- This lab will focus on midterm I review.
- 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 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!
- You must stay for the duration of the lab. If you finish early, you may help other students. If you don't finish by the end of the lab, we recommend you complete it on your own time.
- Your TAs are available to answer questions in lab, during office hours, and on Piazza.

Vitamins (maximum 30 minutes)

---

1. State **True** or **False** for each of the following:

   a) $100log(n^n) + log(n^{0.5})$ is $O(nlog(n))$

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

   c) $2n^{0.5} + 15^n$ is $O(log(n))$

2. 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.

a)
```
def func(n):
    for i in range(n):
        for j in range(i):
            print(i + j)
```

b)
```
def func(lst):
    i = len(lst)
    while (i > 1):
        for j in range(i):
            print(lst[i])
        i //= 2
```

c)
```
def func(lst):
    for i in range(0, int(len(lst)**(0.5))):
        if i not in lst:
            print(i,"not in lst")
```

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

e)
```
def func(lst):
    for i in range(len(lst)):
        for j in range(len(lst)):
            if i*j not in lst:
                print(i*j,"not in lst")
```

Coding

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

1. Write a function to convert a string containing only digits to an int and returns it. Your solution must run in $\Theta(n)$, where n is the number of digits in the string. You may only use the `int( )` to convert a single digit to int such as `"1"` to `1` but not the entire string.

   ex.) `str_to_int("1134")` returns `1134`

```
def str_to_int(int_str):
"""
: int_str type: string
: return type: int
"""
```

2. Write a function that takes in an **unsorted list** of non-negative integers and returns the largest integer number resulting from the product of the 2 numbers. Assume that the list will always have 2 or more elements. Your solution must run in $\Theta(n)$, where n is the number of elements in the list.

   ex.) `max_two_product([11, 3, 4, 9, 2, 7, 8, 10, 5)` returns `110` because 11*10

```
def max_two_product(lst):
"""
: lst type: list[int]
: return type: int
"""
```

For the following recursion problems, think about how you would update the value being returned. You may want to look back at **Lab 6 - Recursion** for your approach in solving these problems.

3. Write a **recursive** function that takes in an input string and returns a list of all characters that are not vowels while maintaining the order the characters appear in. If the input string contains only vowel letter characters, your function should return an empty list. Aim for a linear run-time.

   ex) if the input string is `"Midterm next week :("`, the function should return `["M","d","t","r","m"," ","n","x","t"," ","w","k"," ",":","("]`.

```
def non_vowels(input_str, low, high):
    """
    : input_str type: str
    : low, high type: int
    : return type: lst[str]
    """
```

4. Write a **recursive** function to find the maximum even number in a non-empty, **non-sorted** list of integers. The function parameters, low and high, are int values that are used to determine the range of indices to consider. If there is no even number in the list, return None instead. Aim for a linear run-time.

   ex) if the input list is [13, 9, -16, 3, 4, 2], the function should return 4.

```
def find_max_even(lst, low, high):
    """
    : lst type: list[int]
    : low, high type: int
    : return type: int, None (no even)
    """
```

5. Write a function that takes a string as input and returns a new string with its vowels reversed. For example, an input of “tandon” would return “tondan”. Your function must run in $\Theta(n)$ and you may assume all strings will only contain lowercase characters.

```
def reverse_vowels(input_str):
"""
: input_str type: string
: return type: string
"""

list_str = list(input_str)
#list constructor guarantees Theta(n)
```

Hint: you may want to use:

1. The **list constructor** to convert 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"`.

EXTRA 1: DEFINITION OF BIG O

Use the **definitions of** O **and** Θ in order to show the following:

d) $3n^2 - 12n - 8$ is $O(n^2)$

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

f) $\sqrt{3n^2 - 8n - 2}$ is $\Theta(n)$

EXTRA 2: SQUARE ROOT

Implement a function that calculates an approximation of the square root of a number with two-decimal points accuracy that runs in $\Theta(\sqrt{n})$.

```
def square_root(num):
    """
    : num type: positive int
    : return type: float
    """
```

Implement the square root function that runs in $\Theta(\log(n))$.

