suppose the given list is **not sorted** but still contains all the numbers from 0 to n with one missing.

For instance, lst = [8, 6, 0, 4, 3, 5, 1, 2] is a list of numbers from the range 0 to 8, with the number 7 missing. Therefore, the function below will return 7.

How would you solve this problem? Do not use the idea in step a, or sort the list and reuse your solution in step b. The time complexity of this function should be $O(n)$.

```
def find_missing(lst):
      """
      : nums type: list[int] (unsorted)
      : return type: int
      """
```