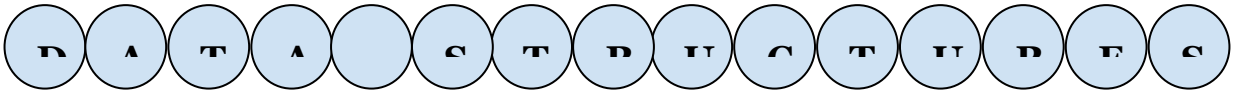
CS261 Data Structures

Assignment 1

v 1.12 (revised 6/20/2021)

Python Fundamentals Review



Contents

Summary

For this assignment, you will write a few short Python functions. There are several main objectives:

- Ensure that you are familiar with basic Python syntax constructs.
- Your programming environment is set up correctly.
- You are familiar with submitting assignments through Gradescope and troubleshooting your solution based on Gradescope output.
- You know how to import and use classes that have been pre-written for you.

For the CS261 class, we assume you are comfortable with:

- Iterating over a list of elements using for and while loops
- Accessing elements in the list using their indices
- Passing functions as parameters to other functions
- Using classes written for you (import into your code, create objects)
- Writing your own classes (including extending existing classes)
- Writing unit tests for your code
- Debugging your solutions

None of the programs in this assignment or in CS261 in general will require Python knowledge beyond what was covered in CS161 and CS162. If you completed CS161 / CS162 classes in Python, you should be able to complete this assignment. In case you need help, please post questions on Ed Discussions / Teams or feel free to contact the instructors or the ULAs.

General Instructions

1. Programs in this assignment must be written in Python v3 and submitted to Gradescope before the due date specified in the syllabus. You may resubmit your code as many times as necessary. Gradescope allows you to choose which submission will be graded.

2. In Gradescope, your code will run through several tests. Any failed tests will provide a brief explanation of testing conditions to help you with troubleshooting. Your goal is to pass all tests.

3. We encourage you to create your own test programs and cases even though this work won't have to be submitted and won't be graded. Gradescope tests are limited in scope and may not cover all edge cases. Your submission must work on all valid inputs. We reserve the right to test your submission with more tests than Gradescope.

4. Your code must have an appropriate level of comments. At a minimum, each method should have a descriptive docstring. Additionally, put comments throughout the code to make it easy to follow and understand.

5. You will be provided with a starter "skeleton" code, on which you will build your implementation. Methods defined in skeleton code must retain their names and input / output parameters. Variables defined in skeleton code must also retain their names. We will only test your solution by making calls to methods defined in the skeleton code and by checking values of variables defined in the skeleton code.

   You can add more helper methods and variables, as needed. You also are allowed to add optional default parameters to method definitions.

   However, certain classes and methods can not be changed in any way. Please see comments in the skeleton code for guidance. In particular, content of any methods pre-written for you as part of the skeleton code must not be changed.

6. Both the skeleton code and code examples provided in this document are part of assignment requirements. Please read all of them very carefully. They have been carefully selected to demonstrate requirements for each method. Refer to them for the detailed description of expected method behavior, input / output parameters, and handling of edge cases.

7. For each method, you can choose to implement a recursive or iterative solution. When using a recursive solution, be aware of maximum recursion depths on large inputs. We will specify the maximum input size that your solution must handle.

8. Unless indicated otherwise, we will test your implementation with different types of objects, not just integers. We guarantee that all such objects will have correct implementation of methods __eq__, __lt__, __gt__, __ge__, __le__ and __str__.

Specific Instructions

There are 14 separate problems in this assignment. For each problem you will have to write a Python function according to the provided specifications. A "skeleton" code and some basic test cases for each problem are provided in the file `assignment1.py`

Most problems will take as input and sometimes return as output an object of the StaticArray class. StaticArray class has been pre-written for you and is located in the file `static_array.py`

It is a very simple class that simulates the behavior of a fixed size array. It has only four methods, that allow you to:

1) Create a new static array that will store a fixed number of elements. Once the StaticArray is created, its size can not be changed.
2) Change the value of any element using its index
3) Read the value of any element using its index
4) Query the size of the array.

