Assignment 1 - Binary Search Practice

This assignment is a part of the course "Data Structures and Algorithms in Python".

In this assignment, you'll get to practice some of the concepts and skills covered in the following notebooks:

1. Binary Search and Complexity Analysis
2. Solving Programming Problems Systematically

As you go through this notebook, you will find a **???** in certain places. To complete this assignment, you must replace all the **???** with appropriate values, expressions or statements to ensure that the notebook runs properly end-to-end.

Some things to keep in mind:

- Make sure to run all the code cells, otherwise you may get errors like `NameError` for undefined variables.
- Do not change variable names, delete cells or disturb other existing code. It may cause problems during evaluation.
- In some cases, you may need to add some code cells or new statements before or after the line of code containing the **???**.
- Since you'll be using a temporary online service for code execution, save your work by running `jovian.commit` at regular intervals.
- Questions marked **(Optional)** will not be considered for evaluation, and can be skipped. They are for your learning.

You can make submissions on this page: https://jovian.ai/learn/data-structures-and-algorithms-in-python/assignment/assignment-1-binary-search-practice

If you are stuck, you can ask for help on the community forum: https://jovian.ai/forum/c/data-structures-and-algorithms-in-python/assignment-1/87 . You can get help with errors or ask for hints, but **please don't ask for OR share the full working answer code** on the forum.

How to run the code and save your work

The recommended way to run this notebook is to click the "Run" button at the top of this page, and select "Run on Binder". This will run the notebook on mybinder.org, a free online service for running Jupyter notebooks.

This tutorial is an executable Jupyter notebook. You can *run* this tutorial and experiment with the code examples in a couple of ways: *using free online resources* (recommended) or *on your computer*.

Option 1: Running using free online resources (1-click, recommended)

The easiest way to start executing the code is to click the **Run** button at the top of this page and select **Run on Binder**. You can also select "Run on Colab" or "Run on Kaggle", but you'll need to create an account on Google Colab or Kaggle to use these platforms.

Option 2: Running on your computer locally

To run the code on your computer locally, you'll need to set up Python, download the notebook and install the required libraries. We recommend using the Conda distribution of Python. Click the **Run** button at the top of this

page, select the **Run Locally** option, and follow the instructions.

Saving your work

Before staring the assignment, let's save a snapshot of the assignment to your Jovian profile, so that you can access it later, and continue your work.

```
!pip install jovian --upgrade --quiet
```

```
import jovian
```

```
project='_mansipython-binary-search-assignment'
```

```
jovian.commit(project=project, privacy='secret', environment=None)
```

```
[jovian] Updating notebook "mansishah9865/mansipython-binary-search-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/mansishah9865/mansipython-binary-search-assignment
```

```
'https://jovian.ai/mansishah9865/mansipython-binary-search-assignment'
```

You may be asked to provide an API Key to upload your notebook. The privacy of your assignment notebook is set to "Secret", so that you can the evaluators can access it, but it will not shown on your public profile to other users.

To continue working on a saved assignment, just visit your profile and run the saved notebook again.

Problem - Rotated Lists

We'll solve the following problem step-by-step:

> You are given list of numbers, obtained by rotating a sorted list an unknown number of times. Write a function to determine the minimum number of times the original sorted list was rotated to obtain the given list. Your function should have the worst-case complexity of `O(log N)`, where N is the length of the list. You can assume that all the numbers in the list are unique.
>
> Example: The list `[5, 6, 9, 0, 2, 3, 4]` was obtained by rotating the sorted list `[0, 2, 3, 4, 5, 6, 9]` 3 times.
>
> We define "rotating a list" as removing the last element of the list and adding it before the first element. E.g. rotating the list `[3, 2, 4, 1]` produces `[1, 3, 2, 4]`.
>
> "Sorted list" refers to a list where the elements are arranged in the increasing order e.g. `[1, 3, 5, 7]`.

