Grader Output

TakeHomeFinalC1V1

99.99 / 99.99 points earned

8 / 8 autograded cells passed

Graded Cells

Cell 2 (cell-e685174d2d2af687)

Passed | 9.09 / 9.09 points

View feedback

Cell 2 (cell-b076aed5b3a3dbbb)

Passed | 0 / 0 points

View feedback

Cell 4 (cell-c8f90426de75ce5a)

Passed | 27.27 / 27.27 points

View feedback

Cell 4 (cell-3edb602d4e0635b5)

Passed | 0 / 0 points

View feedback

Cell 6 (cell-79929dcc8b79be2e)

Passed | 27.27 / 27.27 points

View feedback

Cell 8 (cell-592ffb401ef72249)

Passed | 9.09 / 9.09 points

View feedback

Cell 10 (cell-786ed556a50ea062)

Passed | 9.09 / 9.09 points

View feedback

Cell 12 (cell-818acc41ac55a5fe)

Passed | 18.18 / 18.18 points

View feedback

Final Exam Instructions

This is a "take home" final exam. You have four problems and **24 hours** to solve them.

- We expect that the problems will cumulatively require 4 - 6 hours of work to finish and the remaining 20 hours will provide you the necessary flexibility.

Format

The problems are algorithmic coding problems that require you to use the concepts you have learned so far to solve problems.

- Each problem has an algorithm design part: you shouldd write pseudocode and work out the worst/expected case running time for your own benefit.
- However, these portions will not be graded.
- Each problem has a coding portion where you will have to code up your solution.
- The coding portion will be graded against automatic tests.

Rules

- The exam is open book and notes.
  - You may access your own notes and any resources available through the coursera class website.
    - You may python language documentation and tutorials online.
- Your code should be efficient: if you have inefficiency the notebook may not pass all the tests in time.
  - If the notebook does not pass all tests within a time limit (set internally by cousera's autograder), you may not receive points.
- Use of external resources other than those sanctioned above is **disallowed**:
  - Asking for help from anyone other than course facilitators is disallowed.
  - Course facilitators will be able to provide clarifications about problems but no extra hints will be provided.
  - Posting problems from this exam on external websites/forums is disallowed.
  - Cutting and pasting code fragments from external sources is prohibited.
  - Searching for solutions online is strictly forbidden.
- Please do not post any part of this exam or solutions online.
- It is highly recommended that you start the exam during a time when a course facilitator is available through email to provide timely clarifications.
- **Do not submit until you are ready**
  - Do not submit the exam until you are fully done.
  - Resubmissions may be permitted but not lead to grade changes.

Useful Tips

- Have you understood the problem completely?
  - Use pencil and paper to work out examples.
  - Ask the facilitator for clarifications, if needed.

        - This will unfortunately take time.
        - Facilitators will not provide hints.
        - It may be helpful to arrange the exam around known office hours.
    - It is helpful to go over each and every problem and have a list of questions for the facilitator.
    - Plan your time carefully: the provided 24 hours should be plenty of time.
    - Get the "easy" problems out of the way.
    - Go through test cases carefully: they can often give you some useful hints.

Useful Python Tips.

Tuples

- Python allows us to have tuples of the form `(x, y)` .
- You can compare tuples using the in-built operators.
    - Tuple comparison in python is lexicographic. For example `(1, 2) < (3, 3)` or `(1,2) < (1, 3)` .

Sorting

- You can use inbuilt sorting functions in python. For example if `lst` is a list, then `sorted(lst)` will sort `lst` in ascending order.
- https://www.w3schools.com/python/ref_func_sorted.asp (https://www.w3schools.com/python/ref_func_sorted.asp)

Enumerated Loops

- You can iterate through list with indices in python using enumerate.
- https://www.geeksforgeeks.org/enumerate-in-python/ (https://www.geeksforgeeks.org/enumerate-in-python/)

Problem 1 (5 points)

Given a Python list/array `a: [a[0], ..., a[n-1]]` of floating point numbers, we would like to find the "sorted rank" of each element. I.e, return an array `r: [ r[0], ..., r[n-1]]` such that `r[i]` is the rank of the element `a[i]` if we were to sort the array `a` in ascending order.

In other words, if `r[i] = j` then the corresponding element `a[i]` would occur in the j^{th} position if the array `a` were to be sorted in the ascending order.