Below is a small example of how to do the four operations described above:

```
#create new StaticArray object to store 5 elements
arr = StaticArray(5)
# set the value of each element equal to its index multiplied by 10
for index in range(5):
    arr[index] = index * 10
# print values of all elements in reverse order
for index in range(4, -1, -1):
    print(arr(index))
# print number of elements stored in the array
print(arr.size())
```

Please review the code and comments in the StaticArray class to better understand available methods, their use, and input / output parameters. Note that since StaticArray is intentionally a very simple class, it does not have many capabilities typically associated with Python lists. You need to write your solutions only using available StaticArray functionality, as described above.

RESTRICTIONS: You are NOT allowed to use ANY built-in Python data structures and/or their methods in any of your solutions. This includes built-in Python lists, dictionaries, or anything else. Variables to hold a single value or a tuple holding two/three values are allowed. It is OK to use built-in Python generator functions like `range()`.

You are NOT allowed to directly access any variables of the StaticArray class (like `self._size` or `self._data`). All work must be done only by using StaticArray class methods.

min_max(arr: StaticArray) -> ():

Write a function that receives a one dimensional array of integers and returns a Python tuple with two values - minimum and maximum values in the input array.

Content of the input array should not be changed. You may assume that the input array will contain only integers and will have at least one element. You do not need to check for those conditions.

Example #1:

```
arr = StaticArray(5)
for i, value in enumerate([7, 8, 6, -5, 4]):
    arr[i] = value
print(min_max(arr))
```

Output:

```
(-5, 8)
```

Example #2:

```
arr = StaticArray(1)
arr[0] = 100
print(min_max(arr))
```

Output:

```
(100, 100)
```

Example #3:

```
print('\n# min_max example 3')
test_cases = (
    [3, 3, 3],
    [-10, -30, -5, 0, -10],
    [25, 50, 0, 10],
)
for case in test_cases:
    arr = StaticArray(len(case))
    for i, value in enumerate(case):
        arr[i] = value
    print(min_max(arr))
```

Output:

```
(3, 3)
(-30, 0)
(0, 50)
```

fizz_buzz(arr: StaticArray) -> StaticArray:

Write a function that receives a StaticArray of integers and returns a new StaticArray object with the content from the original array, modified as follows:

1) If the number in the original array is divisible by 3, the corresponding element in the new array should be the string 'fizz'.
2) If the number in the original array is divisible by 5, the corresponding element in the new array should be the string 'buzz'.
3) If the number in the original array is both a multiple of 3 and a multiple of 5, the corresponding element in the new array should be the string 'fizzbuzz'.
4) In all other cases, the element in the new array should have the same value as in the original array.

The content of the input array should not be changed. You may assume that the input array will contain only integers and will have at least one element. You do not need to check for these conditions.

Example #1:

```
source = [_ for _ in range(-5, 20, 4)]
arr = StaticArray(len(source))
for i, value in enumerate(source):
    arr[i] = value
print(fizz_buzz(arr))
print(arr)
```

Output:

```
STAT_ARR Size: 7 ['buzz', -1, 'fizz', 7, 11, 'fizzbuzz', 19]
STAT_ARR Size: 7 [-5, -1, 3, 7, 11, 15, 19]
```

reverse(arr: StaticArray) -> None:

Write a function that receives a StaticArray and reverses the order of the elements in the array. Reversal must be done 'in place', meaning that the original input array will be modified. You may assume that the input array will contain at least one element. You do not need to check for this condition.

Example #1:

```
source = [_ for _ in range(-20, 20, 7)]
arr = StaticArray(len(source))
for i, value in enumerate(source):
    arr.set(i, value)
print(arr)
reverse(arr)
print(arr)
reverse(arr)
print(arr)
```

Output:

```
STAT_ARR Size: 6 [-20, -13, -6, 1, 8, 15]
STAT_ARR Size: 6 [15, 8, 1, -6, -13, -20]
STAT_ARR Size: 6 [-20, -13, -6, 1, 8, 15]
```

rotate(arr: StaticArray, steps: int) -> StaticArray:

Write a function that receives two parameters - a StaticArray and an integer value (called `steps`). The function will create and return a new StaticArray, where all elements are from the original array but their position has shifted right or left `steps` number of times. The original array should not be modified.