The Method

Here's the systematic strategy we'll apply for solving problems:

1. State the problem clearly. Identify the input & output formats.

2. Come up with some example inputs & outputs. Try to cover all edge cases.
3. Come up with a correct solution for the problem. State it in plain English.
4. Implement the solution and test it using example inputs. Fix bugs, if any.
5. Analyze the algorithm's complexity and identify inefficiencies, if any.
6. Apply the right technique to overcome the inefficiency. Repeat steps 3 to 6.

This approach is explained in detail in Lesson 1 of the course. Let's apply this approach step-by-step.

Solution

1. State the problem clearly. Identify the input & output formats.

While this problem is stated clearly enough, it's always useful to try and express in your own words, in a way that makes it most clear for you. It's perfectly OK if your description overlaps with the original problem statement to a large extent.

Q: Express the problem in your own words below (to edit this cell, double click on it).

Problem

> An array is given which is rotated and we need to find that how many times it is rotated

Q: The function you write will take one input called `nums`. What does it represent? Give an example.

Input

1. nums: [1,2,3,4] list

Q: The function you write will return a single output called `rotations`. What does it represent? Give an example.

Output

3. `rotations`: 0 Integer

Based on the above, we can now create a signature of our function:

```
def count_rotations(nums):
    Min = 1000
    index = 0
    for i in range(0,len(nums)):
        if nums[i] < Min:
            Min = nums[i]
            index = i
```

```
    return index
```

After each, step remember to save your notebook

```
jovian.commit(project=project)
```

```
[jovian] Updating notebook "mansishah9865/mansipython-binary-search-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/mansishah9865/mansipython-binary-search-assignment
```

```
'https://jovian.ai/mansishah9865/mansipython-binary-search-assignment'
```

2. Come up with some example inputs & outputs. Try to cover all edge cases.

Our function should be able to handle any set of valid inputs we pass into it. Here's a list of some possible variations we might encounter:

1. A list of size 10 rotated 3 times.
2. A list of size 8 rotated 5 times.
3. A list that wasn't rotated at all.
4. A list that was rotated just once.
5. A list that was rotated n-1 times, where n is the size of the list.
6. A list that was rotated n times (do you get back the original list here?)
7. An empty list.
8. A list containing just one element.
9. (can you think of any more?)

We'll express our test cases as dictionaries, to test them easily. Each dictionary will contain 2 keys: `input` (a dictionary itself containing one key for each argument to the function and `output` (the expected result from the function). Here's an example.

```
test = {
    'input': {
        'nums': [19, 25, 29, 3, 5, 6, 7, 9, 11, 14]
    },
    'output': 3
}
```

We can test the function by passing the input to it directly or by using the `evaluate_test_case` function from `jovian` .

```
nums0 = test['input']['nums']
output0 = test['input']['nums']
result0 = count_rotations(nums0)
```

```
result0, result0 == output0
```

```
(3, False)
```

```
from jovian.pythondsa import evaluate_test_case
```

```
evaluate_test_case(count_rotations, test)
```

```
Input:
{'nums': [19, 25, 29, 3, 5, 6, 7, 9, 11, 14]}

Expected Output:
3



Actual Output:
3

Execution Time:
0.006 ms

Test Result:
PASSED


(3, True, 0.006)
```

Let's create one test case for each of the scenarios listed above. We'll store our test cases in an array called `tests`.

Q: Create proper test cases for each of the scenarios listed above.

```
test0 = test
```

```
# A list of size 8 rotated 5 times.
test1 = {
    'input': {
        'nums': [18,13,12,8,5,1,2,3]
    },
    'output': 5
}
```

```
# A list that wasn't rotated at all.
test2 = {
    'input': {
```

```
        'nums': [1,4,6,8,9]
    },
    'output': 0
}
```

