Python Algorithm

Linear Search Example

Problem -1:

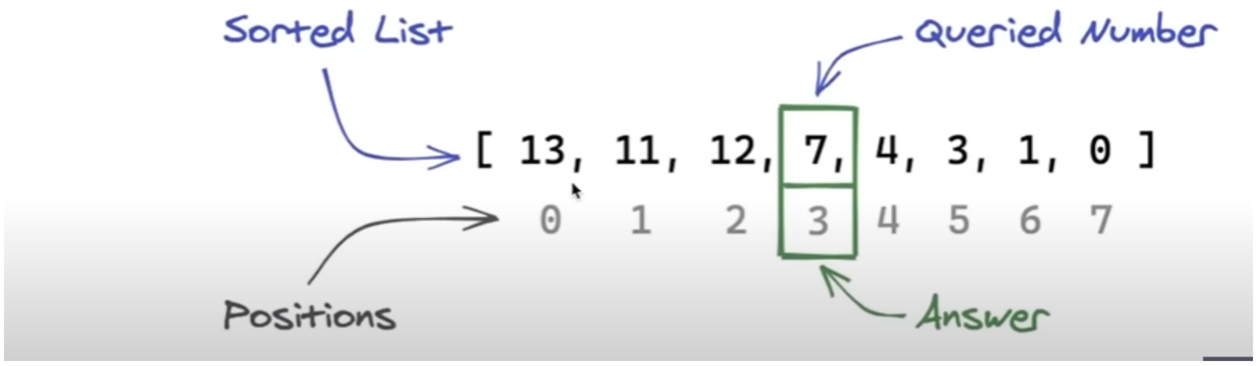
- Alice has some cards with numbers written on them. She arranges the cards in decreasing order, and lays them out face down in a sequence on a table. She challenges Bob to pick the card containing a given number by turning over as few as possible. Write a function to help Bob locate the card.

The Method:

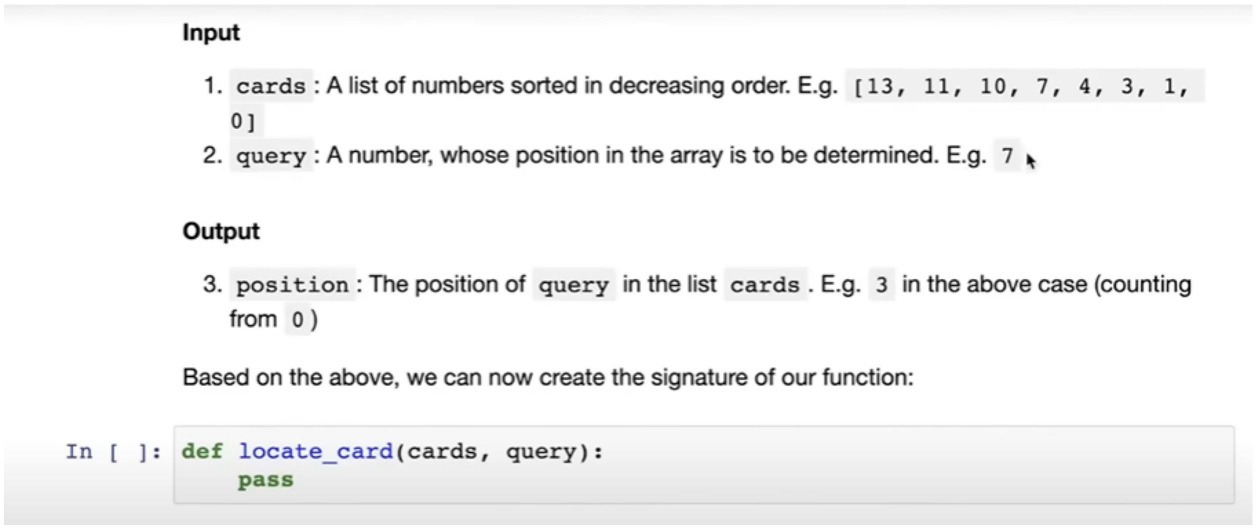
1. State the problem clearly. Identify the input & output formats.
2. Come up with some example inputs & outputs. Try to cover all edge cases.
3. come up with a correct solution for the problem. State it in plain English.
4. Implements the solution and test using the example inputs. Fix bugs, if any.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

Solution:

- In above problem, we can represent the sequence of cards as a list of numbers. Turning over a specific card is equivalent to accessing the value of the number at the corresponding position the list.