If `steps` is a positive integer, elements should be rotated right. Otherwise, rotation is to the left. Please see the code examples below for additional details. You may assume that the input array will have at least one element. You do not need to check for this condition.

Please note that the value of the `steps` parameter can be very large (from -10^9 to 10^9). Your implementation should be able to rotate an array of at least 1,000,000 elements in a reasonable amount of time (under a minute).

Example #1:

```
source = [_ for _ in range(-20, 20, 7)]
arr = StaticArray(len(source))
for i, value in enumerate(source):
    arr.set(i, value)
print(arr)
for steps in [1, 2, 0, -1, -2, 28, -100, 2**28, -2**31]:
    print(rotate(arr, steps), steps)
print(arr)
```

Output:

```
STAT_ARR Size: 6 [-20, -13, -6, 1, 8, 15]
STAT_ARR Size: 6 [15, -20, -13, -6, 1, 8] 1
STAT_ARR Size: 6 [8, 15, -20, -13, -6, 1] 2
STAT_ARR Size: 6 [-20, -13, -6, 1, 8, 15] 0
STAT_ARR Size: 6 [-13, -6, 1, 8, 15, -20] -1
STAT_ARR Size: 6 [-6, 1, 8, 15, -20, -13] -2
STAT_ARR Size: 6 [-6, 1, 8, 15, -20, -13] 28
STAT_ARR Size: 6 [8, 15, -20, -13, -6, 1] -100
STAT_ARR Size: 6 [-6, 1, 8, 15, -20, -13] 268435456
STAT_ARR Size: 6 [-6, 1, 8, 15, -20, -13] -2147483648
STAT_ARR Size: 6 [-20, -13, -6, 1, 8, 15]
```

Example #2:

```
array_size = 1_000_000
source = [random.randint(-10**9, 10**9) for _ in range(array_size)]
arr = StaticArray(len(source))
for i, value in enumerate(source):
    arr[i] = value
print(f'Started rotating large array of {array_size} elements')
rotate(arr, 3**14)
rotate(arr, -3**15)
print(f'Finished rotating large array of {array_size} elements')
```

Output:

```
Started rotating large array of 1000000 elements
Finished rotating large array of 1000000 elements
```

sa_range(start: int, end: int) -> StaticArray:

Write a function that receives two integers `start` and `end` and returns a StaticArray that contains all consecutive values between `start` and `end` (inclusive).

Example #1:

```
cases = [
    (1, 3), (-1, 2), (0, 0), (0, -3),
    (-105, -99), (-99, -105)
]
for start, end in cases:
    print(start, end, sa_range(start, end))
```

Output:

```
1 3 STAT_ARR Size: 3 [1, 2, 3]
-1 2 STAT_ARR Size: 4 [-1, 0, 1, 2]
0 0 STAT_ARR Size: 1 [0]
0 -3 STAT_ARR Size: 4 [0, -1, -2, -3]
-105 -99 STAT_ARR Size: 7 [-105, -104, -103, -102, -101, -100, -99]
-99 -105 STAT_ARR Size: 7 [-99, -100, -101, -102, -103, -104, -105]
```

is_sorted(arr: StaticArray) -> int:

Write a function that receives a StaticArray and returns an integer that describes whether the array is sorted. The method should return 1 if the array is sorted in strictly ascending order. It should return 2 if the list is sorted in strictly descending order. Otherwise the method should return 0. Arrays consisting of a single element are considered sorted in strictly ascending order.

You may assume that the input array will contain at least one element and that values stored in the array are all of the same type (either all numbers, or strings, or custom objects, but never a mix of these). You do not need to write checks for these conditions.