A list that was rotated just once. A list that was rotated n-1 times, where n is the size of the list. A list that was rotated n times (do you get back the original list here?) An empty list. A list containing just one element.

```
# A list that was rotated just once.
test3 = {
    'input': {
        'nums': [5,1,2,3,4]
    },
    'output': 1
}
```

```
# A list that was rotated n-1 times, where n is the size of the list.
test4 = {
    'input': {
        'nums': [2,3,4,5,1]
    },
    'output': 4
}
```

```
# A list that was rotated n times, where n is the size of the list
test5 = {
    'input': {
        'nums': [2,4,6,8,9]
    },
    'output': 0
}
```

HINT: Read the question carefully to determine the correct output for the above test case.

```
# An empty list.
test6 = {
    'input': {
        'nums': []
    },
    'output': -1
}
```

```
# A list containing just one element.
test7 = {
    'input': {
        'nums': [1]
    },
    'output': 0
}
```

```
tests = [test0, test1, test2, test3, test3, test5, test6, test7]
```

Q (Optional): Include any further test cases below, for other interesting scenarios you can think of.

```

```

```

```

Evaluate your function against all the test cases together using the `evaluate_test_cases` (plural) function from `jovian` .

```
from jovian.pythondsa import evaluate_test_cases
```

```
evaluate_test_cases(count_rotations, tests)
```

```
TEST CASE #0

Input:
{'nums': [19, 25, 29, 3, 5, 6, 7, 9, 11, 14]}

Expected Output:
3


Actual Output:
3

Execution Time:
0.007 ms

Test Result:
PASSED


TEST CASE #1

Input:
{'nums': [18, 13, 12, 8, 5, 1, 2, 3]}

Expected Output:
5
```

Actual Output:
5

Execution Time:
0.005 ms

Test Result:
PASSED

TEST CASE #2

Input:
{'nums': [1, 4, 6, 8, 9]}

Expected Output:
0

Actual Output:
0

Execution Time:
0.004 ms

Test Result:
PASSED

TEST CASE #3

Input:
{'nums': [5, 1, 2, 3, 4]}

Expected Output:
1

Actual Output:
1

Execution Time:

0.006 ms

Test Result:
PASSED

TEST CASE #4

Input:
{'nums': [5, 1, 2, 3, 4]}

Expected Output:
1

Actual Output:
1

Execution Time:
0.004 ms

Test Result:
PASSED

TEST CASE #5

Input:
{'nums': [2, 4, 6, 8, 9]}

Expected Output:
0

Actual Output:
0

Execution Time:
0.007 ms

Test Result:
PASSED

TEST CASE #6

Input:
{'nums': []}

Expected Output:
-1

Actual Output:
0

Execution Time:
0.004 ms

Test Result:
FAILED

TEST CASE #7

Input:
{'nums': [1]}

Expected Output:
0

Actual Output:
0

Execution Time:
0.003 ms

Test Result:
PASSED

SUMMARY

TOTAL: 8, PASSED: 7, FAILED: 1

[(3, True, 0.007),

```
 (5, True, 0.005),
 (0, True, 0.004),
 (1, True, 0.006),
 (1, True, 0.004),
 (0, True, 0.007),
 (0, False, 0.004),
 (0, True, 0.003)]
```

Verify that all the test cases were evaluated. We expect them all to fail, since we haven't implemented the function yet.

Let's save our work before continuing.

```
jovian.commit(project=project)
```

```
[jovian] Updating notebook "mansishah9865/mansipython-binary-search-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/mansishah9865/mansipython-binary-search-assignment
```

```
'https://jovian.ai/mansishah9865/mansipython-binary-search-assignment'
```

3. Come up with a correct solution for the problem. State it in plain English.

Our first goal should always be to come up with a *correct* solution to the problem, which may not necessarily be the most *efficient* solution. Try to think of a solution before you read further.

Coming up with the correct solution is quite easy, and it's based on this insight: If a list of sorted numbers is rotated `k` times, then the smallest number in the list ends up at position `k` (counting from 0). Further, it is the only number in the list which is smaller than the number before it. Thus, we simply need to **check for each number in the list whether it is smaller than the number that comes before it** (if there is a number before it). Then, our answer i.e. the number of rotations is simply the position of this number is . If we cannot find such a number, then the list wasn't rotated at all.

