Forum Code Workbook

Problem Set 1 — Algorithmic Strategies

Important notes:

- Watch this video recorded by Joram Erbarth (M23) with advice on how to prepare for CS110 assignments. Most of the suggestions will also apply to other CS courses, so make sure to bookmark this video for future reference. You will also notice that the video refers to submitting primary and secondary resources, but for this specific assignment, since you will answer questions directly in the notebook, you won't need to worry about uploading your work.
- Make sure to include your work whenever you see the labels `###YOUR DOCSTRING HERE` or `###YOUR CODE HERE` (there are several code cells per question you can use throughout the notebook, but you need not use them all).
- Please refer to the CS110 course guide on how to submit your assignment materials.
- If you have any questions, do not hesitate to reach out to the TAs in the Slack channel #cs110-algo, or come to the instructors' OHs.

Question 1 of 10

Setting up:

Start by stating your name and identifying your collaborators. Please comment on the nature of the collaboration (for example, if you briefly discussed the strategy to solve problem 1, say so, and explicitly point out what you discussed). Example:

Name: Ahmed Souza

Collaborators: Lily Shakespeare, Anitha Holmes

Details: I discussed the iterative strategy of problem 1 with Lily, and asked Anitha to help me design an experiment for problem 2.

Normal

Name: Maria Eduarda Távora Carneiro

Collaborators: Robson Amorim, Thomas Linck, Nicole Dantas

Details: Robson and Thomas discussed with me possible ways to create the list with the

create the strobogrammatic number etc). Nothing too concrete was discussed, it was an informal conversation with a flow of possible ideas. With Nicole, I discussed possible ideas for the recursions. Again mostly a flow of ideas.

Problem 1

A strobogrammatic number is a number that looks the same when rotated 180 degrees (i.e., upside down). For instance, 8 stays the same after rotating 180 degrees; therefore, it is a strobogrammatic number. Here, we will define 1 as a strobogrammatic number, though this will depend on the font used.

Your goal is to define two different strategies to return all the strobogrammatic numbers of length n, where n is a positive integer. The questions below will guide you through the necessary work to achieve this.

Question 2 of 10

A. Explain how you would design two approaches (an iterative and a recursive one) to solving this computational problem in plain English to determine whether a number is strobogrammatic. Avoid using technical jargon (such as "indices" and while/for loops), and focus on explaining in as simple terms as possible how the algorithm works and how it is guaranteed, upon termination, to return the correct answer.

Normal

1. Get the number
2. Check if the number has any values different than 0, 1, 6 and 9
3. If yes, the number is not strobogrammatic
4. If no:
    a. Check if the amount of 6s and 9s are the same
    b. If no, the number is not strobogrammatic
    c. If yes:
        i. the number has to be a "palindrome" or have the 9s and the 6s at opposite positions

Iterative approach:

1. To check if the number is a "palindrome" we can have two pointers for the number, one at the beginning and one at the end of the number
2. If the value on pointer one is equal to pointer 2 we continue iterating through the number, however, if the number is 9 in one pointer, the number on the other one has

to be 6
3. If the numbers are different or the 9s don't correspond with the 6s, the number is not strobogrammatic

Recursive approach:

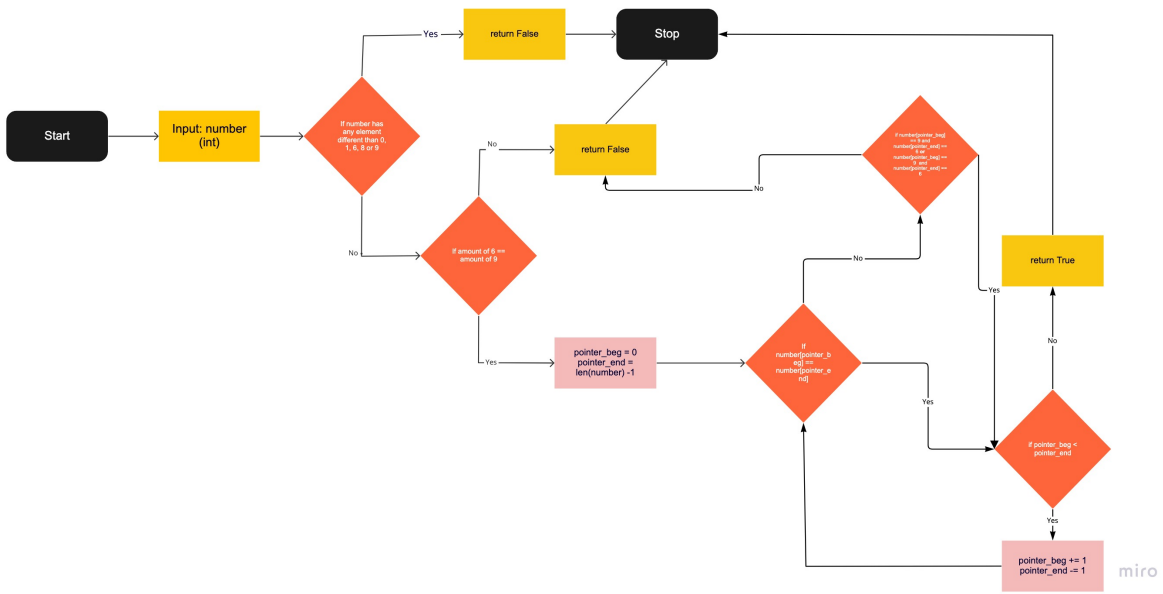
1. We transform the number into a list
2. To check if the number is a "palindrome" we can have two pointers for the number, one at the beginning and one at the end of the number
3. If the value on pointer one is equal to pointer two we continue iterating through the number, however, if the number is 9 in one pointer, the number on the other one has to be 6
4. For each pair of numbers checked we pop them from the list
5. We recall the function to check if the 1st pointer is equal to the last or if one is a 9 and the other a 6