Example #1:

```
test_cases = (
    [-100, -8, 0, 2, 3, 10, 20, 100],
    ['A', 'B', 'Z', 'a', 'z'],
    ['Z', 'T', 'K', 'A', '5'],
    [1, 3, -10, 20, -30, 0],
    [-10, 0, 0, 10, 20, 30],
    [100, 90, 0, -90, -200],
    ['apple']
)
for case in test_cases:
    arr = StaticArray(len(case))
    for i, value in enumerate(case):
        arr[i] = value
    print('Result:', is_sorted(arr), arr)
```

Output:

```
Result: 1 STAT_ARR Size: 8 [-100, -8, 0, 2, 3, 10, 20, 100]
Result: 1 STAT_ARR Size: 5 ['A', 'B', 'Z', 'a', 'z']
Result: 2 STAT_ARR Size: 5 ['Z', 'T', 'K', 'A', '5']
Result: 0 STAT_ARR Size: 6 [1, 3, -10, 20, -30, 0]
Result: 0 STAT_ARR Size: 6 [-10, 0, 0, 10, 20, 30]
Result: 2 STAT_ARR Size: 5 [100, 90, 0, -90, -200]
Result: 1 STAT_ARR Size: 1 ['apple']
```

sa_sort(arr: StaticArray) -> None:

Write a function that receives a StaticArray and sorts its content in non-descending order. The original array should be sorted, and your method should not return anything.

You may assume that the input array will contain at least one element and that values stored in the array are all of the same type (either all numbers, or strings, or custom objects, but never a mix of these). You do not need to write checks for these conditions.

You can implement any sort method of your choice. Sorting does not have to be very efficient or fast; a simple insertion or selection sort will suffice. Duplicates in the original array can be placed in any relative order in the sorted array (in other words, your sort does not have to be 'stable').

Your implementation should be able to sort at least 5,000 elements in a reasonable amount of time (under a minute).

Example #1:

```
test_cases = (
    [1, 10, 2, 20, 3, 30, 4, 40, 5],
    ['zebra2', 'apple', 'tomato', 'apple', 'zebra1'],
    [(1, 1), (20, 1), (1, 20), (2, 20)],
    [random.randint(-10**7, 10**7) for _ in range(5_000)]
)
for case in test_cases:
    arr = StaticArray(len(case))
    for i, value in enumerate(case):
        arr[i] = value
    print(arr if len(case) < 50 else 'Started sorting large array')
    sa_sort(arr)
    print(arr if len(case) < 50 else 'Finished sorting large array')
```

Output:

```
STAT_ARR Size: 9 [1, 10, 2, 20, 3, 30, 4, 40, 5]
STAT_ARR Size: 9 [1, 2, 3, 4, 5, 10, 20, 30, 40]
STAT_ARR Size: 5 ['zebra2', 'apple', 'tomato', 'apple', 'zebra1']
STAT_ARR Size: 5 ['apple', 'apple', 'tomato', 'zebra1', 'zebra2']
STAT_ARR Size: 4 [(1, 1), (20, 1), (1, 20), (2, 20)]
STAT_ARR Size: 4 [(1, 1), (1, 20), (2, 20), (20, 1)]
Started sorting large array
Finished sorting large array
```

remove_duplicates(arr: StaticArray) -> StaticArray:

Write a function that receives a StaticArray where the elements are already in sorted order and returns a new StaticArray with duplicate values removed. The original array should not be modified.

You may assume that the input array will contain at least one element, values stored in the array are all of the same type (either all numbers, or strings, or custom objects, but never a mix of these), and that elements of the input array are already in sorted order. You do not need to write checks for these conditions.

Example #1:

```
test_cases = (
    [1], [1, 2], [1, 1, 2], [1, 20, 30, 40, 500, 500, 500],
    [5, 5, 5, 4, 4, 3, 2, 1, 1], [1, 1, 1, 1, 2, 2, 2, 2]
)
for case in test_cases:
    arr = StaticArray(len(case))
    for i, value in enumerate(case):
        arr[i] = value
    print(arr)
    print(remove_duplicates(arr))
print(arr)
```

Output:

```
STAT_ARR Size: 1 [1]
STAT_ARR Size: 1 [1]
STAT_ARR Size: 2 [1, 2]
STAT_ARR Size: 2 [1, 2]
STAT_ARR Size: 3 [1, 1, 2]
STAT_ARR Size: 2 [1, 2]
STAT_ARR Size: 7 [1, 20, 30, 40, 500, 500, 500]
STAT_ARR Size: 5 [1, 20, 30, 40, 500]
STAT_ARR Size: 9 [5, 5, 5, 4, 4, 3, 2, 1, 1]
STAT_ARR Size: 5 [5, 4, 3, 2, 1]
STAT_ARR Size: 8 [1, 1, 1, 1, 2, 2, 2, 2]
STAT_ARR Size: 2 [1, 2]
STAT_ARR Size: 8 [1, 1, 1, 1, 2, 2, 2, 2]
```

count_sort(arr: StaticArray) -> StaticArray:

Write a function that receives a StaticArray and returns a new StaticArray with the same content sorted in non-ascending order. The original array should not be modified.

