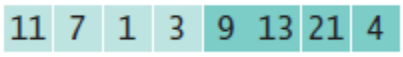
CPS 305, Lab 1
Computer Science Department
Ryerson University
Fall 2020

Q1) The Problem: You are given an array whose size is an even number, and you are to switch the first and the second half.

- Example
  - This array

  - will become

Calculate the average of both the halves.

Q2) Consider the task of rearranging all elements in an array so that the even numbers come first. Otherwise, the order doesn't matter. For example, the array

1 4 14 2 1 3 5 6 23 could be rearranged as

4 2 14 6 1 5 3 23 1

Q3) In Code Fragment 5.1, we perform an experiment to compare the length of a Python list to its underlying memory usage. Determining the sequence of array sizes requires a manual inspection of the output of that program. Redesign the experiment so that the program outputs only those values of *k* at which the existing capacity is exhausted. For example, on a system consistent with the results of Code Fragment 5.2, your program should output that the sequence of array capacities are 0, 4, 8, 16, 25, ...

```
1  import sys                                  # provides getsizeof function
2  data = [ ]
3  for k in range(n):                          # NOTE: must fix choice of n
4    a = len(data)                             # number of elements
5    b = sys.getsizeof(data)                   # actual size in bytes
6    print('Length: {0:3d}; Size in bytes: {1:4d}'.format(a, b))
7    data.append(None)                         # increase length by one
```

Code Fragment 5.1: An experiment to explore the relationship between a list's length and its underlying size in Python.

Q4) Our DynamicArray class, as given in Code Fragment 5.3, does not support use of negative indices with get item . Update that method to better match the semantics of a Python list.

```
1  import ctypes                                    # provides low-level arrays
2
3  class DynamicArray:
4    """A dynamic array class akin to a simplified Python list."""
5
6    def __init__(self):
7      """Create an empty array."""
8      self._n = 0                                  # count actual elements
9      self._capacity = 1                           # default array capacity
10     self._A = self._make_array(self._capacity)   # low-level array
11
12   def __len__(self):
13     """Return number of elements stored in the array."""
14     return self._n
15
16   def __getitem__(self, k):
17     """Return element at index k."""
18     if not 0 <= k < self._n:
19       raise IndexError('invalid index')
20     return self._A[k]                            # retrieve from array
21
22   def append(self, obj):
23     """Add object to end of the array."""
24     if self._n == self._capacity:               # not enough room
25       self._resize(2 * self._capacity)           # so double capacity
26     self._A[self._n] = obj
27     self._n += 1
28
29   def _resize(self, c):                          # nonpublic utitity
30     """Resize internal array to capacity c."""
31     B = self._make_array(c)                      # new (bigger) array
32     for k in range(self._n):                     # for each existing value
33       B[k] = self._A[k]
34     self._A = B                                  # use the bigger array
35     self._capacity = c
36
37   def _make_array(self, c):                      # nonpublic utitity
38     """Return new array with capacity c."""
39     return (c * ctypes.py_object)( )             # see ctypes documentation
```

Code Fragment 5.3: An implementation of a DynamicArray class, using a raw array from the ctypes module as storage.

Q5) Our implementation of insert for the DynamicArray class, as given in Code Fragment 5.5, has the following inefficiency. In the case when a resize occurs, the resize operation takes time to copy all the elements from an old array to a new array, and then the subsequent loop in the body of insert shifts many of those elements. Give an improved implementation of the insert method, so that, in the case of a resize, the elements are shifted into their final position during that operation, thereby avoiding the subsequent shifting.

```
1   def insert(self, k, value):
2     """Insert value at index k, shifting subsequent values rightward."""
3     # (for simplicity, we assume 0 <= k <= n in this verion)
4     if self._n == self._capacity:                    # not enough room
5       self._resize(2 * self._capacity)               # so double capacity
6     for j in range(self._n, k, -1):                  # shift rightmost first
7       self._A[j] = self._A[j-1]
8     self._A[k] = value                               # store newest element
9     self._n += 1
```

Code Fragment 5.5: Implementation of insert for our DynamicArray class.