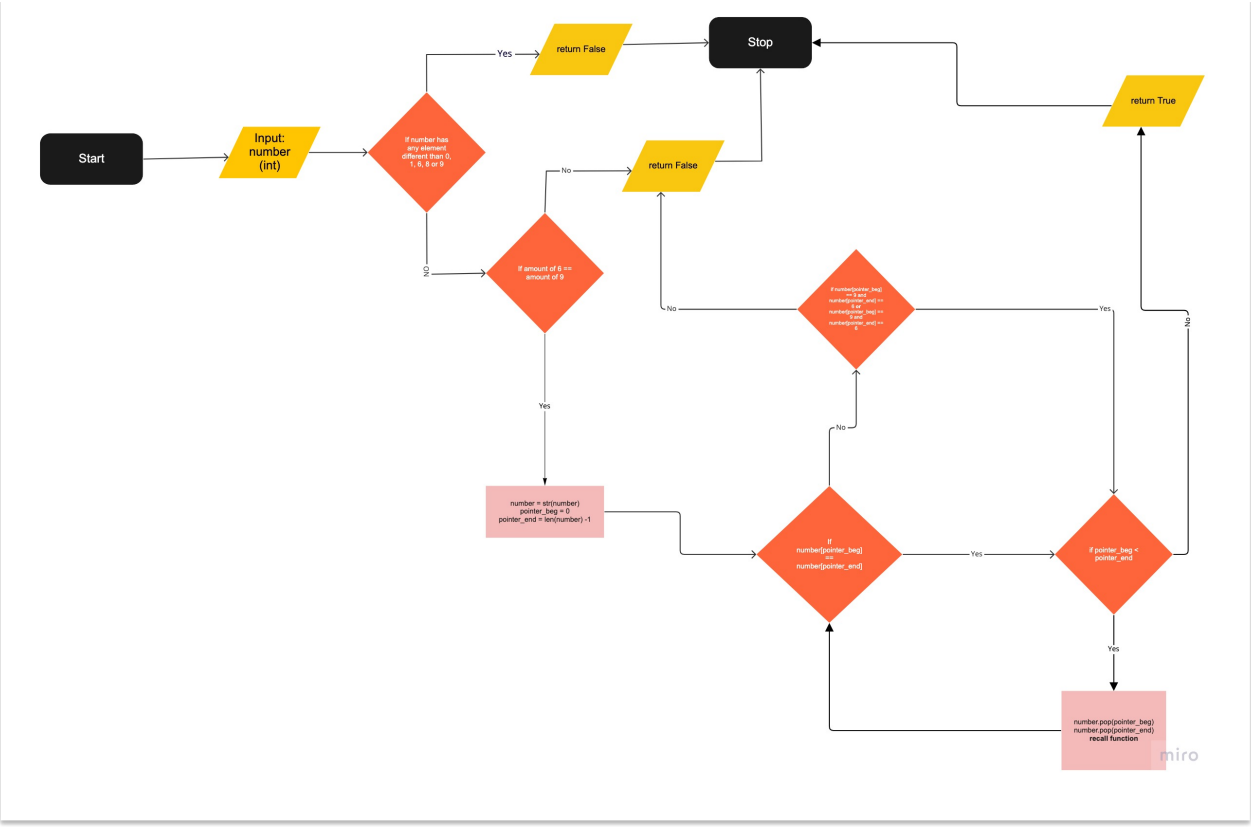
In both approaches, if the numbers are not the same or 9s do not correspond to the 6s, we can end the code since the number won't be strobogrammatic.

Question 3 of 10

B. Using your descriptions from question 1 A, produce two flowcharts that correctly describe each approach. Please be particularly careful about describing the termination condition for both algorithms. Upload your flowcharts below.

Q1_iter.jpg (152 kB)





Drop or upload a file here

C. Provide both recursive and iterative Python implementations that return all the strobogrammatic numbers of length n as a sorted array.

Remember to add at least 3 test cases to demonstrate that your function is correctly implemented in Python.

Code Cell 1 of 22

In [1]

```
def strobogrammatic_iterative(n):
    '''
    Return all the strobogrammatic numbers that are of length n through
recursive
    approach
    ------------
    Parameters:
    n: int
        The targeted length of the digit
    ------------
```

```
        All strobogrammatic numbers that are of the targeted length
    '''

    numbers_list = []

    # generate range of number with lenght n
    for number in range(10**(n-1), (10**(n)-1)):
        # transform it into a str so we can iterate trough it
        number_str = str(number)

        # don't do nothing if the number has a non-strob. number
        if '2' in number_str or '3' in number_str or '4' in number_str or '5' in number_str or '7' in number_str:
            pass
        else:
            # transform each character of a number in an element of a list
            number_array = list(number_str)



            length = len(number_array)
            counter = 0

            # iterating trough the number
            while counter < length:

                # checking if the number starts and ends with its correspondent
                if number_array[0] == '0' and number_array[-1] == '0'or number_array[0] == '1' and number_array[-1] == '1' or number_array[0] == '8' and number_array[-1] == '8' or number_array[0] == '9' and number_array[-1] == '6' or number_array[0] == '6' and number_array[-1] == '9':
                    holder = number_str

                    # checking if the number has only one character after possible .pop()
                    if len(number_array) == 1:
                        # checking if is strob.
                        if number_array[0] == '0' or number_array[0] == '1' or number_array[0] == '8':
                            numbers_list.append(holder)
                            counter += length # done with the checking
```

```
                            counter += length

                    # checking if the number has two characters after possible .pop()
                    elif len(number_array) == 2:
                        # checking if after possible .pop() teh number is strob.
                        if number_array[0] + number_array[1] == '69' or number_array[0] + number_array[1] == '96' or number_array[0] + number_array[1] == '88' or number_array[0] + number_array[1] == '11' or number_array[0] + number_array[1] == '00':
                            numbers_list.append(holder) # here is strob
                            counter += length # done with checking
                        else:
                            counter += length

                    #### .POP() ####
                    # if the number has the same character at its beginning and end or 9 and 6 in exchanged 1st and last positions
                    # we can pop those numbers from the list and proceed to check the others in the middle of the number
                    else:
                        number_array.pop(0)
                        number_array.pop(-1)
                        counter += 1
                else:
                    counter += length




    return numbers_list

######### RECURSIVE ############
def unfilter_strob (n):
    '''
    Return all the strobogrammatic numbers that are of length n through recursive
    approach unsorted and unfiltered
    ------------
    Parameters:
    n: int
        The targeted length of the digit
```

```
    Returns: list
        All unfiltered strobogrammatic numbers that are of the targeted length but some have 0 at its beginning and ending
    '''

    # base cases
    if n == 0:
        return ['']
    if n == 1:
        return ['0', '1', '8']

    else:
        # recursion
        # if n > 1 we will append numbers to its beginning and end and store those new number in numbers_result
        recall = unfilter_strob(n-2)
        numbers_result = []

        # for each number generated by the recursion we need to add to its beginning new numbers until we have number of length n
        for middle in recall:
            # I realized for n >= 4 they were not printing numbers with zeros on the middle. For even number they don't
            # print any and for odd numbers they only print the ones that come with one zero on the middle (coming from the recall of n = 1)
            # so i added the appending of zeroes
            numbers_result.append("0" + middle + "0")
            numbers_result.append("1" + middle + "1")
            numbers_result.append("6" + middle + "9")
            numbers_result.append("8" + middle + "8")
            numbers_result.append("9" + middle + "6")


    return numbers_result


def strobogrammatic_recursive(n):
    '''
    Return all the strobogrammatic numbers that are of length n through recursive
    approach sorted and filtered
    ------------
```

```
    n: int
        The targeted length of the digit
    ------------
    Returns: list
        All strobogrammatic numbers that are of the targeted length
    '''

    # return of a list with strob. numbers unfilterd and unsorted
    unfiltered = unfilter_strob(n)

    if n == 0:
        return unfiltered
    if n == 1:
        return unfiltered

    # iterate trough the elements of the list
    for i in unfiltered:
        # analize the 1st character of the element
        if i[0] == '0':
            index = unfiltered.index(i)
            # remove it if it starts with 0
            unfiltered.remove(i)
            # put a 'holder' in its place so we don't mess up the indexing of the list
            unfiltered.insert(index, "holder")

    # only add to the list the strob. numbers
    filtered_strob = [i for i in unfiltered if i!= 'holder']
    # sort the list
    filtered_strob.sort(key = int)



    return filtered_strob
```