You may assume that the input array will contain at least one element and that all elements will be integers in the range [-10^9, 10^9]. It is guaranteed that the difference between the maximum and minimum values in the input will be less than 1,000. You do not need to write checks for these conditions.

Implement a FAST solution that can sort at least 5,000,000 elements in a reasonable amount of time (under a minute). For some ideas on how to approach this problem, please review the 'Counting Sort' article here: https://en.wikipedia.org/wiki/Counting_sort

Note that using a traditional sorting algorithm (even a fast sorting algorithm like merge sort, shell sort, etc.) will not pass the largest test case of 5M elements.

Example #1:

```
test_cases = (
    [1, 2, 4, 3, 5], [5, 4, 3, 2, 1], [0, -5, -3, -4, -2, -1, 0],
    [-3, -2, -1, 0, 1, 2, 3], [1, 2, 3, 4, 3, 2, 1, 5, 5, 2, 3, 1],
    [10100, 10721, 10320, 10998], [-100320, -100450, -100999, -100001],
)
for case in test_cases:
    arr = StaticArray(len(case))
    for i, value in enumerate(case):
        arr[i] = value
    print(arr if len(case) < 50 else 'Started sorting large array')
    result = count_sort(arr)
    print(result if len(case) < 50 else 'Finished sorting large array')
```

Output:

```
STAT_ARR Size: 5 [1, 2, 4, 3, 5]
STAT_ARR Size: 5 [5, 4, 3, 2, 1]
STAT_ARR Size: 5 [5, 4, 3, 2, 1]
STAT_ARR Size: 5 [5, 4, 3, 2, 1]
STAT_ARR Size: 7 [0, -5, -3, -4, -2, -1, 0]
STAT_ARR Size: 7 [0, 0, -1, -2, -3, -4, -5]
STAT_ARR Size: 7 [-3, -2, -1, 0, 1, 2, 3]
STAT_ARR Size: 7 [3, 2, 1, 0, -1, -2, -3]
STAT_ARR Size: 12 [1, 2, 3, 4, 3, 2, 1, 5, 5, 2, 3, 1]
STAT_ARR Size: 12 [5, 5, 4, 3, 3, 3, 2, 2, 2, 1, 1, 1]
STAT_ARR Size: 4 [10100, 10721, 10320, 10998]
STAT_ARR Size: 4 [10998, 10721, 10320, 10100]
STAT_ARR Size: 4 [-100320, -100450, -100999, -100001]
STAT_ARR Size: 4 [-100001, -100320, -100450, -100999]
```

Example #2:

```
array_size = 5_000_000
min_val = random.randint(-10**9, 10**9 - 998)
max_val = min_val + 998
case = [random.randint(min_val, max_val) for _ in range(array_size)]
arr = StaticArray(len(case))
for i, value in enumerate(case):
    arr[i] = value
print(f'Started sorting large array of {array_size} elements')
result = count_sort(arr)
print(f'Finished sorting large array of {array_size} elements')
```

Output:

```
Started sorting large array of 5000000 elements
Finished sorting large array of 5000000 elements
```

sa_intersection(arr1, arr2, arr3) -> StaticArray:

Write a function that receives three StaticArrays where the elements are already in sorted order and returns a new StaticArray with only those elements that appear in all three input arrays. The original arrays should not be modified. If there are no elements that appear in all three input arrays, return a StaticArray with a single None element in it.

You may assume that values stored in the input arrays are all of the same type (either all numbers, or strings, or custom objects, but never a mix of these) and that elements of the input arrays are already in the non-descending sorted order. You do not need to write checks for these conditions.

For an additional challenge, try implementing a solution with O(N) runtime complexity, where N is the total number of elements in all input arrays.