- We need to write a program to find the position of a given number in a list of number arranged in decreasing order. We also need to minimize the number of time we access elements from the list.



Input

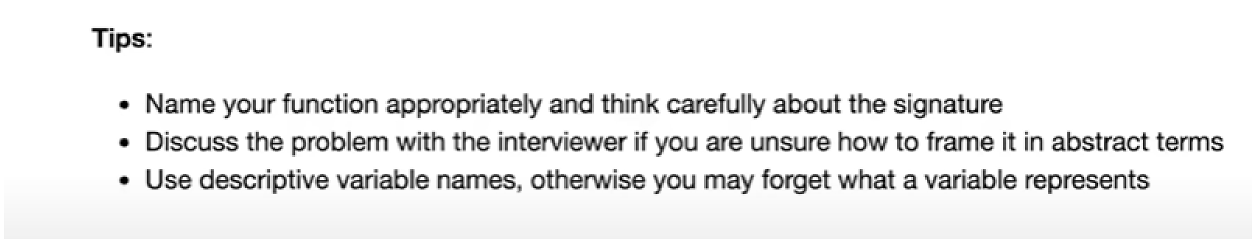
1. `cards` : A list of numbers sorted in decreasing order. E.g. `[13, 11, 10, 7, 4, 3, 1, 0]`
2. `query` : A number, whose position in the array is to be determined. E.g. `7`

Output

3. `position` : The position of `query` in the list `cards` . E.g. `3` in the above case (counting from `0`)

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

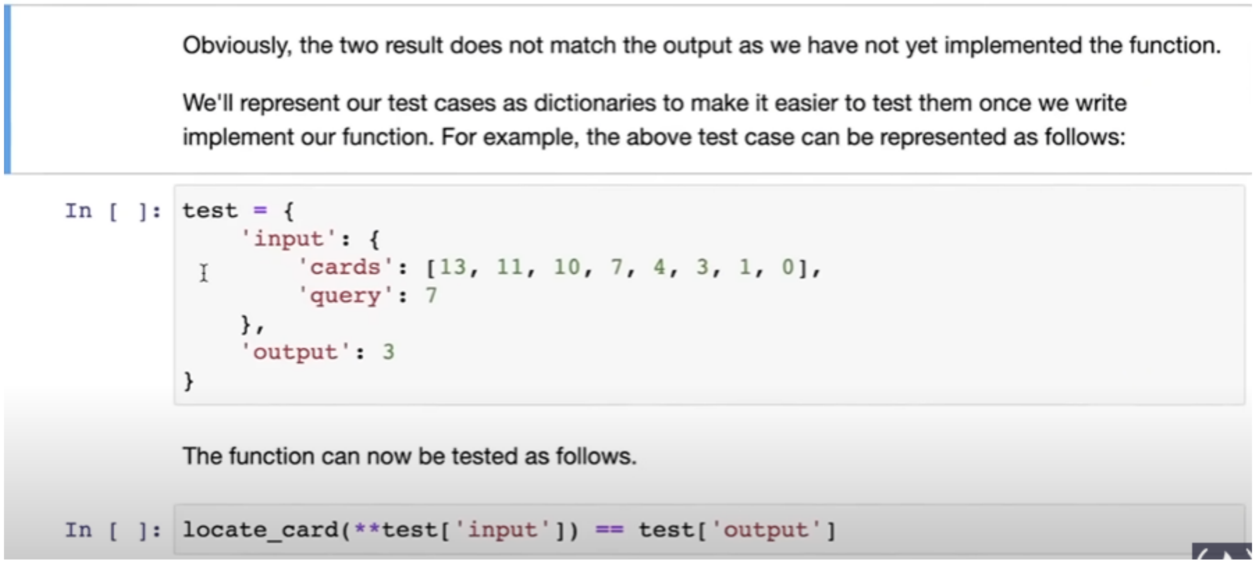
```
In [ ]: def locate_card(cards, query):
            pass
```



Tips:

- Name your function appropriately and think carefully about the signature
- Discuss the problem with the interviewer if you are unsure how to frame it in abstract terms
- Use descriptive variable names, otherwise you may forget what a variable represents

Implement the Test cases to check the desired output is obtained or not in comparision to input.