EXTRA 3: JUMP SEARCH

6. For this question, you will write a new searching algorithm, *jump search*, that will search for a value in a **sorted list**. With jump search, you will separate your list into n/k groups of k elements each. It then finds the group where the element you are looking for should be in, and makes a linear search in this group only.

   For example, let's say k = 4 for the following list of n = 20 elements:

   [1, 3, 6, 7 , 10, 12, 15, 20 , 22, 24, 29, 33 , 39, 55, 61, 64 , 99, 101, 134, 150]

   Here, we have a list of n/k = 20/4 = 5 groups. If we were to check for val = 15, we would start with the first element (index 0) of the first group and check if we've found our value.

   Since, $1 != 15 \ and \ 1 < 15$, we will jump k = 4 elements and move to 10 (at index 4).

   [1, 3, 6, 7 , 10, 12, 15, 20 , 22, 24, 29, 33 , 39, 55, 61, 64 , 99, 101, 134, 150]

   Since $10 != 15 \ and \ 10 < 15$, we jump another k = 4 steps and move to 22 (at index 8).

   [1, 3, 6, 7 , 10, 12, 15, 20 , 22, 24, 29, 33 , 39, 55, 61, 64 , 99, 101, 134, 150]

   Now, we have $22 != 15 \ and \ 22 > 15$, so we don't need to jump further. Instead, we will hop back k elements because we know our val has to be somewhere between 10 and 22, (index 4 and index 8). We jump back up to k = 4 elements until we find our val = 15.

   [1, 3, 6, 7 , 10, 12, 15, 20 , 22, 24, 29, 33 , 39, 55, 61, 64 , 99, 101, 134, 150]
   ← ← ←

   With the sample list above, we jump from 1 to 10, and 10 to 22. Then we linear search (n/k

elements) back between 22 and 10 and found 15.

Recap:

1. Divide the list into `n/k` groups of `k` elements.
2. Jump `k` steps each time until either `val == lst[i]` or `val < lst[i]`.
3. if `val == lst[i]`, return the index i.
4. if `val < lst[i]`, jump back one step each time for a maximum of `k` steps.
5. If `val` is somewhere in those `k` steps, return the index.
6. Otherwise, return `None`.

6a.

Write a function that performs jump search on a sorted list, given an additional parameter, k. This parameter will let the user divide the list into k groups for the search.

Consider some edge cases such as if n isn't divisible by k (there will be a group with fewer than k elements) or if val > all elements in the list?

Analyze the worst case run-time of your function in terms of n, the size of the list, and k :

```
def jump_search(lst, val, k):
"""
: lst type: list[int]
: val, k type: int
: return type: int (if found), None(if not found)
"""
```

6b.

Let's now optimize our jump search algorithm, by choosing the best value for k with respect to n, so that it would minimize the worst-case runtime of jump search.

Note: k is no longer passed as a parameter.

Analyze the run-time of jump search in terms of n, the length of the list and compare its efficiency to that of linear search and binary search:

```
def jump_search(lst, val):
"""
: lst type: list[int]
```

```
    : val type: int
    : return type: int (if found), None(if not found)
    """
```

Here is a simple test code to verify that your searching algorithm works.

```
#TEST CODE

lst = [-1111, -818, -646, -50, -25, -3, 0, 1, 2, 11, 33, 45, 46, 51,
58, 72, 74, 75, 99, 110, 120, 121, 345, 400, 500, 999, 1000, 1114,
1134, 10010, 500000, 999999]

print("TESTING VALUES IN LIST:\n")

for val in lst:
    if jump_search(lst, val) is None:
        print(val, "FAILED")
    #else:
        #print(val, "PASSED")


#just to make sure you're not stuck in an infinite loop
print("TEST COMPLETED")
```