Example #1:

```
test_cases = (
    ([1, 2, 3], [3, 4, 5], [2, 3, 4]),
    ([1, 2], [2, 4], [3, 4]),
    ([1, 1, 2, 2, 5, 75], [1, 2, 2, 12, 75, 90], [-5, 2, 2, 2, 20, 75, 95])
)
for case in test_cases:
    arr = []
    for i, lst in enumerate(case):
        arr.append(StaticArray(len(lst)))
        for j, value in enumerate(sorted(lst)):
            arr[i][j] = value
    print(sa_intersection(arr[0], arr[1], arr[2]))
```

Output:

```
STAT_ARR Size: 1 [3]
STAT_ARR Size: 1 [None]
STAT_ARR Size: 3 [2, 2, 75]
```

sorted_squares(arr: StaticArray) -> StaticArray:

Write a function that receives a StaticArray where the elements are already in sorted order and returns a new StaticArray with squares of the values from the original array, sorted in non-descending order. The original array should not be modified.

You may assume that the input array will have at least one element, will contain only integers in the range [-10^9, 10^9], and that elements of the input array are already in non-descending sorted order. You do not need to check for these conditions.

Implement a FAST solution that can process at least 5,000,000 elements in a reasonable amount of time (under a minute).

Note that using a traditional sorting algorithm (even a fast sorting algorithm like merge sort, or a shell sort, etc.) will not pass the largest test case of 5M elements. Also, a solution using count_sort() as a helper method will not work here because of the wide range of values in the input array.

Example #1:

```
test_cases = (
    [1, 2, 3, 4, 5],
    [-5, -4, -3, -2, -1, 0],
    [-3, -2, -2, 0, 1, 2, 3],
)
for case in test_cases:
    arr = StaticArray(len(case))
    for i, value in enumerate(sorted(case)):
        arr[i] = value
    print(arr)
    result = sorted_squares(arr)
    print(result)
```

Output:

```
STAT_ARR Size: 5 [1, 2, 3, 4, 5]
STAT_ARR Size: 5 [1, 4, 9, 16, 25]
STAT_ARR Size: 6 [-5, -4, -3, -2, -1, 0]
STAT_ARR Size: 6 [0, 1, 4, 9, 16, 25]
STAT_ARR Size: 7 [-3, -2, -2, 0, 1, 2, 3]
STAT_ARR Size: 7 [0, 1, 4, 4, 4, 9, 9]
```

Example #2:

```
array_size = 5_000_000
case = [random.randint(-10**9, 10**9) for _ in range(array_size)]
arr = StaticArray(len(case))
for i, value in enumerate(sorted(case)):
     arr[i] = value
print(f'Started sorting large array of {array_size} elements')
result = sorted_squares(arr)
print(f'Finished sorting large array of {array_size} elements')
```

Output:

```
Started sorting large array of 5000000 elements
Finished sorting large array of 5000000 elements
```

add_numbers(arr1: StaticArray, arr2: StaticArray) -> StaticArray:

Write a function that receives two StaticArray, each representing a non-negative number, and returns a new StaticArray representing a sum of these two numbers. The original arrays should not be modified.

Each number is stored in the StaticArray as individual digits, that are in the same order as in the number. For example number 1234 would be stored as [1, 2, 3, 4], and number 0 would be stored as [0].

You may assume that each input array contains at least one element, values stored in the arrays are integers, and no leading zeros are stored in either array. You do not need to write checks for these conditions.

Restriction: For this solution, you are not allowed to use any string functions or convert between numbers and strings.

Example #1:

```
test_cases = (
    ([1, 2, 3], [4, 5, 6]),
    ([0], [2, 5]),([0], [0])
    ([2, 0, 9, 0, 7], [1, 0, 8]),
    ([9, 9, 9], [9, 9, 9, 9])
)
for num1, num2 in test_cases:
    n1 = StaticArray(len(num1))
    n2 = StaticArray(len(num2))
    for i, value in enumerate(num1):
        n1[i] = value
    for i, value in enumerate(num2):
        n2[i] = value
    print('Original nums:', n1, n2)
    print('Sum: ', add_numbers(n1, n2))
```