Note that if `r[i] = 0` , this means that `a[i]` is the least element. Likewise, if `r[j]` is `n-1` , then `a[j]` is the maximal element of the array `a` . The original array `a` should not be modified.

You may use inbuilt sorting routine in python or write your own $\Theta(n \log(n))$ time sorting routine.

Example

```
a: [-1, 5, -2, 3, 0, 2]
r: [1, 5, 0, 4, 2, 3 ]
```

Hint

Given an input list:

```
a: [-1, 5, -2, 3, 0, 2]
```

First "mark" each element by its index in the list `a` so that you can note the original position of an element after you sort it.

```
a_marked: [ (-1, 0), (5, 1), (-2, 2), (3, 3), (0, 4), (2, 5) ]
```

What will happen if you try to sort the list/array `a_marked` ?

In [1]:

Student's answer (Top)

```
def getSortedRank(a):
    # Pair each element with its index
    a_marked = [(val, idx) for idx, val in enumerate(a)]

    # Sort the marked list based on element value
    a_marked.sort(key=lambda x: x[0])

    # Create a list to store sorted ranks
    rank = [0] * len(a)

    # Assign ranks based on the sorted order
    for i, (_, original_idx) in enumerate(a_marked):
        rank[original_idx] = i

    return rank

# Test the function
a = [-1, 5, -2, 3, 0, 2]
print(getSortedRank(a))  # Output: [1, 5, 0, 4, 2, 3]
```

In [2]:

Grade cell: `cell-e685174d2d2af687` Score: 9.09 / 9.09 (Top)

```
from random import randint

def testRankArray(a, rank):
    n = len(a)
    sarray = [None]*n
    # Use the result to create a "sorted" array
    for (i, j) in enumerate(rank):
        sarray[j] = a[i]
    assert sum(rank) == (n-1)*(n)/2
    assert sum(sarray) == sum(a)
    # check that the sorted array is really sorted
    elt0 = sarray[0]
    for elt in sarray[1:]:
        assert elt0 <= elt, 'Test failed'
        elt0 = elt
    return

print(' -- Test 1 -- ')
r = getSortedRank([-1, 5, -2, 3, 0, 2])
print(r)
assert r == [1, 5, 0, 4, 2, 3] , 'Test 1 failed'

print(' -- Test 2 --')
a1 =[-1, 6, 7, 8, 2, 3, 2, 1, 0, 5, 4, 2]
r1 = getSortedRank(a1)
print(r1)
testRankArray(a1, r1)

print('--- Random Test 3 ---')

def makeTestArray(n):
    a = [0]*n
    for i in range(n):
        a[i] = randint(-2*n, 2*n)
    return a

a3 = makeTestArray(20)
r3 = getSortedRank(a3)
print(f'a = {a3}')
print(f'result = {r3}')
testRankArray(a3, r3)

print('--- Random Test 4 ---')

a4 = makeTestArray(200)
r4 = getSortedRank(a4)
print('array too long to print')
#print(f'a = {a4}')
#print(f'result = {r4}')
testRankArray(a4, r4)
print('passed')

print('--- Random Test 5 ---')

a5 = makeTestArray(2000)
r5 = getSortedRank(a5)
print('array too long to print')
```

```
#print(f'a = {a5}')
#print(f'result = {r5}')
testRankArray(a5, r5)
print('passed')
print('--- Random Test 6 ---')

a6 = makeTestArray(20000)
r6 = getSortedRank(a6)
print('array too long to print')
#print(f'a = {a6}')
#print(f'result = {r6}')
testRankArray(a6, r6)
print('passed')
print('All tests passed (5 points)')
```

```
Congratulations! All test cases in this cell passed.
```

Problem 2 (15 points)

Given a Python list (array) `a: [ a[0], ..., a[n-1] ]` of $n > 2$ floating point numbers, we would like to find two elements `a[i]` and `a[j]` such that $i < j$ and the absolute value of the difference $|a[i] - a[j]|$ is minimized.

The input to the function a list `a` and output should be a tuple of indices `(i,j)`, wherein

- $0 \leq i < j < n$ and
- the absolute difference $|a[i] - a[j]|$ is the minimized.

If there are multiple pairs (i, j) that produce the same absolute difference your algorithm can output any one of them.