Example: In the list `[19, 25, 29, 3, 5, 6, 7, 9, 11, 14]`, the number `3` is the only number smaller than its predecessor. It occurs at the position `3` (counting from `0`), hence the array was rotated `3` times.

We can use the *linear search* algorithm as a first attempt to solve this problem i.e. we can perform the check for every position one by one. But first, try describing the above solution in your own words, that make it clear to you.

Q (Optional): Describe the linear search solution explained above problem in your own words.

1. **???**
2. **???**
3. **???**
4. **???**

(add more steps if required)

Let's save and upload our work before continuing.

```
import jovian
```

```
jovian.commit(project=project)
```

```
[jovian] Updating notebook "mansishah9865/mansipython-binary-search-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/mansishah9865/mansipython-binary-search-assignment
```

```
'https://jovian.ai/mansishah9865/mansipython-binary-search-assignment'
```

4. Implement the solution and test it using example inputs. Fix bugs, if any.

Q: Implement the solution described in step 3.

```
def count_rotations_linear(nums):
    position = 0                # What is the intial value of position?

    for i in range(1,len(nums)-1):                    # When should the loop be termir

        # Success criteria: check whether the number at the current position is smaller
        if nums[i] < nums[i-1] and nums[i] < nums[i+1]:
            position = i
            break # How to perform the check?
    if len(nums) == 0:
        return -1
    return position                  # What if none of the positions passed the check
```

Let's test out the function with the first test case.

```
linear_search_result = evaluate_test_case(count_rotations_linear, test)
```

```
Input:
{'nums': [19, 25, 29, 3, 5, 6, 7, 9, 11, 14]}

Expected Output:
3



Actual Output:
3

Execution Time:
0.007 ms

Test Result:
```

```
PASSED
```

Make sure your function passes the test. Fix bugs, if any.

Let's test it out with all the test cases.

```
linear_search_results = evaluate_test_cases(count_rotations_linear, tests)
```

```
TEST CASE #0

Input:
{'nums': [19, 25, 29, 3, 5, 6, 7, 9, 11, 14]}

Expected Output:
3


Actual Output:
3

Execution Time:
0.007 ms

Test Result:
PASSED


TEST CASE #1

Input:
{'nums': [18, 13, 12, 8, 5, 1, 2, 3]}

Expected Output:
5


Actual Output:
5

Execution Time:
0.004 ms

Test Result:
```

PASSED

TEST CASE #2

Input:
{'nums': [1, 4, 6, 8, 9]}

Expected Output:
0

Actual Output:
0

Execution Time:
0.004 ms

Test Result:
PASSED

TEST CASE #3

Input:
{'nums': [5, 1, 2, 3, 4]}

Expected Output:
1

Actual Output:
1

Execution Time:
0.003 ms

Test Result:
PASSED

TEST CASE #4

Input:
{'nums': [5, 1, 2, 3, 4]}

Expected Output:
1

Actual Output:
1

Execution Time:
0.003 ms

Test Result:
PASSED

TEST CASE #5

Input:
{'nums': [2, 4, 6, 8, 9]}

Expected Output:
0

Actual Output:
0

Execution Time:
0.003 ms

Test Result:
PASSED

TEST CASE #6

Input:
{'nums': []}

Expected Output:
-1

```
Actual Output:
-1

Execution Time:
0.003 ms

Test Result:
PASSED


TEST CASE #7

Input:
{'nums': [1]}

Expected Output:
0


Actual Output:
0

Execution Time:
0.003 ms

Test Result:
PASSED


SUMMARY

TOTAL: 8, PASSED: 8, FAILED: 0
```

Once again, make sure all the tests pass. Fix errors and bugs, if any.

NOTE: During evaluation, your submission will be tested against a much larger set of test cases (not listed here). Make sure to test your solution thoroughly.

If you are stuck, you can ask for help on the community forum: https://jovian.ai/forum/c/data-structures-and-algorithms-in-python/assignment-1/87 . You can get help with errors or ask for hints, but **please don't ask for OR share the full working answer code** on the forum.