Output:

```
Original nums: STAT_ARR Size: 3 [1, 2, 3] STAT_ARR Size: 3 [4, 5, 6]
Sum:  STAT_ARR Size: 3 [5, 7, 9]
Original nums: STAT_ARR Size: 1 [0] STAT_ARR Size: 2 [2, 5]
Sum:  STAT_ARR Size: 2 [2, 5]
Original nums: STAT_ARR Size: 1 [0] STAT_ARR Size: 1 [0]
Sum:  STAT_ARR Size: 1 [0]
Original nums: STAT_ARR Size: 5 [2, 0, 9, 0, 7] STAT_ARR Size: 3 [1, 0, 8]
Sum:  STAT_ARR Size: 5 [2, 1, 0, 1, 5]
Original nums: STAT_ARR Size: 3 [9, 9, 9] STAT_ARR Size: 4 [9, 9, 9, 9]
Sum:  STAT_ARR Size: 5 [1, 0, 9, 9, 8]
```

spiral_matrix(rows: int, cols: int, start: int) -> StaticArray:

Write a function that receives three integers (`rows`, `cols`, and `start`), then creates and returns a 2D matrix (represented as a StaticArray of StaticArrays). The dimensions of the matrix should be `rows` x `cols`. The matrix should be filled with the integers that start from the provided `start` value and sequentially increase by 1 (if the `start` value is >= 0) or decrease by 1 (if the `start` value is < 0).

When the `start` value is non-negative, it should be placed in the upper right corner of the matrix, and all subsequent integers should be put into the matrix in a clockwise spiral order. See figure 1 below for more details. When the `start` value is negative, it should be placed in the lower left corner of the matrix, and all subsequent integers should be put into the matrix in a counterclockwise spiral order. See figure 2 below for more details.

Please review the code examples below, (especially example #2) for clarification. You may assume that `rows` and `cols` will be positive integers. You do not need to write checks for these conditions.

Fig 1: spiral_matrix(4, 4, 20)

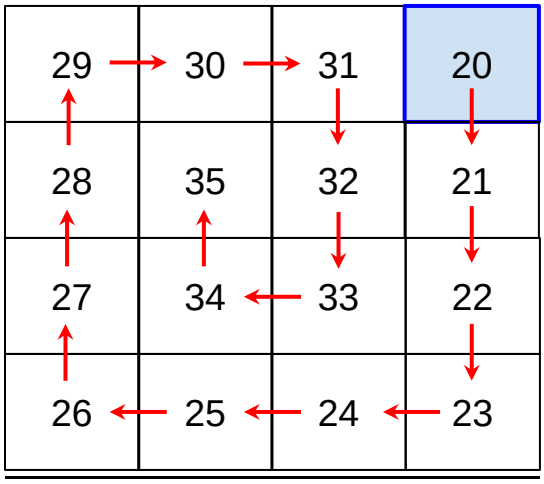
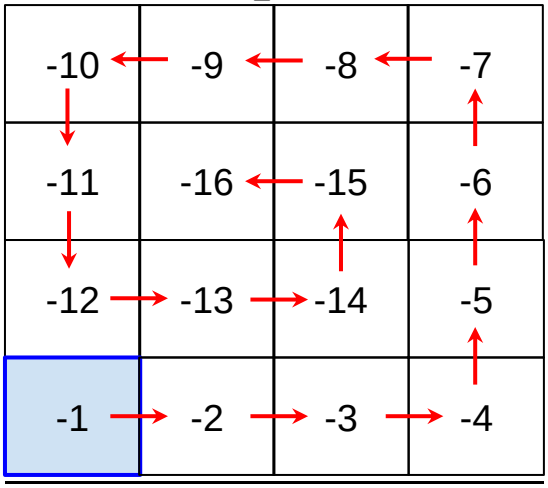


Fig 2: spiral_matrix(4, 4, -1)



Example #1:

```
matrix = spiral_matrix(1, 1, 7)
print(matrix)
if matrix: print(matrix[0])
matrix = spiral_matrix(3, 2, 12)
if matrix: print(matrix[0], matrix[1], matrix[2])
```