Example:

```
Input a : [ 1, 9, 18,  14, 17, 11]
Output: i,j = (2,4)
```

Note that `a[2] = 18` and `a[4] = 17`. The absolute difference 1 is the minimum possible among all pairs.

First, describe the algorithm briefly in pseudocode or the main idea in words. Write down the complexity.

- Your algorithm must have a complexity $O(n^{2-\epsilon})$. In other words, the trivial $O(n^2)$ algorithm of going through all pairs of indices is not acceptable.
- You may use the solution to the previous problem assuming that it's complexity is $\Theta(n \log(n))$.

Student's answer Score: 0.0 / 0.0 (Top)

YOUR ANSWER HERE

Comments:
No response.

Implement your algorithm: You can use any in-built python list API function, including functions to sort a list in python. https://www.w3schools.com/python/python_ref_list.asp (https://www.w3schools.com/python/python_ref_list.asp)

In [3]: Student's answer (Top)

```python
def findMinAbsDiff(a):
    assert len(a) > 2

    # Enumerate the array to preserve original indices
    a_enum = list(enumerate(a))

    # Sort the array based on values
    a_sorted = sorted(a_enum, key=lambda x: x[1])

    # Initialize variables
    min_diff = float('inf')
    indices = None

    # Iterate through the sorted array to find minimum absolute difference
    for i in range(len(a_sorted) - 1):
        diff = abs(a_sorted[i][1] - a_sorted[i+1][1])
        if diff < min_diff:
            min_diff = diff
            indices = (a_sorted[i][0], a_sorted[i+1][0])

    # Sort the indices to ensure i < j
    indices = tuple(sorted(indices))

    return indices

# Test the function
a = [1, 9, 18, 14, 17, 11]
print(findMinAbsDiff(a))  # Output: (2, 4)
```

In [4]:

Grade cell: cell-c8f90426de75ce5a Score: 27.27 / 27.27 (Top)

```
from random import shuffle
print(' -- Test 1 -- ')
(i, j) = findMinAbsDiff([ 1, 9, 18,  14, 17, 11])
print(f'i={i}, j={j}')
assert i == 2 and j == 4, 'Test 1 failed'
print('passed')
print('-- Test 2 --')
a1 = [1, 5, 9, 11, 2, 15, 25, 12, 18]
(i1, j1) = findMinAbsDiff(a1)
print(f'i={i1}, j={j1}')
assert abs(a1[i1] - a1[j1]) == 1, f'Test 2 failed: Minimmum difference must be 1 your code produces {abs(a1[i1] - a1[j1])}'
print('passed')
print('-- Test 3 --')
a2 = list(range(-10, 10, 3))
a2.append(3)
(i2, j2) = findMinAbsDiff(a2)
print(f'i={i2}, j={j2}')
assert abs(a2[i2] - a2[j2]) == 1, f'Test 2 failed: Minimmum difference must be 1 your code produces {abs(a2[i2] - a2[j2])}'
print('passed')

print('-- Test 4 --')
a3 = list(range(-100, 100, 3))
a3.append(11)
shuffle(a3)
(i3, j3) = findMinAbsDiff(a3)
print(f'i={i3}, j={j3}')
assert abs(a3[i3] - a3[j3]) == 0, f'Test 4 failed: Minimmum difference must be 0 your code produces {abs(a3[i3] - a3[j3])}'
print('passed')



print('-- Test 5 --')
a4 = list(range(-100, 100000, 3))
a4.append(12)
shuffle(a4)
(i4, j4) = findMinAbsDiff(a4)
print(f'i={i4}, j={j4}')
assert abs(a4[i4] - a4[j4]) == 1, f'Test 5 failed: Minimmum difference must be 1 your code produces {abs(a4[i4] - a4[j4])}'
print('passed')

print('All tests passed (15 points)!')
```

```
Congratulations! All test cases in this cell passed.
```

Problem 3 (25 points)

You are given $k > 2$ lists `a1, ..., ak` , each of size n. Each list `ai: [ ai[0], ..., ai[n-1] ]` has `n` numbers. Design an algorithm that finds out all the numbers that are present in all k lists. Your algorithm should run in expected time $\Theta(n \times k)$ and take no more than $\Theta(n)$ extra space.

Describe your algorithm's pseudocode below (not graded).

Student's answer Score: 0.0 / 0.0 (Top)

YOUR ANSWER HERE

Comments:
No response.