5. Analyze the algorithm's complexity and identify inefficiencies, if any.

Count the maximum number of iterations it may take for the algorithm to return the result.

Q: What is the worst-case complexity (running time) of the algorithm expressed in the Big O Notation? Assume that the size of the list is `N` (uppercase).

```
linear_search_complexity = "???"
```

6. Apply the right technique to overcome the inefficiency. Repeat steps 3 to 6.

As you might have guessed, we can apply *Binary Search* to solve this problem. The key question we need to answer in binary search is: Given the middle element, how to decide if it is the answer (smallest number), or whether the answer lies to the left or right of it.

If the middle element is smaller than its predecessor, then it is the answer. However, if it isn't, this check is not sufficient to determine whether the answer lies to the left or the right of it. Consider the following examples.

`[7, 8, 1, 3, 4, 5, 6]` (answer lies to the left of the middle element)

`[1, 2, 3, 4, 5, -1, 0]` (answer lies to the right of the middle element)

Here's a check that will help us determine if the answer lies to the left or the right: *If the middle element of the list is smaller than the last element of the range, then the answer lies to the left of it. Otherwise, the answer lies to the right.*

Do you see why this strategy works?

7. Come up with a correct solution for the problem. State it in plain English.

Before we implement the solution, it's useful to describe it in a way that makes most sense to you. In a coding interview, you will almost certainly be asked to describe your approach before you start writing code.

Q (Optional): Describe the binary search solution explained above problem in your own words.

1. **???**
2. **???**
3. **???**
4. **???**

(add more steps if required)

Let's save and upload our work before continuing.

```
jovian.commit(project=project)
```

```
[jovian] Updating notebook "mansishah9865/mansipython-binary-search-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/mansishah9865/mansipython-binary-search-assignment
```

```
'https://jovian.ai/mansishah9865/mansipython-binary-search-assignment'
```

8. Implement the solution and test it using example inputs. Fix bugs, if any.

Q: Implement the binary search solution described in step 7.

If you are stuck, you can ask for help on the community forum: https://jovian.ai/forum/c/data-structures-and-algorithms-in-python/assignment-1/87 . You can get help with errors or ask for hints, but **please don't ask for OR share the full working answer code** on the forum.

```
def count_rotations_binary(nums):
    lo = 0
    hi = len(nums) - 1
    if len(nums) == 1:
        return 0

    while lo <= hi:
        mid = (lo+hi)//2
        mid_number = nums[mid]

        # Uncomment the next line for logging the values and fixing errors.
        print("lo:", lo, ", hi:", hi, ", mid:", mid, ", mid_number:", mid_number)

        if mid >=0 and mid_number < nums[mid-1] and mid_number < nums[mid+1]:
            # The middle position is the answer
            return mid

        elif mid_number < nums[len(nums)-1] :
            # Answer lies in the left half
            hi = mid - 1

        else:
            # Answer lies in the right half
            lo = mid + 1

    return -1
```

Let's test out the function with the first test case.

```
binary_search_result = evaluate_test_case(count_rotations_binary, test)
```

```
Input:
{'nums': [19, 25, 29, 3, 5, 6, 7, 9, 11, 14]}

Expected Output:
3

lo: 0 , hi: 9 , mid: 4 , mid_number: 5
lo: 0 , hi: 3 , mid: 1 , mid_number: 25
lo: 2 , hi: 3 , mid: 2 , mid_number: 29
lo: 3 , hi: 3 , mid: 3 , mid_number: 3
```

```
Actual Output:
3

Execution Time:
0.878 ms

Test Result:
PASSED
```

Make sure your function passes the test. Fix bugs, if any.