Code Cell 2 of 22

In [2]
```
1 ### iterative tests for n > 0
2 print(strobogrammatic_iterative(1))
3 print(strobogrammatic_iterative(2))
4 print(strobogrammatic_iterative(3))
5
```

Run Code

Out [2]
```
['1', '8']
['11', '69', '88', '96']
['101', '111', '181', '609', '619', '689', '808', '818', '888', '906', '916',
'986']
```

Code Cell 3 of 22

In [3]
```
1 ### iterative tests for n > 0
2 print(strobogrammatic_iterative(4))
3 print(strobogrammatic_iterative(5))
4 print(strobogrammatic_iterative(6))
```

Run Code

Out [3]
```
['1001', '1111', '1691', '1881', '1961', '6009', '6119', '6699', '6889',
'6969', '8008', '8118', '8698', '8888', '8968', '9006', '9116', '9696',
'9886', '9966']
['10001', '10101', '10801', '11011', '11111', '11811', '16091', '16191',
'16891', '18081', '18181', '18881', '19061', '19161', '19861', '60009',
'60109', '60809', '61019', '61119', '61819', '66099', '66199', '66899',
'68089', '68189', '68889', '69069', '69169', '69869', '80008', '80108',
'80808', '81018', '81118', '81818', '86098', '86198', '86898', '88088',
'88188', '88888', '89068', '89168', '89868', '90006', '90106', '90806',
'91016', '91116', '91816', '96096', '96196', '96896', '98086', '98186',
```

```
['100001', '101101', '106901', '108801', '109601', '110011', '111111',
'116911', '118811', '119611', '160091', '161191', '166991', '168891',
'169691', '180081', '181181', '186981', '188881', '189681', '190061',
'191161', '196961', '198861', '199661', '600009', '601109', '606909',
'608809', '609609', '610019', '611119', '616919', '618819', '619619',
'660099', '661199', '666999', '668899', '669699', '680089', '681189',
'686989', '688889', '689689', '690069', '691169', '696969', '698869',
'699669', '800008', '801108', '806908', '808808', '809608', '810018',
'811118', '816918', '818818', '819618', '860098', '861198', '866998',
'868898', '869698', '880088', '881188', '886988', '888888', '889688',
'890068', '891168', '896968', '898868', '899668', '900006', '901106',
'906906', '908806', '909606', '910016', '911116', '916916', '918816',
'919616', '960096', '961196', '966996', '968896', '969696', '980086',
'981186', '986986', '988886', '989686', '990066', '991166', '996966',
'998866', '999666']
```

Code Cell 4 of 22

In [4]:

```
### recursive tests for n > 0
print(strobogrammatic_recursive(1))
print(strobogrammatic_recursive(2))
print(strobogrammatic_recursive(3))
```

Run Code

Out [4]:

```
['0', '1', '8']
['11', '69', '88', '96']
['101', '111', '181', '609', '619', '689', '808', '818', '888', '906', '916',
'986']
```

Code Cell 5 of 22

In [5]:

```
### recursive tests for n > 0
print(strobogrammatic_recursive(4))
print(strobogrammatic_recursive(5))
print(strobogrammatic_recursive(6))
```

Run Code

Out [5]

```
['1001', '1111', '1691', '1881', '1961', '6009', '6119', '6699', '6889',
'6969', '8008', '8118', '8698', '8888', '8968', '9006', '9116', '9696',
'9886', '9966']
['10001', '10101', '10801', '11011', '11111', '11811', '16091', '16191',
'16891', '18081', '18181', '18881', '19061', '19161', '19861', '60009',
'60109', '60809', '61019', '61119', '61819', '66099', '66199', '66899',
'68089', '68189', '68889', '69069', '69169', '69869', '80008', '80108',
'80808', '81018', '81118', '81818', '86098', '86198', '86898', '88088',
'88188', '88888', '89068', '89168', '89868', '90006', '90106', '90806',
'91016', '91116', '91816', '96096', '96196', '96896', '98086', '98186',
'98886', '99066', '99166', '99866']
['100001', '101101', '106901', '108801', '109601', '110011', '111111',
'116911', '118811', '119611', '160091', '161191', '166991', '168891',
'169691', '180081', '181181', '186981', '188881', '189681', '190061',
'191161', '196961', '198861', '199661', '600009', '601109', '606909',
'608809', '609609', '610019', '611119', '616919', '618819', '619619',
'660099', '661199', '666999', '668899', '669699', '680089', '681189',
'686989', '688889', '689689', '690069', '691169', '696969', '698869',
'699669', '800008', '801108', '806908', '808808', '809608', '810018',
'811118', '816918', '818818', '819618', '860098', '861198', '866998',
'868898', '869698', '880088', '881188', '886988', '888888', '889688',
'890068', '891168', '896968', '898868', '899668', '900006', '901106',
'906906', '908806', '909606', '910016', '911116', '916916', '918816',
'919616', '960096', '961196', '966996', '968896', '969696', '980086',
'981186', '986986', '988886', '989686', '990066', '991166', '996966',
'998866', '999666']
```

Question 4 of 10

Why are your test cases appropriate or possibly sufficient?

Normal

My tests seem to indicate both approaches are generating satisfying outputs. For the value of n, the code is generating the expected result. However, it is important to point out that for the iterative approach if n = 0 or n < 0 the code will generate an error on the code. If n = 0 the recursive code will return [' '] but will generate an error if n < 0. Those cases were not given special treatment because numbers can only have 1 as their minimum length.

Please run the following code cell to check if your code passes some corner cases.

Code Cell 6 of 22 - Hidden Code

Run Code

```
Out [6]   Testing your code...
          All tests have completed successfully! Excellent work!
```

Code Cell 7 of 22

```
In [7]   1 ### PLEASE DO NOT USE THIS CELL, IT WILL BE USED FOR FURTHER TESTING
```

Run Code

Question 5 of 10

D. Design experiments to determine the average run time and number of steps that both implementations take for increasing values of n. Consider n to be smaller than 20. Which implementation is better and why? (Please refrain from quoting/deriving bigO results as the question merely asks you to focus on the experimental analysis from your specific implementations).

