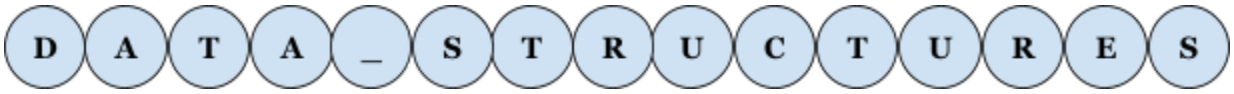
CS261 Data Structures

Assignment 1

Spring 2022

Python Fundamentals Review



Contents

Summary

For this assignment, you will write a few short Python functions. The primary objectives are to 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 solutions based on Gradescope output
- You know how to import and use classes that have been pre-written for you

For this course, we assume you are comfortable with:

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

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

General Instructions

1. The code for this assignment must be written in Python v3 and submitted to Gradescope before the due date specified on Canvas and in the Course Schedule. You may resubmit your code as many times as necessary (this is encouraged). 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 cases even though this work doesn'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 additional tests beyond Gradescope.

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

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 the comments in the skeleton code for guidance. The 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 are required to use an iterative solution**. Recursion is not permitted.

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 10 separate problems in this assignment. For each problem, you will write a Python function according to the provided specifications. "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. The StaticArray class has been pre-written for you, and is located in the file `static_array.py`

StaticArray is a very simple class that simulates the behavior of a fixed size array. It has only four methods, and contains code to support bracketed indexing (`[]`):

1) `init()` - Create a new static array that will store a fixed number of elements. Once the StaticArray is created, its size cannot be changed.
2) `set()` - Change the value of any element using its index.
3) `get()` - Read the value of any element using its index.
4) `length()` - Query number of elements in the array.

Please review the code and comments in the StaticArray class to better understand the available methods, their use, and input / output parameters. Note that since StaticArray is intentionally a very simple class, it does not possess the many capabilities typically associated with Python lists. You need to write your solutions using only the 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 for holding a single value, or a tuple holding two / three values, are allowed. It is also 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 by using the StaticArray class methods.

You may not use any imports beyond the ones included in the assignment source code.

min_max(arr: StaticArray) -> tuple:

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

The content of the input array must 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.

For full credit, the function must be implemented with O(N) complexity.

Example #1:

```
arr = StaticArray(5)
for i, value in enumerate([7, 8, 6, -5, 4]):
    arr[i] = value
print(arr)
result = min_max(arr)
print(f"Min: {result[0]: 3}, Max: {result[1]: 3}")
```

Output:

```
STAT_ARR Size: 5 [7, 8, 6, -5, 4]
Min:  -5, Max:   8
```

Example #2:

```
arr = StaticArray(1)
arr[0] = 100
print(arr)
result = min_max(arr)
print(f"Min: {result[0]}, Max: {result[1]}")
```

Output:

```
STAT_ARR Size: 1 [100]
Min: 100, Max: 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(arr)
    result = min_max(arr)
    print(f”Min: {result[0]: 3}, Max: {result[1]}”)
```

Output:

```
STAT_ARR Size: 3 [3, 3, 3]
Min:   3, Max: 3
STAT_ARR Size: 5 [-10, -30, -5, 0, -10]
Min: -30, Max: 0
STAT_ARR Size: 4 [25, 50, 0, 10]
Min:   0, Max: 50
```

fizz_buzz(arr: StaticArray) -> StaticArray:

Write a function that receives a StaticArray of integers and returns a new StaticArray object with the content of 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 will be the string 'fizz'.
2) If the number in the original array is divisible by 5, the corresponding element in the new array will 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 will be the string 'fizzbuzz'.
4) In all other cases, the element in the new array will have the same value as in the original array.

The content of the input array must 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.

For full credit, the function must be implemented with O(N) complexity.

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. The reversal must be done 'in place', meaning that the original input array will be modified, and you may not create another array (nor need to). You may assume that the input array will contain at least one element. You do not need to check for this condition.

For full credit, the function must be implemented with O(N) complexity.

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 of the elements are from the original array, but their position has shifted right or left `steps` number of times. The original array must not be modified.

If `steps` is a positive integer, the elements will be rotated to the right. Otherwise, rotation will be to the left. Please see the code examples below for additional details. You may assume that the input array will contain 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 must be able to rotate an array of at least 1,000,000 elements in a reasonable amount of time (under a minute).

For full credit, the function must be implemented with O(N) complexity.

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]:
    space = " " if steps >= 0 else ""
    print(f"{rotate(arr, steps)} {space}{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 the consecutive integers between `start` and `end` (inclusive).

For full credit, the function must be implemented with O(N) complexity.