Let's test it out with all the test cases.

```
binary_search_results = evaluate_test_cases(count_rotations_binary, tests)
```

```
TEST CASE #0
lo: 0 , hi: 9 , mid: 4 , mid_number: 5
lo: 0 , hi: 3 , mid: 1 , mid_number: 25
lo: 2 , hi: 3 , mid: 2 , mid_number: 29
lo: 3 , hi: 3 , mid: 3 , mid_number: 3

Input:
{'nums': [19, 25, 29, 3, 5, 6, 7, 9, 11, 14]}

Expected Output:
3


Actual Output:
3

Execution Time:
2.208 ms

Test Result:
PASSED


TEST CASE #1
lo: 0 , hi: 7 , mid: 3 , mid_number: 8
lo: 4 , hi: 7 , mid: 5 , mid_number: 1

Input:
```

{'nums': [18, 13, 12, 8, 5, 1, 2, 3]}

Expected Output:
5

Actual Output:
5

Execution Time:
0.158 ms

Test Result:
PASSED

TEST CASE #2
lo: 0 , hi: 4 , mid: 2 , mid_number: 6
lo: 0 , hi: 1 , mid: 0 , mid_number: 1

Input:
{'nums': [1, 4, 6, 8, 9]}

Expected Output:
0

Actual Output:
0

Execution Time:
0.21 ms

Test Result:
PASSED

TEST CASE #3
lo: 0 , hi: 4 , mid: 2 , mid_number: 2
lo: 0 , hi: 1 , mid: 0 , mid_number: 5
lo: 1 , hi: 1 , mid: 1 , mid_number: 1

Input:

{'nums': [5, 1, 2, 3, 4]}

Expected Output:
1

Actual Output:
1

Execution Time:
0.152 ms

Test Result:
PASSED

TEST CASE #4
lo: 0 , hi: 4 , mid: 2 , mid_number: 2
lo: 0 , hi: 1 , mid: 0 , mid_number: 5
lo: 1 , hi: 1 , mid: 1 , mid_number: 1

Input:
{'nums': [5, 1, 2, 3, 4]}

Expected Output:
1

Actual Output:
1

Execution Time:
0.189 ms

Test Result:
PASSED

TEST CASE #5
lo: 0 , hi: 4 , mid: 2 , mid_number: 6
lo: 0 , hi: 1 , mid: 0 , mid_number: 2

Input:

{'nums': [2, 4, 6, 8, 9]}

Expected Output:
0

Actual Output:
0

Execution Time:
0.103 ms

Test Result:
PASSED

TEST CASE #6

Input:
{'nums': []}

Expected Output:
-1

Actual Output:
-1

Execution Time:
0.002 ms

Test Result:
PASSED

TEST CASE #7

Input:
{'nums': [1]}

Expected Output:
0

```
Actual Output:
0

Execution Time:
0.001 ms

Test Result:
PASSED


SUMMARY

TOTAL: 8, PASSED: 8, FAILED: 0
```

Once again, make sure all the tests pass. Fix errors and bugs, if any.

NOTE: During evaluation, your submission will be tested against a much larger set of test cases (not listed here). Make sure to test your solution thoroughly.

If you are stuck, you can ask for help on the community forum: https://jovian.ai/forum/c/data-structures-and-algorithms-in-python/assignment-1/87 . You can get help with errors or ask for hints, but **please don't ask for OR share the full working answer code** on the forum.

Let's save our work before continuing.

```
jovian.commit(project=project)
```

```
[jovian] Updating notebook "mansishah9865/mansipython-binary-search-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/mansishah9865/mansipython-binary-search-assignment
```

```
'https://jovian.ai/mansishah9865/mansipython-binary-search-assignment'
```

9. Analyze the algorithm's complexity and identify inefficiencies, if any.

Q: What is the worst-case complexity (running time) of the algorithm expressed in the Big O Notation? Assume that the size of the list is `N` (uppercase).

Hint: Count the maximum number of iterations it may take for the algorithm to return the result.

```
binary_search_complexity = "???"
```

Is binary search the optimal solution to the problem? How can you prove it? Discuss in the forums.

