Ex. No: 8 30-05-2023	UIT2201 — Programming and Data Structures

Aim:

To execute the following programs and note the output.

PART – A

1. Implement List ADT using python and perform appropriate operations. Use special methods and proper docstrings.

Code:

```
import ctypes


class DynamicArray:
    """
        A dynamic array implementation that can store various types of
values.
    """

    def __init__(self, val):
        """
            Initializes a new DynamicArray instance.

            Args: val: An integer, list, tuple, or string to initialize the
array with. - If an integer is provided,
            it sets the initial capacity of the array. - If a list or tuple
is provided, it sets the initial capacity
            and populates the array with the values. - If a string is
provided, it sets the initial capacity and
            populates the array with individual characters.

            Raises:
                TypeError: If the provided value is not of type int, list,
tuple, or string.
        """
        if isinstance(val, int):
            self.n = 0
            self.capacity = val
            self.array = self.make_array(self.capacity)
        elif isinstance(val, list) or isinstance(val, tuple):
            self.n = len(val)
            self.capacity = len(val)
            self.array = val
        elif isinstance(val, str):
            self.n = len(val)
            self.capacity = len(val)
            self.array = [i for i in val]

    def make_array(self, size):
        """
```

```
        Creates and returns a new ctypes array of the specified size.

        Args:
            size: The size of the new array.

        Returns:
            A ctypes array of size 'size'.

    """
    temp = (size * ctypes.py_object)()
    return temp

def append(self, val):
    """
        Appends a value to the end of the dynamic array.

        Args:
            val: The value to be appended.
    """
    if self.n == self.capacity:
        self.resize(self.capacity * 2)
    self.array[self.n] = val
    self.n += 1

def resize(self, size):
    """
        Resizes the dynamic array to the specified size.

        Args:
            size: The new size of the array.

    """
    new_array = self.make_array(size)
    for i in range(self.n):
        new_array[i] = self.array[i]
    self.array = new_array
    self.capacity = size

def get_capacity(self):
    """
       Returns the current capacity of the dynamic array.

       Returns:
           The current capacity of the array.
   """
    return self.capacity

def __str__(self):
    """
        Returns a string representation of the dynamic array.

        Returns:
            A string representation of the dynamic array.
    """
    array_string = '<'
    for i in range(self.n):
        array_string += str(self.array[i])
        if i != self.n - 1:
            array_string += ','
    array_string += '>'
    return array_string
```

```
def __len__(self):
    """
        Returns the number of elements in the dynamic array.

        Returns:
            The number of elements in the array.
    """
    return len(self.array)

def __getitem__(self, idx):
    """
        Returns the element at the specified index.

        Args:
            idx: The index of the element to retrieve.

        Returns:
            The element at the specified index.

        Raises:
            IndexError: If the index is out of range.
    """
    if idx >= self.n:
        raise IndexError("Index out of range!")
    return self.array[idx]

def __setitem__(self, idx, value):
    """
        Sets the value at the specified index.

        Args:
            idx: The index of the element to set.
            value: The value to set at the specified index.

        Raises:
            IndexError: If the index is out of range.
    """
    if idx >= self.n:
        raise IndexError("Index out of range!")
    self.array[idx] = value

def insert(self, idx, elt):
    """
        Inserts an element at a given index in the array.

        Args:
            idx: The index to insert the element at.
            elt: The element to insert

        Raises:
            IndexError: If the index is out of range
    """
    if not 0 <= idx <= self.n:
        raise IndexError("Index out of range!")
    if self.n == self.capacity:
        self.resize(2 * self.capacity)
    for i in range(self.n, idx, -1):
        self.array[i] = self.array[i - 1]
    self.array[idx] = elt
    self.n += 1
```

```
    def delete(self, idx):
        """
            Deletes an element at a given index in the array.

            Args:
                idx: The index of the element to delete.

            Raises:
                IndexError: If the index is out of range
        """
        if not 0 <= idx < self.n:
            raise IndexError("Index out of range!")
        for i in range(idx, self.n - 1):
            self.array[i] = self.array[i + 1]
        self.n -= 1
        if self.n < self.capacity // 4:  # if the size of the array is
smaller than 25%, then shrink the array
            self.resize(self.capacity // 2)

    def extend(self, lst):
        """
            Extends a list of elements in the array
            Args:
                lst: list of elements to be extended
        """
        for elt in lst:
            self.append(elt)

    def __contains__(self, search_elt):
        """
            Checks if an element is present in the dynamic array
            Args:
                search_elt: The element to be searched in the array

            Returns:
                bool: True, if the search element is present in the array,
otherwise False.
        """
        for idx in range(0, self.n):
            if self.array[idx] == search_elt:
                return True
        return False

    def index(self, elt):
        """
            Checks if an element is present in the array and returns the
index of the element
            Args:
                elt: Element whose index is to be found out

            Returns:
                idx (int): If the element is found, index of the element is
returned, else -1.
        """
        for idx in range(0, self.n):
            if self.array[idx] == elt:
                return idx
        return -1

    def count(self, count_elt):
```

```
        """
            Returns the number of occurrences of an element in the dynamic
array
            Args:
                count_elt: Element whose number of occurrences is to be
found

            Returns:
                count (idx): The number of occurrences of an element in the
dynamic array
        """
        count = 0
        for idx in range(0, self.n):
            if self.array[idx] == count_elt:
                count += 1
        return count


# driver code
if __name__ == "__main__":
    import random

    d1 = DynamicArray(10)
    for i in range(5):
        d1.append(random.randint(0, 100))
    print(d1)
    d1.insert(2, -5)
    print(d1)
    d1.insert(6, 25)
    print(d1)
    d1.delete(6)
    print(d1)
    d1.extend([1, 2, 3, 4, 5, 6, 4])
    print(d1)
    print(2 in d1)
    print(d1.index(5))
    print(d1.count(4))
```

Inputs and Output:

```
<7,46,22,81,0>
<7,46,-5,22,81,0>
<7,46,-5,22,81,0,25>
<7,46,-5,22,81,0>
<7,46,-5,22,81,0,1,2,3,4,5,6,4>
True
10
2
```

2. Write a function (takes an integer n as an argument) that creates an empty list and append n random objects to that list. Your function should record the time taken T for these n appends, and return the average time T/n. Run the experiment for different (very large) values of n and note down the average time taken per 'append()' operation. How does this average time increase as n increases? Comment on your observation.

Code:

```
from dynamicarray import DynamicArray
from timeit import default_timer

if __name__ == "__main__":
    d1 = DynamicArray(30000)
    elts = [1, 5, 10, 100, 500, 1000, 5000, 10_000, 50_000, 1_00_000]
    for elt in elts:
        number_of_elements_to_append = elt
        start = default_timer()
        for _ in range(number_of_elements_to_append):
            d1.append(None)
        stop = default_timer()
        print(f"Size: {number_of_elements_to_append}; Time taken: {(stop -
start) / number_of_elements_to_append}"
              f"Original time: {stop-start}")
```

Complexity Analysis:

Ratio Analysis of Appending			
Number of elements appended	Execution time f(n)	n	f(n)/n
1	5.40E-06	1	5.40E-06
5	2.30E-06	5	4.60E-07
10	2.60E-06	10	2.60E-07
100	2.62E-05	100	2.62E-07
500	0.0001145	500	2.29E-07
1000	0.0002335	1000	2.33E-07
5000	0.0011711	5000	2.34E-07
10000	0.0022882	10000	2.29E-07
50000	0.0183549	50000	3.67E-07
100000	0.0323433	100000	3.23E-07

The time complexity of appending operation is O(n), where f(n) is taken as the execution time.

==========