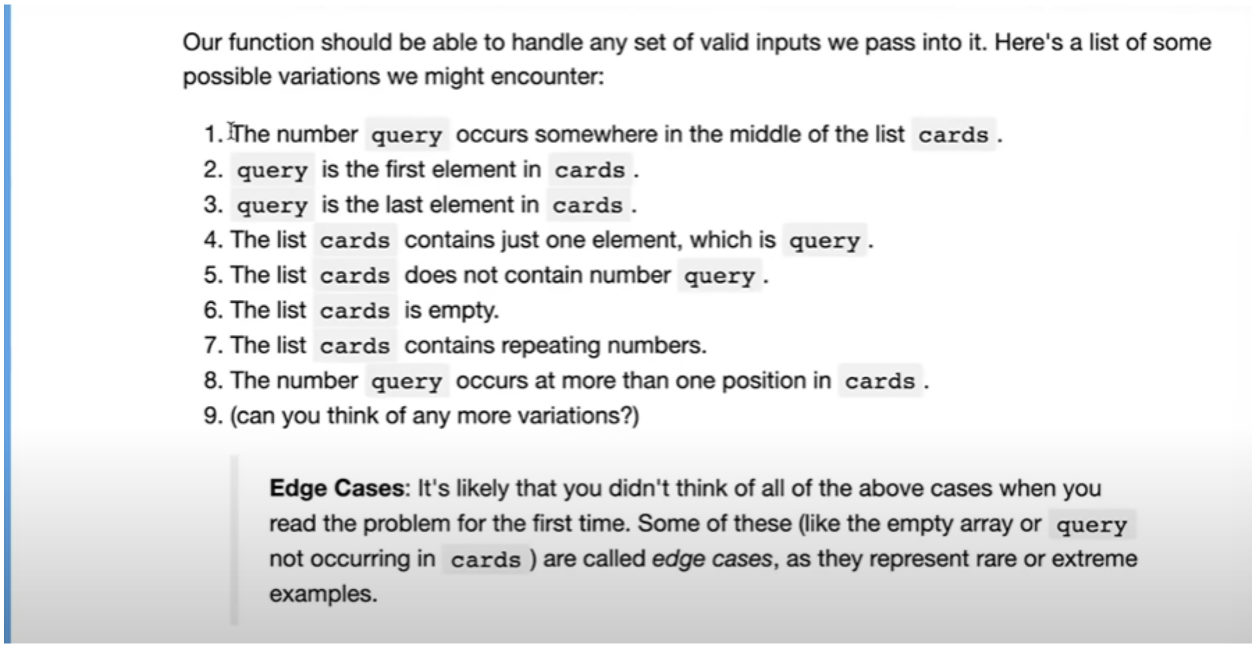
Obviously, the two result does not match the output as we have not yet implemented the function.

We'll represent our test cases as dictionaries to make it easier to test them once we write implement our function. For example, the above test case can be represented as follows:

```
In [ ]: test = {
            'input': {
                'cards': [13, 11, 10, 7, 4, 3, 1, 0],
                'query': 7
            },
            'output': 3
        }
```

The function can now be tested as follows.

```
In [ ]: locate_card(**test['input']) == test['output']
```



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. The number `query` occurs somewhere in the middle of the list `cards`.
2. `query` is the first element in `cards`.
3. `query` is the last element in `cards`.
4. The list `cards` contains just one element, which is `query`.
5. The list `cards` does not contain number `query`.
6. The list `cards` is empty.
7. The list `cards` contains repeating numbers.
8. The number `query` occurs at more than one position in `cards`.
9. (can you think of any more variations?)