Normal

Comparison of runtimes:

For the iterative approach, I reached the following results:
n -> runtime
1 -> 0.00029087066650390625
2 -> 0.00032806396484375
3 -> 0.0006608963012695312
4 -> 0.002731800079345703
5 -> 0.023017168045043945
6 -> 0.3581092357635498

7 -> 4.072934865951538
8 -> 41.27037596702576

For the recursive approach, I reached the following results:
n -> runtime
1 -> 0.0002562999725341797
2 -> 0.0002589225769042969
3 -> 0.0002753734588623047
4 -> 0.0003058910369873047
5 -> 0.0003054141998291016
6 -> 0.0003263465881347656
7 -> 0.0008523464202880859
8 -> 0.0018112659454345703
9 -> 0.012666702270507812
10 -> 0.033956289291381836
11 -> 0.5442736148834229
12 -> 1.559579610824585
13 -> 14.461968660354614
14 -> 40.46727466583252
15 -> 367.4832446575165

Even though both cases for small n values have similar outcomes, it is noticeable that when n grows, the iterative approach quickly becomes too slow. On my computer, for example, I could only test until n = 8; for n = 9 it already took too much time. However, for the recursive approach, I reached n = 15, and it is noticeable that when n =14, we have a runtime similar to the iterative process when n = 8. Having analyzed that, we can say the recursive method is much more efficient regarding the amount of time to generate an output given growing n's.

Comparison of steps:

For the iterative approach, I reached the following results:
n: steps
1:8
2:42
3:250
4:1218
5:6258
6:31098
7:156298
8:780498

For the recursive approach, I didn't manage to reach any results. I tried many different

believe it would probably escalate slower than the iterative approach, just like the runtime experiment. It is also important to point out that adding steps can be helpful but is also an arbitrary method since the definition of where to put those steps can vary. Having said that, using the runtime makes more sense when we want to arrive at a result closer to the actual metrics. The runtime can be slightly influenced by the capacity of the computer and how much of the RAM is being used at the moment, but it is way less arbitrary than the step counting method.

Code Cell 8 of 22

In [8]:

```
### iterative approach runtime
# this code was based on CS110 Session

import time

start_time = time.time()

def strobogrammatic_iterative(n):
    '''
    Return all the strobogrammatic numbers that are of length n through recursive
    approach
    ------------
    Parameters:
    n: int
        The targeted length of the digit
    ------------
    Returns: list
        All strobogrammatic numbers that are of the targeted length
    '''

    numbers_list = []

    # generate range of number with lenght n
    for number in range(10**(n-1), (10**(n)-1)):
        # transform it into a str so we can iterate trough it
        number_str = str(number)

        # don't do nothing if the number has a non-strob. number
        if '2' in number_str or '3' in number_str or '4' in number_str or
```

```
            pass
        else:
            # transform each character of a number in an element of a
list
            number_array = list(number_str)



            length = len(number_array)
            counter = 0

            # iterating trough the number
            while counter < length:

                # checking if the number starts and ends with its
correspondent
                if number_array[0] == '0' and number_array[-1] == '0'or
number_array[0] == '1' and number_array[-1] == '1' or number_array[0] ==
'8' and number_array[-1] == '8' or number_array[0] == '9' and
number_array[-1] == '6' or number_array[0] == '6' and number_array[-1] ==
'9':
                    holder = number_str

                    # checking if the number has only one character after
possible .pop()
                    if len(number_array) == 1:
                        # checking if is strob.
                        if number_array[0] == '0' or number_array[0] ==
'1' or number_array[0] == '8':
                            numbers_list.append(holder)
                            counter += length # done with the checking
                        else:
                            counter += length

                    # checking if the number has two characters after
possible .pop()
                    elif len(number_array) == 2:
                        # checking if after possible .pop() teh number is
strob.
                        if number_array[0] + number_array[1] == '69' or
number_array[0] + number_array[1] == '96' or number_array[0] +
number_array[1] == '88' or number_array[0] + number_array[1] == '11' or
number_array[0] + number_array[1] == '00':
                            numbers_list.append(holder) # here is strob
                            counter += length # done with checking
                        else:
```

```
                            counter += length


                    #### .POP() ####
                    # if the number has the same character at its beginning and end or 9 and 6 in exchanged 1st and last positions
                    # we can pop those numbers from the list and proceed to check the others in the middle of the number
                    else:
                        number_array.pop(0)
                        number_array.pop(-1)
                        counter += 1
                else:
                    counter += length




    return numbers_list

strobogrammatic_iterative(1)

end_time = time.time()

run_time = end_time - start_time
print(run_time)
```

Run Code

Out [8] 0.0003008842468261719

Code Cell 9 of 22

In [9]

```
### iterative approach steps
# this code was based on CS110 Session


```

```
    '''
    Return all the strobogrammatic numbers that are of length n through recursive
    approach
    ------------
    Parameters:
    n: int
        The targeted length of the digit
    ------------
    Returns: list
        All strobogrammatic numbers that are of the targeted length
    '''

    numbers_list = []
    step = 0

    # generate range of number with lenght n
    for number in range(10**(n-1), (10**(n)-1)):
        # transform it into a str so we can iterate trough it
        number_str = str(number)

        # don't do nothing if the number has a non-strob. number
        if '2' in number_str or '3' in number_str or '4' in number_str or '5' in number_str or '7' in number_str:
            pass
        else:
            # transform each character of a number in an element of a list
            number_array = list(number_str)

            step += 1

            length = len(number_array)
            counter = 0

            # iterating trough the number
            while counter < length:

                # checking if the number starts and ends with its correspondent
                if number_array[0] == '0' and number_array[-1] == '0'or number_array[0] == '1' and number_array[-1] == '1' or number_array[0] ==
```

```
number_array[-1] == '6' or number_array[0] == '6' and number_array[-1] == '9':
                    holder = number_str
                    step += 1


                    # checking if the number has only one character after possible .pop()
                    if len(number_array) == 1:
                        step += 1
                        # checking if is strob.
                        if number_array[0] == '0' or number_array[0] == '1' or number_array[0] == '8':
                            numbers_list.append(holder)
                            counter += length # done with the checking
                        else:
                            counter += length


                    # checking if the number has two characters after possible .pop()
                    elif len(number_array) == 2:
                        step += 1
                        # checking if after possible .pop() teh number is strob.
                        if number_array[0] + number_array[1] == '69' or number_array[0] + number_array[1] == '96' or number_array[0] + number_array[1] == '88' or number_array[0] + number_array[1] == '11' or number_array[0] + number_array[1] == '00':
                            numbers_list.append(holder) # here is strob
                            counter += length # done with checking
                        else:
                            counter += length


                    #### .POP() ####
                    # if the number has the same character at its beginning and end or 9 and 6 in exchanged 1st and last positions
                    # we can pop those numbers from the list and proceed to check the others in the middle of the number
                    else:
                        step += 1
                        number_array.pop(0)
                        number_array.pop(-1)
                        counter += 1
                else:
                    step += 1
```

```
                    counter += length




    return step

print(strobogrammatic_iterative(9))

```

Run Code

Out [9] 3906498

Code Cell 10 of 22

In [ ]

```
### recursive approach runtime
# this code was based on CS110 Session
start_time = time.time()
def unfilter_strob (n):
    '''
    Return all the strobogrammatic numbers that are of length n through recursive
    approach unsorted and unfiltered
    ------------
    Parameters:
    n: int
        The targeted length of the digit
    ------------
    Returns: list
        All unfiltered strobogrammatic numbers that are of the targeted length but some have 0 at its beginning and ending
    '''

    # base cases
    if n == 0:
```