Part A (15 points)

Implement a function `returnAllCommonElement(list_of_lists)` that takes in a list of lists `[a1, ..., ak]` where each `ai` is a list of numbers.

You may use a python dictionary or implement your own hashtable. For python dictionaries (typically implemented as open-address hash tables) assume that expected cost of insertion, find is $\Theta(1)$, and cost of iterating through the entire hashtable equals the number of elements in the hashtable.

It returns a list of numbers that are common to all lists.

- There is no requirement that the output list be sorted.
- However, each common element must occur exactly once in the output list.

Example

```
list_of_lists = [ [ 1, 5, 8, -3, 4, 1, 3], [2, 5, 10, -8, 4, 3, 2] ]

expected output = [ 5, 4, 3 ]
```

In [5]:

Student's answer (Top)

```python
def returnAllCommonElements(list_of_lists):
    # Create a dictionary to store counts of elements
    counts = {}

    # Iterate through each list in the list of lists
    for lst in list_of_lists:
        # Create a set for constant-time lookup
        lst_set = set(lst)
        # Update counts dictionary
        for num in lst_set:
            counts[num] = counts.get(num, 0) + 1

    # Initialize a list to store common elements
    common_elements = []

    # Iterate through counts dictionary
    for num, count in counts.items():
        # If count is equal to the number of lists,
        # it means the element is present in all lists
        if count == len(list_of_lists):
            common_elements.append(num)

    return common_elements

# Test the function
list_of_lists = [[1, 5, 8, -3, 4, 1, 3], [2, 5, 10, -8, 4, 3, 2]]
print(returnAllCommonElements(list_of_lists))  # Output: [5, 4, 3]
```

In [6]: Grade cell: cell-79929dcc8b79be2e Score: 27.27 / 27.27 (Top)

```
print(' -- Test 1 --')
list1 = [ [ 1, 5, 8, -3, 4, 1, 3], [2, 5, 10, -8, 4, 3, 2] ]
out1 = returnAllCommonElements(list1)
print(out1)
assert len(out1) == 3
assert 5 in out1
assert 4 in out1
assert 3 in out1
print('passed')
print(' -- Test 2 --')
list2 = [ [1, 3, 5], [5, 4, 7], [8, 1, 5], [-4, 3, 5], [1, 1,
5], [1, 5, 5] ]
out2 = returnAllCommonElements(list2)
print(out2)
assert len(out2)== 1
assert 5 in out2
print('passed')

print('-- Test 3 --')
list3 = [[1, -5, 4, -2, -1], [2, -3, 1, -2, 4, 6, 1, 5, -2], [4,
5, 6, 7, 8, 1, 3, -2]]
out3 = returnAllCommonElements(list3)
print(out3)
assert len(out3) == 3
assert 1 in out3
assert 4 in out3
assert -2 in out3
print('passed')
print('-- Test 4 --')
list4 = [ [1, 2, 4, 7, 2, 6, 3, 6, 7], [8, 9,  3, 4, 8, 9], [1,
4, 56, 12, 67, 8, 0, 18], [0, 1, 7, 8, 0, 1, 3, 5, 6, 0, 19]]
out4= returnAllCommonElements(list4)
print(out4)
assert len(out4) == 0

print('All Tests Passed: 15 points')
```

Congratulations! All test cases in this cell passed.

Part B (5 points)

We will now work on an algorithm that avoids the use of a hashtable (and thus any form of randomization) but assume that the lists `a1,...,` `ak` are already sorted in ascending order.

Write a function `findCommonSorted(lst1, lst2)` that takes in two sorted lists `lst1` and `lst2` and returns a sorted list of all common elements between them.

- Your algorithm must run in time $\Theta(n_1 + n_2)$ where n_1, n_2 are the sizes of the two lists. As a hint, modify the merge algorithm.
- Note that the output list must be sorted in ascending order as well.