Let's save our work before continuing.

```
import jovian
```

```
jovian.commit()
```

```
[jovian] Updating notebook "mansishah9865/mansipython-binary-search-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/mansishah9865/mansipython-binary-search-assignment
```

```
'https://jovian.ai/mansishah9865/mansipython-binary-search-assignment'
```

Make a Submission

To make a submission, visit the assignment page and submit the link to your notebook.

You can also make a submission by executing the following statement:

```
jovian.submit(assignment='python-binary-search-assignment')
```

You can view your previous submissions under the "Submission History" section of the assignment page. Only your last sumission will be considered for evaluation.

Bonus Questions

The questions in this section are optional, and will not affect your grade. Discuss the bonus questions here: https://jovian.ai/forum/t/optional-bonus-questions-discussion-assignment-1/15486

You can also copy over the bonus questions to a new notebook to share your solution on the forum without sharing your assignment notebook. Duplicate this template: https://jovian.ai/aakashns/python-problem-solving-template

Optional Bonus 1: Using the Generic Binary Search Algorithm

The `jovian` library provides a generic implementation of the binary search algorithm.

```
from jovian.pythondsa import binary_search
```

You can view it's source code using the `??` command in Jupyter or on the Github repository for the `jovian` library.

```
??binary_search
```

Q (Optional): Implement the `count_rotations` function using the generic `binary_search` function.

Hint: You'll need to define the condition which returns `"found"`, `"left"` or `"right"` by performing the appropriate check on the middle position in the range.

```
def count_rotations_generic(nums):
    def condition(mid):
        pass # replace this with your code
```

```
    return binary_search(0, len(nums)-1, condition)
```

```
evaluate_test_case(count_rotations_generic, test)
```

```
evaluate_test_cases(count_rotations_generic, test)
```

```
jovian.commit()
```

Discuss your solution on the forum: https://jovian.ai/forum/c/data-structures-and-algorithms-in-python/assignment-1/87

Optional Bonus 2: Handling repeating numbers

So far we've assumed that the numbers in the list are unique. What if the numbers can repeat? E.g. `[5, 6, 6, 9, 9, 9, 0, 0, 2, 3, 3, 3, 3, 4, 4]` . Can you modify your solution to handle this special case?

Q (Optional): Create additional test cases where the list can contain repeating numbers

```
extended_tests = list(tests)
```

```
extended_test.append({
    # add your test case here
})
```

```
extended_test.append({
    # add your test case here
})
```

```
# add more test cases if required
```

```
jovian.commit()
```

Q (Optional): Modify your solution (if required) to handle the case where the list can contain repeating numbers.

```
def count_rotations_generic(nums):
    pass # replace this with your code
```

Test your function to make sure it works properly.

```

```

```

```

Discuss your solution on the forum: https://jovian.ai/forum/c/data-structures-and-algorithms-in-python/assignment-1/87

Optional Bonus 3: Searching in a Rotated List

Here's a slightly advanced extension to this problem:

> You are given list of numbers, obtained by rotating a sorted list an unknown number of times. You are also given a target number. Write a function to find the position of the target number within the rotated list. You can assume that all the numbers in the list are unique.
>
> Example: In the rotated sorted list `[5, 6, 9, 0, 2, 3, 4]`, the target number `2` occurs at position `5`.

Q (Optional): Create some test cases for the above problem.

```
tests2 = []
```

```
# add test cases here
```

```

```

Q (Optional): Implement a solution to the above problem using binary search.

HINT: One way to solve this problem is to identify two sorted subarrays within the given array (using the `count_rotations_binary` function defined above), then perform a binary search on each subarray to determine the position of the target element. Another way is to modify `count_rotations_binary` to solve the problem directly.

```
def find_element(nums, target):
    pass
```

Test your solution using the cells below.

```

```

You can test your solution to the above problem here: https://leetcode.com/problems/search-in-rotated-sorted-array/

Discuss your approach on the forum: https://jovian.ai/forum/c/data-structures-and-algorithms-in-python/assignment-1/87

```
jovian.commit()
```