Python 3 (1GB RAM) | Edit Run All Cells Kernel Busy | Interrupt

```
    if n == 1:
        return ['0', '1', '8']

    else:
        # recursion
        # if n > 1 we will append numbers to its beginning and end and store those new number in numbers_result
        recall = unfilter_strob(n-2)
        numbers_result = []

        # for each number generated by the recursion we need to add to its beginning new numbers until we have number of length n
        for middle in recall:
            # I realized for n >= 4 they were not printing numbers with zeros on the middle. For even number they don't
            # print any and for odd numbers they only print the ones that come with one zero on the middle (coming from the recall of n = 1)
            # so i added the appending of zeroes
            numbers_result.append("0" + middle + "0")
            numbers_result.append("1" + middle + "1")
            numbers_result.append("6" + middle + "9")
            numbers_result.append("8" + middle + "8")
            numbers_result.append("9" + middle + "6")


    return numbers_result


def strobogrammatic_recursive(n):
    '''
    Return all the strobogrammatic numbers that are of length n through recursive
    approach sorted and filtered
    ------------
    Parameters:
    n: int
        The targeted length of the digit
    ------------
    Returns: list
        All strobogrammatic numbers that are of the targeted length
    '''

```

```
    unfiltered = unfilter_strob(n)

    if n == 0:
        return unfiltered
    if n == 1:
        return unfiltered

    # iterate trough the elements of the list
    for i in unfiltered:
        # analize the 1st character of the element
        if i[0] == '0':
            index = unfiltered.index(i)
            # remove it if it starts with 0
            unfiltered.remove(i)
            # put a 'holder' in its place so we don't mess up the indexing of the list
            unfiltered.insert(index, "holder")

    # only add to the list the strob. numbers
    filtered_strob = [i for i in unfiltered if i!= 'holder']
    # sort the list
    filtered_strob.sort(key = int)




    return filtered_strob

strobogrammatic_recursive(15)
end_time = time.time()
run_time = end_time - start_time
print(run_time)

```

Run Code

Code Cell 11 of 22

```
In [ ]:  1 ### recursive approach along
```

```
# this code was based on CS110 Session


def unfilter_strob (n):
    '''
    Return all the strobogrammatic numbers that are of length n through recursive
    approach unsorted and unfiltered
    ------------
    Parameters:
    n: int
        The targeted length of the digit
    ------------
    Returns: list
        All unfiltered strobogrammatic numbers that are of the targeted length but some have 0 at its beginning and ending
    '''

    global step
    step = 0

    # base cases
    if n == 0:
        step += 1
        return ['']

    if n == 1:
        step += 1
        return ['0', '1', '8']



    else:
        # recursion
        # if n > 1 we will append numbers to its beginning and end and store those new number in numbers_result
        recall = unfilter_strob(n-2)
        numbers_result = []

        # for each number generated by the recursion we need to add to its beginning new numbers until we have number of length n
        for middle in recall:
```

```
            # I realized for n >= 4 they were not printing numbers with zeros on the middle. For even number they don't
            # print any and for odd numbers they only print the ones that come with one zero on the middle (coming from the recall of n = 1)
            # so i added the appending of zeroes
            numbers_result.append("0" + middle + "0")
            step += 1
            numbers_result.append("1" + middle + "1")
            step += 1
            numbers_result.append("6" + middle + "9")
            step += 1
            numbers_result.append("8" + middle + "8")
            step += 1
            numbers_result.append("9" + middle + "6")
            step += 1
    return numbers_result


def strobogrammatic_recursive(n):
    '''
    Return all the strobogrammatic numbers that are of length n through recursive
    approach sorted and filtered
    ------------
    Parameters:
    n: int
        The targeted length of the digit
    ------------
    Returns: list
        All strobogrammatic numbers that are of the targeted length
    '''

    # return of a list with strob. numbers unfilterd and unsorted
    step = 0
    unfiltered = unfilter_strob(n)

    if n == 0:
        step += 1
        return unfiltered
    if n == 1:
        step += 1
```

```
79
80      # iterate trough the elements of the list
81      for i in unfiltered:
82          step += 1
83          # analize the 1st character of the element
84          if i[0] == '0':
85              index = unfiltered.index(i)
86              step += 1
87              # remove it if it starts with 0
88              unfiltered.remove(i)
89              step += 1
90              # put a 'holder' in its place so we don't mess up the
indexing of the list
91              unfiltered.insert(index, "holder")
92              step += 1
93
94      # only add to the list the strob. numbers
95      filtered_strob = [i for i in unfiltered if i!= 'holder']
96      step += 1
97      # sort the list
98      filtered_strob.sort(key = int)
99      step += 1
100
101
102
103     return filtered_strob, step
104
```

Run Code

```
Out [ ]    ['0', '1', '8']
```

Question 2

You are writing an algorithm for a post office of your rotation city that will send trucks to different houses in the city. The post office wants to minimize the number of trucks needed, so all the homes with the same postal code should only require one truck.

The houses' postal codes are stored in a sorted array. Many places can share the same postal code. To simplify the database, you have implemented a dictionary that maps the postal codes (that can also contain letters) to more specific numbers. For example, 94102 has been mapped to 1.

Given a `target` postal code, your task is to return the beginning and end index of this `target` value in the array (inclusive) and the total number of postal codes. If this postal code is not in the database, return `None`.
For example, given the postal array `[1, 2, 2, 3, 4, 4, 4, 5, 5, 5, 5, 7]`, and our target `4`, the answer should be `[4, 6, 3]`, where `4` is the start-index, `6` is the end-index, and `3` is the number of postal codes of this value. If our target is `1` for the same postal array, the answer should be `[0, 0, 1]`. If the target is `6`, the answer should be `None`.

Question 6 of 10

A. Explain how you would design two approaches (an iterative and a recursive one) to solve this computational problem in plain English. Avoid using technical jargon (such as "indices" and while/for loops), and focus on explaining in as simple terms as possible how the algorithm works and how it is guaranteed, upon termination, to return the correct answer.