In [7]:

Student's answer (Top)

```
def findCommonSorted(list1, list2):
    # Initialize pointers for both lists
    i, j = 0, 0

    # Initialize an empty list to store common elements
    common_elements = []

    # Iterate through both lists
    while i < len(list1) and j < len(list2):
        # If both elements are equal, add to common elements
        if list1[i] == list2[j]:
            common_elements.append(list1[i])
            i += 1
            j += 1
        # If element in list1 is less, move pointer in list1
        elif list1[i] < list2[j]:
            i += 1
        # If element in list2 is less, move pointer in list2
        else:
            j += 1

    return common_elements

# Test the function
list1 = [1, 3, 5, 7, 9]
list2 = [2, 4, 5, 6, 8, 10]
print(findCommonSorted(list1, list2))  # Output: [5]
```

In [8]: Grade cell: `cell-592ffb401ef72249` Score: 9.09 / 9.09 (Top)

```
print('--Test 1--')
list1 = [ -2, 3, 5, 10, 12,  15, 18]
list2 = [-10, -5, -2, 1, 4, 5, 11, 18]
out12 = findCommonSorted(list1, list2)
print(out12)
assert out12 == [-2, 5, 18]
print('passed')
print('--Test 2--')
list3 = [-1, 0, 2, 5, 7, 19, 22, 26, 29, 32, 36]
list4 = [-10, -4, -2, 0, 5, 7, 12, 18, 20, 21, 25, 29, 36]
out34 = findCommonSorted(list3, list4)
print(out34)
assert out34 == [0, 5, 7, 29, 36]
print('passed')
print('--Test 3--')
list5 = list(range(0, 100000,2))
list6 = list(range(1, 100001, 2))
out56 = findCommonSorted(list5, list6)
assert len(out56) ==0
print('passed')

print('All tests passed: 5 points!')
```

```
Congratulations! All test cases in this cell passed.
```

Part C (5 points)

Using the `findCommonSorted` function that finds all common elements in two lists, implement an algorithm for finding common elements in `k` sorted lists : `list_of_lists` but assume that each individual list `aj` is sorted.

In [9]:

Student's answer (Top)

```python
def findCommonSorted(list1, list2):
    # Initialize pointers for both lists
    i, j = 0, 0

    # Initialize an empty list to store common elements
    common_elements = []

    # Iterate through both lists
    while i < len(list1) and j < len(list2):
        # If both elements are equal, add to common elements
        if list1[i] == list2[j]:
            common_elements.append(list1[i])
            i += 1
            j += 1
        # If element in list1 is less, move pointer in list1
        elif list1[i] < list2[j]:
            i += 1
        # If element in list2 is less, move pointer in list2
        else:
            j += 1

    return common_elements

def findAllCommonElementsSorted(list_of_lists):
    assert len(list_of_lists) >= 2

    # Initialize common elements with the first two lists
    common_elements = findCommonSorted(list_of_lists[0], list_of
_lists[1])

    # Iterate through remaining lists
    for lst in list_of_lists[2:]:
        # Find common elements with current list and previous co
mmon elements
        common_elements = findCommonSorted(common_elements, lst)

    return common_elements

# Test the function
list_of_lists = [[1, 3, 5, 7, 9], [2, 4, 5, 6, 8, 10], [3, 5, 7,
9, 11]]
print(findAllCommonElementsSorted(list_of_lists))  # Output: [5]
```

In [10]:

Grade cell: cell-786ed556a50ea062 Score: 9.09 / 9.09 (Top)

```
print(' -- Test 1 --')
list1 = [ [-3, 1, 3, 4, 5, 8], [-8, 2, 2, 3, 4, 5, 10] ]
out1 = findAllCommonElementsSorted(list1)
print(out1)
assert(out1 == [3, 4, 5])
print('passed')
print(' -- Test 2 --')
list2 = [ [1, 3, 5], [4, 5, 7], [1,  5, 8], [-4, 3, 5], [1, 1,
5], [1, 5, 5] ]
out2 = findAllCommonElementsSorted(list2)
print(out2)
assert len(out2)== 1
assert 5 in out2
print('passed')

print('-- Test 3 --')
list3 = [[ -5, -2, -1, 1, 4], [-3, -2, -2,  1, 1, 2, 4, 5, 6],
[-2, 1, 3, 4, 5, 6, 7, 8]]
out3 = findAllCommonElementsSorted(list3)
print(out3)
assert out3 == [-2, 1, 4]
print('passed')
print('-- Test 4 --')
list4 = [ [1, 2, 2, 3, 4,  6,  6, 7,  7], [3, 4, 8, 8, 9, 9],
[0, 1, 4, 8, 12,18,  56, 67], [0, 0, 0, 1, 1, 3, 5, 6,  7, 8]]
out4= findAllCommonElementsSorted(list4)
print(out4)
assert len(out4) == 0
print('All Tests Passed: 5 points')
```

```
Congratulations! All test cases in this cell passed.
```

Problem 4 (15 points)

You are given two lists: `a1, a2`. Each list `ai: [ei1,..., ein]` (i = 1, 2) is a **sorted** list of `n` numbers.