> **Edge Cases**: It's likely that you didn't think of all of the above cases when you read the problem for the first time. Some of these (like the empty array or `query` not occurring in `cards`) are called *edge cases*, as they represent rare or extreme examples.

```
tests = [
  {
  'input': {
    'cards': [13,11,10,7,4,3,2,0],
    'query': 7
  },
  'output': 3
  },
  {
    'input': {
    'cards': [4,2,1,-1],
    'query': 4
  },
  'output': 0
  },
  {
    'input': {
    'cards': [8,8,8,6,6,6,3,2,1],
    'query': 6
  },
  'output': 4
  }
]


def locate_card(cards_list,search_card_query):
  location = 0

  for i in range(len(cards_list)):
    if cards_list[i] == search_card_query:
      return (location)
    location  = location +1

for test in tests:
  cards_list = test['input']['cards']
  search_card_query = test['input']['query']
  output = test['output']

  output_of_search = locate_card(cards_list,search_card_query)
  print(output_of_search)

  if (output_of_search == output):
    print("Searched Passed")
  else:
    print("Searched Failed")
```

```
3
Searched Passed
0
Searched Passed
3
Searched Failed
```

TEST CASE #0

Input:

{'cards': [13, 11, 10, 7, 4, 3, 2, 0], 'query': 7}

Expected Output:
3

Actual Output:
3

Execution Time:
0.016 ms

Test Result:
PASSED

TEST CASE #1

Input:
{'cards': [4, 2, 1, -1], 'query': 4}

Expected Output:
0

Actual Output:
0

Execution Time:
0.008 ms

Test Result:
PASSED

TEST CASE #2

Input:
{'cards': [8, 8, 8, 6, 6, 6, 3, 2, 1], 'query': 6}

Expected Output:
4

Actual Output:
3

Execution Time:
0.009 ms

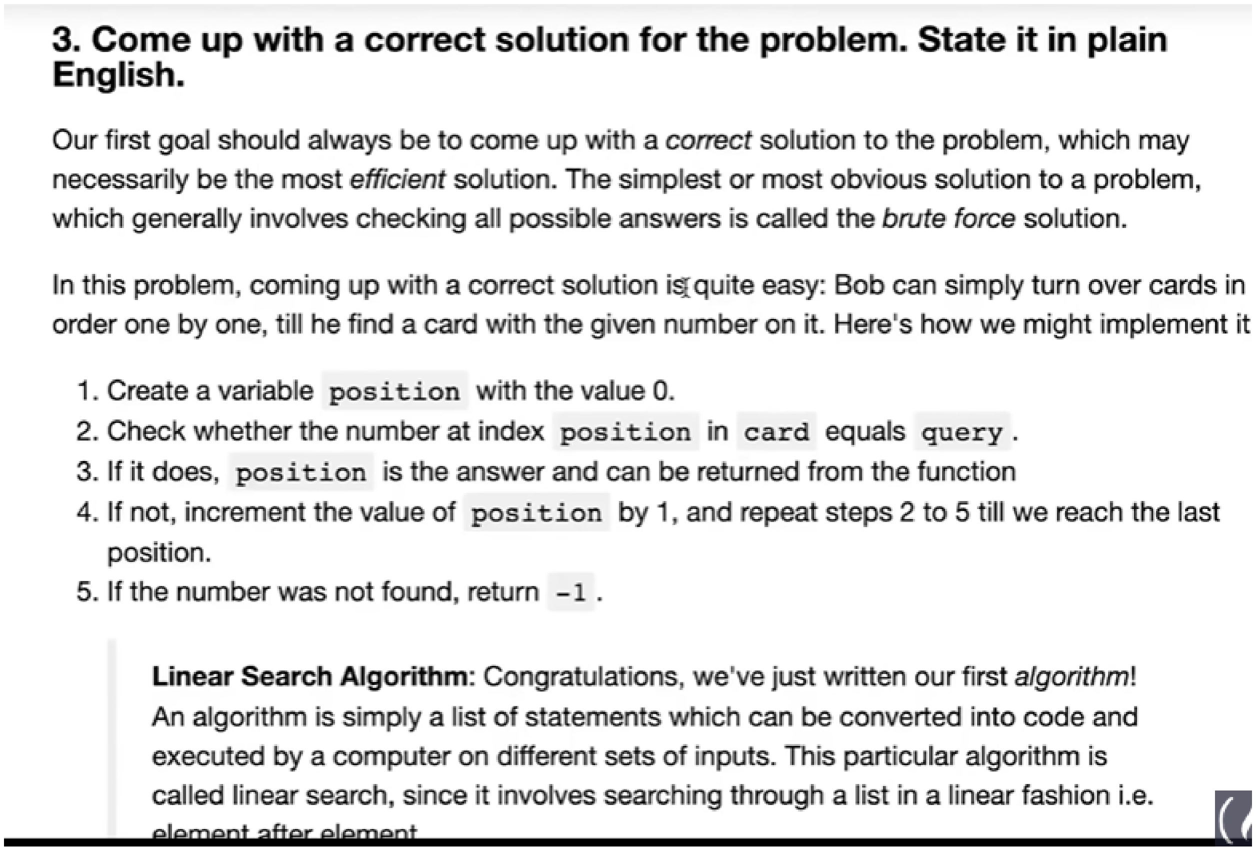
Test Result:
FAILED

SUMMARY

TOTAL: 3, PASSED: 2, FAILED: 1

```
tests  = [
  {
  'input': {
    'cards': [13,11,10,7,4,3,2,0],
    'query': 7
  },
  'output': 3
  },
  {
    'input': {
    'cards': [4,2,1,-1],
    'query': 4
  },
  'output': 0
  },
  {
    'input': {
    'cards': [8,8,8,6,6,6,3,2,1],
    'query': 6
  },
  'output': 4
  }
]

tests.append({

})

cards=  tests['input']['cards']
print(cards)
query = tests['input']['query']