Normal

Iterative Approach:

1. Get the list with the different numbers that represent the postal numbers
2. If this list contains no number or the number we are looking for is not in it we should have as a result the None result
3. However, if the list does contain the number we are looking for we do the following:
    a. We put one pointer at the beginning of the list (as if we have one pointer finger of our hands pointing to the first element of the list)
    b. If our pointer finds the number we are looking for we should write down the position of its first occurrence and keep looking until we find a different number
    c. When we come into another number we stop and note down the last occurrence of our target
    d. The number of occurrences of the target can be computed as the position of its last occurrence minus the position of its first occurrence plus 1 (considering the first position of the list is 0)

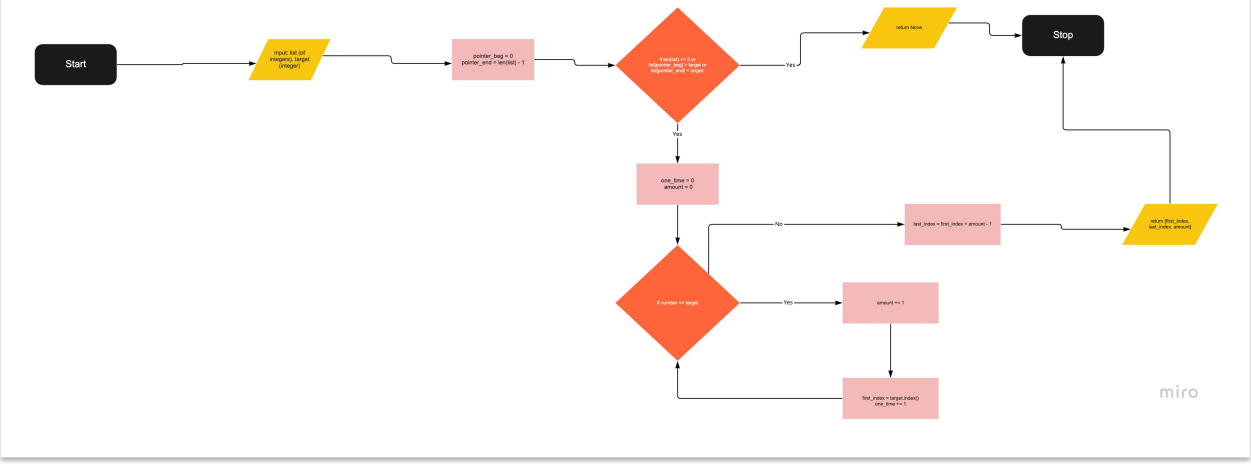
Recursive Approach:

1. Get the list with the different numbers that represent the postal numbers
2. If this list contains no number or the number we are looking for is not in it we should have as a result the None result
3. However, if the list does contain the number we are looking for we do the following:
    a. Check if the first element of the list is the target, if yes note down its position as the first position of the list
    b. If not, check the list again but now from the second position inwards (this is where we recall the function)
        i. this process will give us the position where the target number first appears
    c. We do a similar process to check the last appearance of the target number
    d. We check if the last element of the list is the target, if yes we note down its position as the last position of the list
    e. If not, we recheck the list now starting from the position to the left of the last one (recursion happens here)
        i. This process will generate the position of the last appearance of the target number
    f. When both the position of the first and last occurrence is discovered we can find the number of occurrences following the formula:
        i. last occurrence - first occurrence + 1 = number of occurrences

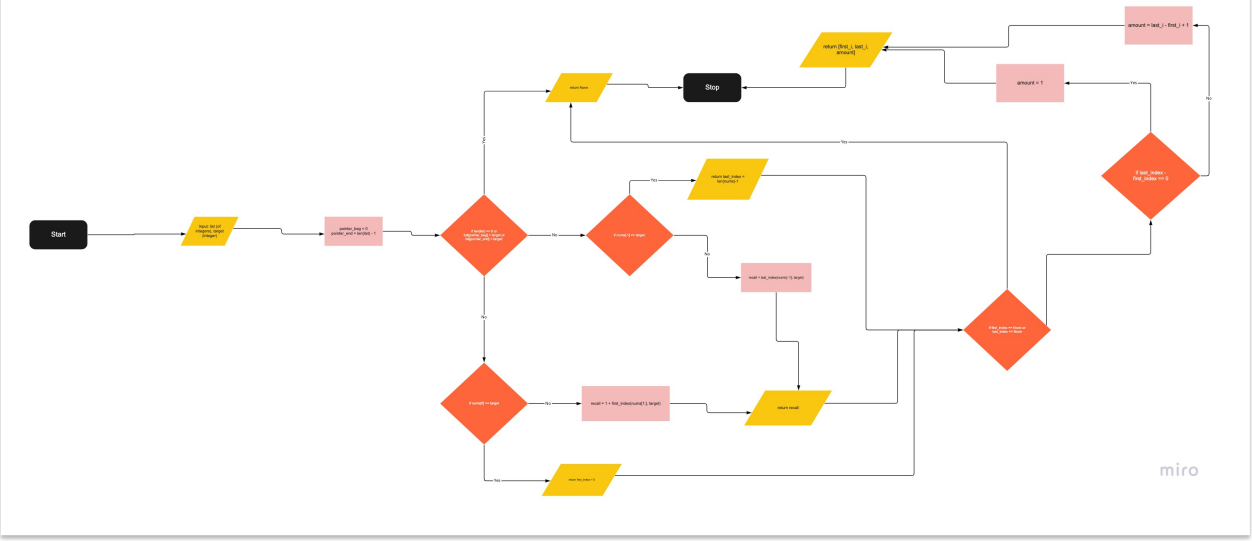
Question 7 of 10

B. Using your descriptions from question 2 A, produce two flowcharts that correctly describe each approach. Please be particularly careful about describing the termination condition for both algorithms. Upload your flowcharts below.

cs110_1_assignment.jpg (112 kB)



cs110_1_assignment (1).jpg (130 kB)



Drop or upload a file here

C. Provide recursive and iterative Python implementations that return a list with three elements (start-index, end-index, and number of matches found in the sorted array).

Remember to add at least 3 test cases to demonstrate that your function is correctly implemented in Python.

You will need to justify why those specific test cases are appropriate and possibly sufficient.

Code Cell 12 of 22

In [ ]

```
def iterative_binary_search_extension(nums, target):
    '''
    Return the start and end index (inclusively)
    of a target element in a sorted array
    ------------
    Parameters:
    nums: list
        a sorted array
    target: int
        the target value to be found
    ------------
    Returns:
```