Output:

```
STAT_ARR Size: 1 [<static_array.StaticArray object at 0x7fd9e034efa0>]
STAT_ARR Size: 1 [7]
STAT_ARR Size: 2 [17, 12] STAT_ARR Size: 2 [16, 13] STAT_ARR Size: 2 [15, 14]
```

Example #2:

```
def print_matrix(matrix: StaticArray) -> None:
    rows, cols = matrix.size(), matrix[0].size()
    for row in range(rows):
        for col in range(cols):
            print('{:4d}'.format(matrix[row][col]), end=' ')
        print()
    print()

test_cases = (
    (4, 4, 1), (3, 4, 0), (2, 3, 10), (1, 2, 1), (1, 1, 42),
    (4, 4, -1), (3, 4, -3), (2, 3, -12), (1, 2, -42),
)
for rows, cols, start in test_cases:
    matrix = spiral_matrix(rows, cols, start)
    if matrix: print_matrix(matrix)
```

Output:

```
  10   11   12    1
   9   16   13    2
   8   15   14    3
   7    6    5    4

   7    8    9    0
   6   11   10    1
   5    4    3    2

  14   15   10
  13   12   11

   2    1

  42

 -10   -9   -8   -7
 -11  -16  -15   -6
 -12  -13  -14   -5
  -1   -2   -3   -4

 -11  -10   -9   -8
 -12  -13  -14   -7
  -3   -4   -5   -6

 -17  -16  -15
 -12  -13  -14

 -42  -43
```

transform_string(source: str, s1: str, s2: str) -> str:

Write a function that receives three strings (`source, s1, and s2`) and then returns a modified string of the same length as source. Source should be processed one character at the time, and the output string should be constructed according to the following rules:

1) If the character from the `source` string is present in `s1`, it should be replaced by the character at the same index in `s2`.
2) Otherwise, if the character is:
    a) Uppercase letter -> replace with ' '
    b) Lowercase letter -> replace with '#'
    c) Digit -> replace with '!'
    d) Anything else -> replace with '='

You may assume that strings `s1` and `s2` have the same length. You do not need to write a check for this condition.

For this problem only, you are allowed to use ANY string functions from Python's standard library.

Example #1:

```
test_cases = (
    'eMKCPVkRI%~}+$GW9EOQNMI!_%{#ED}#=-~WJbFNWSQqDO-..@}',
    'dGAqJLcNC0YFJQEB5JJKETQ0QOODKF8EYX7BGdzAACmrSL0PVKC',
    'aLiAnVhSV9}_+QOD3YSIYPR4MCKYUF9QUV9TVvNdFuGqVU4$/%D',
    'zmRJWfoKC5RDKVYO3PWMATC7BEIIVX9LJR7FKtDXxXLpFG7PESX',
    'hFKGVErCS$**!<OS<_/.>NR*)<<+IR!,=%?OAiPQJILzMI_#[+}',
    'EOQUQJLBQLDLAVQSWERAGGAOKUUKOPUWLQSKJNECCPRRXGAUABN',
    'WGBKTQSGVHHHHHTZZZZZMQKBLC66666NNR11111OKUN2KTGYUIB',
    'YFOWAOYLWGQHJQXZAUPZPNUCEJABRR6MYR1JASNOTF22MAAGTVA',
    'GNLXFPEPMYGHQQGZGEPZXGJVEYE666UKNE11111WGNW2NVLCIOK',
    'VTABNCKEFTJHXATZTYGZVLXLAB6JVGRATY1GEY1PGCO2QFPRUAP',
    'UTCKYKGJBWMHPYGZZZZZWOKQTM66666GLA11111CPF222RUPCJT')
for case in test_cases:
    print(transform_string(case, '612HZ', '261TO'))
```

Output:

```
#     #  =====  !      =====  =====  #     #  =====
#  #  #  !      !      !      !   !  ##   ##  !
# # # #  !===   !      !      !   !  # # # #  !===
##   ##  !      !      !      !   !  #  #  #  !
#     #  =====  =====  =====  =====  #     #  =====

         TTTTT OOOOO      22222   66666    1
           T   O   O          2   6       11
           T   O   O       222    66666    1
           T   O   O      2       6   6    1
           T   OOOOO      22222   66666   111
```