print(query)
output = tests['output']
print(output)

def locate_position(nums, num):
    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == num:
            return mid
        elif nums[mid] < num:
            right = mid - 1
        else:
            left = mid + 1
    return -1
```

```
result = locate_position(cards,query)
print(result)

if(result == output):
  print("Pass")
else:
  print("No Match")
```



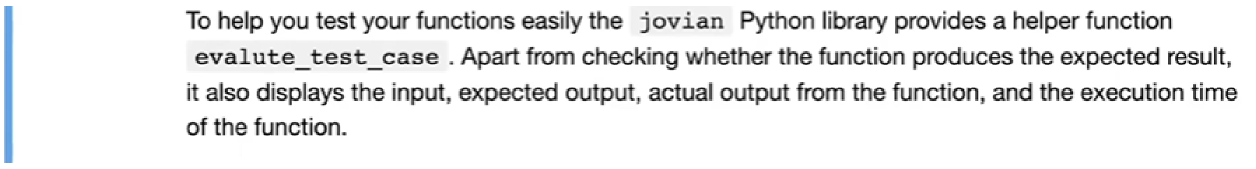
3. Come up with a correct solution for the problem. State it in plain English.

Our first goal should always be to come up with a *correct* solution to the problem, which may necessarily be the most *efficient* solution. The simplest or most obvious solution to a problem, which generally involves checking all possible answers is called the *brute force* solution.

In this problem, coming up with a correct solution is quite easy: Bob can simply turn over cards in order one by one, till he find a card with the given number on it. Here's how we might implement it

1. Create a variable `position` with the value 0.
2. Check whether the number at index `position` in `card` equals `query`.
3. If it does, `position` is the answer and can be returned from the function
4. If not, increment the value of `position` by 1, and repeat steps 2 to 5 till we reach the last position.
5. If the number was not found, return `-1`.

> **Linear Search Algorithm**: Congratulations, we've just written our first *algorithm*! An algorithm is simply a list of statements which can be converted into code and executed by a computer on different sets of inputs. This particular algorithm is called linear search, since it involves searching through a list in a linear fashion i.e. element after element



> To help you test your functions easily the `jovian` Python library provides a helper function `evalute_test_case`. Apart from checking whether the function produces the expected result, it also displays the input, expected output, actual output from the function, and the execution time of the function.