Python 3 (1GB RAM) | Edit Run All Cells Kernel Busy | Interrupt

```
        list of the beginning and end index if found, else None
    '''

    one_time = 0

    amount = 0

    pointer_beg = 0
    pointer_end = len(nums) - 1

    # check if the list if empty, or if the number is not in the list
    if len(nums) == 0 or nums[pointer_beg] > target or nums[pointer_end] < target:
            return None
    else:
        for num in nums:
            # if number is found
            if num == target:
                amount += 1
                # attribute the index of the number only once to first_index
                while one_time < 1:
                    first_index = nums.index(target)
                    one_time += 1
        # calculate the index for last_index
        last_index = first_index + amount - 1

    return [first_index, last_index, amount]


### RECURSIVE #####
def first_index (nums, target):
    '''
    Return the start index
    of a target element in a sorted array
    ------------
    Parameters:
    nums: list
        a sorted array
    target: int
        the target value to be found
```

```
    Returns:
    integer
        first index of the target's occurrence
    '''
    pointer_beg = 0
    pointer_end = len(nums) - 1

    # check if the list if empty, or if the number is not in the list
    if len(nums) == 0 or nums[pointer_beg] > target or nums[pointer_end] < target:
            return None
    # find first index
    else:
        if nums[0] == target:
            return 0
        # if the target is not on the first index of the list the code will examine the list again from the second index inwards
        else:
            recall = first_index(nums[1:], target)
            # if the number does not exist but the 1st number on the list is smaller than it and the last one is bigger
            if recall != None:
                recall = recall + 1
            else:
                return None

    return recall

def last_index(nums, target):
    '''
    Return the end index
    of a target element in a sorted array
    ------------
    Parameters:
    nums: list
        a sorted array
    target: int
        the target value to be found
    ------------
    Returns:
    integer
```

```
    '''
    pointer_beg = 0
    pointer_end = len(nums) - 1

    # check if the list if empty, or if the number is not in the list
    if len(nums) == 0 or nums[pointer_beg] > target or nums[pointer_end] < target:
            return None
    # find last index
    else:
        if nums[-1] == target:
            return len(nums)-1
        # if the target is not on the first index of the list the code will examine the list again from the second index inwards
        else:
            recall = last_index(nums[:-1], target)

    return recall



def recursive_binary_search_extension(nums, target):
    '''
    Return the start and end index (inclusively)
    of a target element in a sorted array
    ------------
    Parameters:
    nums: list
        a sorted array
    target: int
        the target value to be found
    ------------
    Returns:
    list/ None
        list of the beginning and end index if found, else None
    '''

    # call functions to find first and last index
    first_i = first_index(nums, target)
    last_i = last_index(nums, target)

```

```
    if first_i == None or last_i == None:
        return None

    # calculate amount of occurrences
    amount = last_i - first_i + 1

    # if last and first index are the same
    if last_i - first_i == 0:
        amount = 1


    return [first_i, last_i, amount]

```

Run Code

Code Cell 13 of 22

In [ ]

```
### ITERATIVE
print(iterative_binary_search_extension([0,2,2,3,3], 0))
print(iterative_binary_search_extension([0,2,2,3,3], 10))
print(iterative_binary_search_extension([0,2,2,3,3],-1))
```

Run Code

Out [ ]

```
[0, 0, 1]
None
None
```

Code Cell 14 of 22

In [ ]

```
### ITERATIVE
print(iterative_binary_search_extension([0,2,2,3,3,4,5,6,7,8], 8))
```

```
4 print(iterative_binary_search_extension([-1, 0,1,1,1,1,2,2,3,3,9,10],
-1))
5 print(iterative_binary_search_extension([-1, 0,1,1,1,1,2,2,3,3,9,10], 2))
```

Run Code

Out [ ]
```
[9, 9, 1]
[9, 9, 1]
[0, 0, 1]
[6, 7, 2]
```

Code Cell 15 of 22

In [ ]
```
1 ### RECURSIVE
2
3 print(recursive_binary_search_extension([0,2,2,3,3], 0))
4 print(recursive_binary_search_extension([0,2,2,3,3], 10))
5 print(recursive_binary_search_extension([0,2,2,3,3],-1))
```

Run Code

Out [ ]
```
[0, 0, 1]
None
None
```

Code Cell 16 of 22

In [ ]
```
1 ### RECURSIVE
2 print(recursive_binary_search_extension([0,2,2,3,3,4,5,6,7,8], 8))
3 print(recursive_binary_search_extension([0,1,1,1,1,2,2,3,3,9,10], 9))
4 print(recursive_binary_search_extension([-1, 0,1,1,1,1,2,2,3,3,9,10],
-1))
5 print(recursive_binary_search_extension([-1, 0,1,1,1,1,2,2,3,3,9,10], 2))
```

```
6 print(recursive_binary_search_extension([-1, 0,1,1,1,1,2,2,3,3,9,10], 4))
```

Run Code

```
Out [ ]  [9, 9, 1]
         [9, 9, 1]
         [0, 0, 1]
         [6, 7, 2]
         None
```

Question 8 of 10

Why are your test cases appropriate or possibly sufficient?

Normal

My tests are generating the expected output. For each input, we have correspondent results. My test cases deal with negative numbers, numbers that are not in the list, and numbers that are in the middle, beginning, and end of the list. I am not testing for non-integer values.

Please run the following code cell to check if your code passes some corner cases.

Code Cell 17 of 22 - Hidden Code

Run Code

```
Out [ ]  Testing your code...
         ▶ All tests have completed successfully! Excellent work!
```

Code Cell 18 of 22

Run Code

Question 9 of 10

D. Design several experiments to determine which of these formulations are better and why. You may want to consider:

- different edge cases for the input, and
- different metrics to determine which algorithm seems better.

Be as creative as you would like!

Normal

Taking into consideration the runtime of both algorithms we have:

Iterative:
[0,2,2,3,3], 0)) 0.00015807151794433594
[0,2,2,3,3], 10)) 0.00015735626220703125
[0,2,2,3,3],-1)) 0.00015664100646972656
[0,2,2,3,3,4,5,6,7,8], 8)) 0.00016069412231445312
[0,1,1,1,1,2,2,3,3,9,10], 9)) 0.00016093254089355469
[-1, 0,1,1,1,1,2,2,3,3,9,10], -1)) 0.00016474723815917969
[-1, 0,1,1,1,1,2,2,3,3,9,10], 2)) 0.00016808509826660156
[-1,
0,0,0,0,0,0,0,1,1,1,1,2,2,3,3,3,3,3,4,4,4,5,5,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,7,7,7,7,8,8,8,8,8,9,9,9,
9,9,10,10,10,10,10],6) 0.00022983551025390625

Recursive:
[0,2,2,3,3], 0)) 0.00027751922607421875
[0,2,2,3,3], 10)) 0.00027632713317871094
[0,2,2,3,3],-1)) 0.00027680397033691406
[0,2,2,3,3,4,5,6,7,8], 8)) 0.00029255395965576172
[0,1,1,1,1,2,2,3,3,9,10], 9)) 0.00032734870910644453
[-1, 0,1,1,1,1,2,2,3,3,9,10], -1)) 0.00030326843261718750
[-1, 0,1,1,1,1,2,2,3,3,9,10], 2)) 0.00036120414733886672
[-1,
0,0,0,0,0,0,0,1,1,1,1,2,2,3,3,3,3,3,4,4,4,5,5,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,7,7,7,7,8,8,8,8,8,9,9,9,
9,9,10,10,10,10,10],6) 0.00036549568176269531

Even though in both cases we have small lists, I have tried to consider different edge cases. Also, we can notice there is already a difference showing between the recursive and the iterative approach. The recursive is taking more time to run, maybe because of the extra amount of functions and comparisons. Based on that information, I would say for this case, the iterative code I designed is more efficient than the recursive.

