CS/IT 200

Lab 1: Arrays

Recommended Due Date: Thursday, September 17, 11:59pm

Submission

- Submit all code and answers to questions in a single zip file to the Lab 1 assignment folder on Kodiak.

Goal: Analyze and compare the operations of dynamically-sized arrays.

Introduction

In recent lecture periods, we built the `DynamicArray` class in order to gain a better understanding of how lists work in Python. We may not be able to see the code that defines a Python list, but we can make educated guesses about what the methods of `list` might look like. Using these guesses, we can recognize why the efficiency of Python list methods are what they are. In this lab, we will continue this process, making further additions to our list-substitute, `DynamicArray`.

Part I

The `DynamicArray` class is provided in `dynamic_array.py`. Modify this class as described below. Make sure that each new function has a comment describing its purpose and what each parameter represents.

- Modify the `__getitem__` method so that the `DynamicArray` class can use negative indices just like a Python list. Make sure to raise exceptions if/when appropriate.
    - Hint: Remember that the `DynamicArray` may have extra capacity which you shouldn't be counting when trying to figure out the correct index.

- Implement the `__setitem__` method, which allows indexing notation to be used on the left side of an assignment statement. The header for this method is given below, where `k` is the index and `val` is the item to be assigned to that index. You only need to make this work for positive values of `k`; don't worry about negative indices. However, you should raise an exception if `k` is too large.

```
def __setitem__(self, k, val):
```

- Implement a `pop()` method that removes and returns the last element of the array. The method should also shrink the capacity *C* of the array by half any time the number of elements in the array goes below *C/4.* Make sure to raise exceptions if/when appropriate. Your solution must not call `remove()`.

- Implement the `__eq__` method, which will allow us to compare two `DynamicArrays` using the == operator. This method should return `True` if the contents of both

`DynamicArrays` are identical, including its length and the order of the elements. Otherwise, it should return `False`. The header for this method is given below, where `self` will be whichever `DynamicArray` is on the left side of the operator, and `other` will be the `DynamicArray` on the right side.

```
def __eq__(self, other):
```

Create `lab1.py` in which you demonstrate the use of these methods. The table below shows proper usage of each "special" (double underscore) method.

`__getitem__`	Any use of [] notation, except assigning to an index and using `del`, like: `print(myarray[3])` or `y = myarray[0] + myarray[-1]`
`__setitem__`	Assigning to an index using bracket notation, like: `myarray[0] = 'bears'`

`__eq__`	Comparison of two Dynamic Arrays using ==, like:
	`if myfirstarray == mysecondarray:`
	`  print('They match!')`

Part II

Two new methods for the `DynamicArray` class are included in `lab1b.py`. Copy those methods into the `DynamicArray` class in `dynamicarray.py`. You may wish to test these methods out before answering the questions in Part III.

There is no deliverable for this part.

Part III

Answer the questions below. Put your answers into a txt, Word, or PDF file.

Questions 1-4 refer to your work in Part I.

1. After your modification to `__getitem__`, what is the running time of the function? Is this the same or different than it was before?
2. What is the running time of your `__setitem__` function?
3. What is the running time of your `pop` function?
4. What is the running time of your `__eq__` function?

Questions 5-6 refer to the functions given in Part II.

5. Both functions from Part II do the exact same task in different ways. What do they do? What would be a good name for these methods? (Your answer should tell me the overall effect of the functions. Do not walk me through them line-by-line.)

6. What are the running times of the `lab1_v1` and `lab1_v2` functions? Assume that any use of the random module is $O(1)$.

What to Submit

Submit the following in a single zip file:

- Submit all code: `lab1.py` and `dynamic_array.py`
- Submit a file containing the answers to the questions in Part III.

Rubric

Grade	Overall	Part I	Part III
4	A 4 in Part I, and at least a 3 in Part III.	Code contains at most minor bugs that only occur in edge cases. Code is adequately commented.	Correct answers to all questions.
3	At least a 3 for Part I and a 2 for Part III.	One function does not behave as expected. Other functions have only minor bugs. Adequate comments. --OR— All functions have only minor bugs, but lacks sufficient comments.	Correct answers to all but one question.
2	At least a 2 for Part I and a 1 for Part III.	More than one function has a significant bug impairing its use or violating given instructions. May lack sufficient comments.	Correct answers to some questions.
1	At least a 1 for Part I and Part III.	Most functions have a significant bug impairing its use or violating given instructions.	Attempts all questions.
0	0 for Part I or no submission.	Most functions cause errors.	No submission