Example #1:

```
cases = [
    (1, 3), (-1, 2), (0, 0), (0, -3),
    (-95, -89), (-89, -95)
]
for start, end in cases:
    print(f"Start: {start: 4}, End: {end: 4}, {sa_range(start, end)}")
```

Output:

```
Start:    1, End:    3, STAT_ARR Size: 3 [1, 2, 3]
Start:   -1, End:    2, STAT_ARR Size: 4 [-1, 0, 1, 2]
Start:    0, End:    0, STAT_ARR Size: 1 [0]
Start:    0, End:   -3, STAT_ARR Size: 4 [0, -1, -2, -3]
Start:  -95, End:  -89, STAT_ARR Size: 7 [-95, -94, -93, -92, -91, -90, -89]
Start:  -89, End:  -95, STAT_ARR Size: 7 [-89, -90, -91, -92, -93, -94, -95]
```

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 must return:

- 1 if the array is sorted in strictly ascending order.
- -1 if the list is sorted in strictly descending order.
- 0 otherwise.

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.

The original array must not be modified.

For full credit, the function must be implemented with O(N) complexity with no additional data structures (including Static Arrays) being created.

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
    result = is_sorted(arr)
    space = "  " if result >= 0 else " "
    print(f"Result:{space}{result}, {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: -1, 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: -1, STAT_ARR Size: 5 [100, 90, 0, -90, -200]
Result:  1, STAT_ARR Size: 1 ['apple']
```

find_mode(arr: StaticArray) -> tuple:

Write a function that receives a StaticArray that is sorted in order, either non-descending or non-ascending. The function will return, in this order, the mode (most-occurring value) of the array, and its frequency (how many times it appears).

If there is more than one value that has the highest frequency, select the one that occurs first in the array.

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.

For full credit, the function must be implemented with O(N) complexity with no additional data structures (including Static Arrays) being created.

Example #1:

```
test_cases = (
    [1, 20, 30, 40, 500, 500, 500],
    [2, 2, 2, 2, 1, 1, 1, 1],
    ["zebra", "sloth", "otter", "otter", "moose", "koala"],
    ["Albania", "Belgium", "Chile", "Denmark", "Egypt", "Fiji"]
)
for case in test_cases:
    arr = StaticArray(len(case))
    for i, value in enumerate(case):
        arr[i] = value
    mode, frequency = find_mode(arr)
    print(f"{arr}\nMode: {mode}, Frequency: {frequency}\n")
```

Output:

```
STAT_ARR Size: 7 [1, 20, 30, 40, 500, 500, 500]
Mode: 500, Frequency: 3

STAT_ARR Size: 8 [2, 2, 2, 2, 1, 1, 1, 1]
Mode: 2, Frequency: 4

STAT_ARR Size: 6 ['zebra', 'sloth', 'otter', 'otter', 'moose', 'koala']
Mode: otter, Frequency: 2

STAT_ARR Size: 6 ['Albania', 'Belgium', 'Chile', 'Denmark', 'Egypt', 'Fiji']
Mode: Albania, Frequency: 1
```

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 all duplicate values removed. The original array must not be modified.

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), and that elements of the input array are already in sorted order. You do not need to write checks for these conditions.

For full credit, the function must be implemented with O(N) complexity.

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, using the count sort algorithm. The original array must not be modified.

What is count sort? The name of the algorithm may seem confusing, if not deceptive - **it's really not a sort as you might commonly view one**. While the end result is a sorted version of the original contents, the means by which we arrive there may be a bit different.

For example, a sorted array can be generated if you know what values are present, and how many times they occur. Note that other functions in this assignment will help you determine how many different values could be in the array, as well as the range of possible values present.

Getting Started: As mentioned, we want to create a sorted array by counting the number of times each integer appears in the unsorted array. A possible first step would be to find the range of numbers in the array, and use that range to create a new 'count' array that tabulates the number of times each element is present in the original array. The 'count' array will then provide you with the information you need to generate a sorted array.

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 solution that can sort 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, shell sort, etc.) will not pass the largest test case of 5M elements.

For full credit, the function must be implemented with O(n+k) time complexity, where n is the number of elements and k is the range of input.

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
    before = arr if len(case) < 50 else 'Started sorting large array'
    print(f"Before: {before}")
    result = count_sort(arr)
    after = result if len(case) < 50 else 'Finished sorting large array'
    print(f"After: {after}")
```

Output:

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

sorted_squares(arr: StaticArray) -> StaticArray:

Write a function that receives a StaticArray where the elements are 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 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, 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.

For full credit, the function must be implemented with O(N) complexity.

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
```