Code Cell 19 of 22

In [ ]

```
### iterative runtime



start_time = time.time()
def iterative_binary_search_extension(nums, target):
    '''
    Return the start and end index (inclusively)
    of a target element in a sorted array
    ------------
    Parameters:
    nums: list
        a sorted array
    target: int
        the target value to be found
    ------------
    Returns:
    list/ None
        list of the beginning and end index if found, else None
    '''

    one_time = 0

    amount = 0

    pointer_beg = 0
    pointer_end = len(nums) - 1

    # check if the list if empty, or if the number is not in the list
    if len(nums) == 0 or nums[pointer_beg] > target or nums[pointer_end] < target:
            return None
    else:
```

Python 3 (1GB RAM) | Edit Run All Cells Kernel Busy | Interrupt

```
            # if number is found
            if num == target:
                amount += 1
                # attribute the index of the number only once to first_index
                while one_time < 1:
                    first_index = nums.index(target)
                    one_time += 1
        # calculate the index for last_index
        last_index = first_index + amount - 1

    return [first_index, last_index, amount]

iterative_binary_search_extension([-1,0,0,0,0,0,0,0,1,1,1,1,2,2,3,3,3,3,3,4,4,4,5,5,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,7,7,7,8,8,8,8,8,9,9,9,9,9,10,10,10,10,10],6)

end_time = time.time()
run_time = end_time - start_time
print(run_time)

# [0,2,2,3,3], 0)) 0.00015807151794433594
# [0,2,2,3,3], 10)) 0.00015735626220703125
# [0,2,2,3,3],-1)) 0.00015664100646972656
# [0,2,2,3,3,4,5,6,7,8], 8)) 0.00016069412231445312
# [0,1,1,1,1,2,2,3,3,9,10], 9)) 0.00016093254089355469
# [-1, 0,1,1,1,1,2,2,3,3,9,10], -1)) 0.00016474723815917969
# [-1, 0,1,1,1,1,2,2,3,3,9,10], 2)) 0.00016808509826660156
# 0.00022983551025390625
```

Run Code

```
Out [ ]    0.00022983551025390625
```

Code Cell 20 of 22

Run Code

Code Cell 21 of 22

In [ ]

```
### recursive runtime

import time

start_time = time.time()
def first_index (nums, target):
    '''
    Return the start index
    of a target element in a sorted array
    ------------
    Parameters:
    nums: list
        a sorted array
    target: int
        the target value to be found
    ------------
    Returns:
    integer
        first index of the target's occurrence
    '''
    pointer_beg = 0
    pointer_end = len(nums) - 1

    # check if the list if empty, or if the number is not in the list
    if len(nums) == 0 or nums[pointer_beg] > target or nums[pointer_end] < target:
            return None
    # find first index
    else:
        if nums[0] == target:
            return 0
        # if the target is not on the first index of the list the code will examine the list again from the second index inwards
```

```
            recall = first_index(nums[1:], target)
            # if the number does not exist but the 1st number on the list is smaller than it and the last one is bigger
            if recall != None:
                recall = recall + 1
            else:
                return None

    return recall

def last_index(nums, target):
    '''
    Return the end index
    of a target element in a sorted array
    ------------
    Parameters:
    nums: list
        a sorted array
    target: int
        the target value to be found
    ------------
    Returns:
    integer
        last index of the target's occurrence
    '''
    pointer_beg = 0
    pointer_end = len(nums) - 1

    # check if the list if empty, or if the number is not in the list
    if len(nums) == 0 or nums[pointer_beg] > target or nums[pointer_end] < target:
            return None
    # find last index
    else:
        if nums[-1] == target:
            return len(nums)-1
        # if the target is not on the first index of the list the code will examine the list again from the second index inwards
        else:
            recall = last_index(nums[:-1], target)

```

```
72
73
74
75 def recursive_binary_search_extension(nums, target):
76     '''
77     Return the start and end index (inclusively)
78     of a target element in a sorted array
79     ------------
80     Parameters:
81     nums: list
82         a sorted array
83     target: int
84         the target value to be found
85     ------------
86     Returns:
87     list/ None
88         list of the beginning and end index if found, else None
89     '''
90
91     # call functions to find first and last index
92     first_i = first_index(nums, target)
93     last_i = last_index(nums, target)
94
95     # if the functions dont find the number
96     if first_i == None or last_i == None:
97         return None
98
99     # calculate amount of occurrences
100    amount = last_i - first_i + 1
101
102    # if last and first index are the same
103    if last_i - first_i == 0:
104        amount = 1
105
106
107    return [first_i, last_i, amount]
108
109
110 recursive_binary_search_extension([-1,
0,0,0,0,0,0,0,1,1,1,1,2,2,3,3,3,3,3,4,4,4,5,5,6,6,6,6,6,6,6,6,6,6,6,6,6,6
,6,6,7,7,7,7,7,7,7,7,7,7,8,8,8,8,8,9,9,9,9,9,10,10,10,10,10],6)
```

```
end_time = time.time()
run_time = end_time - start_time
print(run_time)

# [0,2,2,3,3], 0)) 0.00027751922607421875
# [0,2,2,3,3], 10)) 0.00027632713317871094
# [0,2,2,3,3],-1)) 0.00027680397033691406
# [0,2,2,3,3,4,5,6,7,8], 8)) 0.0002925395965576172
# [0,1,1,1,1,2,2,3,3,9,10], 9)) 0.0003273487091064453
# [-1, 0,1,1,1,1,2,2,3,3,9,10], -1)) 0.0003032684326171875
# [-1, 0,1,1,1,1,2,2,3,3,9,10], 2)) 0.0003612041473388672
# [-1,
0,0,0,0,0,0,0,1,1,1,1,2,2,3,3,3,3,3,4,4,4,5,5,6,6,6,6,6,6,6,6,6,6,6,6,6
,6,6,7,7,7,7,7,7,7,7,7,7,8,8,8,8,8,9,9,9,9,9,10,10,10,10,10],6)
0.0003654956817626953
```

Run Code

```
Out [ ]    0.0003654956817626953
```

Code Cell 22 of 22

```
In [ ]   1
```

Run Code

```
Out [ ]    [11, 11, 1, 2]
```

Question 10 of 10

References

Please write here all the references you have used for your work

Normal

1. https://www.w3schools.com/python/python_variables_global.asp
2. https://www.adamsmith.haus/python/answers/how-to-insert-a-character-into-a-string-at-an-index-in-python#:~:text=To%20insert%20a%20character%20into%20a%20string%20at%20index%20i,that%20held%20the%20original%20string.
3. https://www.w3schools.com/python/python_dictionaries.asp
4. https://www.geeksforgeeks.org/python-how-to-sort-a-list-of-strings/
5. https://www.geeksforgeeks.org/remove-multiple-elements-from-a-list-in-python/
6. https://www.w3schools.com/python/ref_list_index.asp
7. https://www.geeksforgeeks.org/check-if-element-exists-in-list-in-python/
8. https://stackoverflow.com/questions/29618844/python-recursive-list-search-function
9. https://www.geeksforgeeks.org/recursive-program-to-find-all-indices-of-a-number/
10. https://stackoverflow.com/questions/64539092/how-to-modify-a-global-list-through-a-function-without-returning-a-value

 You are all done! Congratulations on finishing your first CS110 assignment!