Design an algorithm to find the interval $[l, h]$ such that

1. The width of the interval $h - l$ is as small as possible and
2. Each list has at least one element which lies within the interval $[l, h]$.

Example

```
a1 = [ 1, 4, 8, 9, 14, 15, 18 ]
a2 = [ 5, 10, 19, 23]
```

The smallest possible interval is $[4, 5]$. Note that each array has one element in this interval `a1` has `4`, `a2` has the element `5`. Other possible answers are $[9, 10]$ or $[18, 19]$: both intervals contain at least one element from each list and are also of width 1.

Hint Modify the merge algorithm for two sorted arrays as the starting point for designing your algorithm.

- Merge algorithm maintains an index `i` in `a1` and index `j` in `a2`.
- At each step, if `a1[i] < a2[j]` it increments `i` or else it increments `j`.
- Consider the intervals `[a1[i], a2[j]]` at each step.
    - Figure out how to use these intervals to solve your problem.

In [11]:

Student's answer (Top)

```
def findMinContainingInterval(a1, a2):
    # Assume a1, a2 are sorted
    # Return a tuple (lo, hi) of the interval.
    assert len(a1) > 0
    assert len(a2) > 0

    # Initialize pointers for both lists
    i, j = 0, 0

    # Initialize variables for the interval
    lo, hi = float('-inf'), float('inf')

    # Iterate through both lists
    while i < len(a1) and j < len(a2):
        # Update interval based on current elements
        curr_lo = min(a1[i], a2[j])
        curr_hi = max(a1[i], a2[j])

        # Update the interval to minimize its width
        if curr_hi - curr_lo < hi - lo:
            lo, hi = curr_lo, curr_hi

        # Move pointer in the list with the smaller element
        if a1[i] < a2[j]:
            i += 1
        else:
            j += 1

    return (lo, hi)

# Test the function
a1 = [1, 4, 8, 9, 14, 15, 18]
a2 = [5, 10, 19, 23]
print(findMinContainingInterval(a1, a2))  # Output: (4, 5)
```

In [12]: Grade cell: `cell-818acc41ac55a5fe` Score: 18.18 / 18.18 (Top)

```
from random import randint

def arrayHasEltInInterval(a, l, u):
    assert l <= u
    for elt in a:
        if l <= elt and elt <= u:
            return True
    return False

print('-- Test 1 --')
a1 = [ 1, 4, 8, 9, 14, 15, 18 ]
a2 = [ 5, 10,  19, 23]
(l, u) = findMinContainingInterval(a1, a2)
print(l, u)
assert u -l == 1
assert arrayHasEltInInterval(a1, l, u)
assert arrayHasEltInInterval(a2, l, u)
print('passed')

print('-- Test 2 --')
a1 = [1, 5, 10, 11, 18, 21, 28, 37]
a2 = [ -4, 16, 32, 34]
(l, u) =  findMinContainingInterval(a1, a2)
print(l, u)
assert u - l == 2
assert arrayHasEltInInterval(a1, l, u)
assert arrayHasEltInInterval(a2, l, u)
print('passed')

print('-- Test 3 -- ')
a1 = list(range(0, 100000, 5))
a2 = list(range(257, 1000000, 7))
(l, u) =  findMinContainingInterval(a1, a2)
print(l, u)
assert u - l == 0
assert arrayHasEltInInterval(a1, l, u)
assert arrayHasEltInInterval(a2, l, u)
print('passed')

print('-- Test 4--')
a1 = sorted([ randint(-1000000, 1000000) for i in range(10000
0)])
a2 = sorted([ randint(0, 1000) for i in range(100)])
(l, u) =  findMinContainingInterval(a1, a2)
print(l, u)
assert arrayHasEltInInterval(a1, l, u)
assert arrayHasEltInInterval(a2, l, u)

print('All Tests Passed: 10 points.')
```

```
Congratulations! All test cases in this cell passed.
```

That's all folks!

In [ ]: