**Chapter 5:**

**R-5.3 Modify the experiment from Code Fragment 5.1 in order to demonstrate that Python’s list class occasionally shrinks the size of its underlying array when elements are popped from a list.**

**import sys**

**n = 20**

**data = []**

**for i in range(n):**

**data.append(None)**

**for k in range(n):**

**a = len(data)**

**b = sys.getsizeof(data)**

**print("Length: {0:3d}; Size in bytes: {1:4d}".format(a, b))**

**data.pop(0)**

**Length: 20; Size in bytes: 248**

**Length: 19; Size in bytes: 248**

**Length: 18; Size in bytes: 248**

**Length: 17; Size in bytes: 248**

**Length: 16; Size in bytes: 248**

**Length: 15; Size in bytes: 248**

**Length: 14; Size in bytes: 248**

**Length: 13; Size in bytes: 248**

**Length: 12; Size in bytes: 248**

**Length: 11; Size in bytes: 184**

**Length: 10; Size in bytes: 184**

**Length: 9; Size in bytes: 184**

**Length: 8; Size in bytes: 184**

**Length: 7; Size in bytes: 152**

**Length: 6; Size in bytes: 152**

**Length: 5; Size in bytes: 120**

**Length: 4; Size in bytes: 120**

**Length: 3; Size in bytes: 120**

**Length: 2; Size in bytes: 120**

**Length: 1; Size in bytes: 88**

**R-5.7 Let A be an array of size n ≥ 2 containing integers from 1 to n−1, inclusive, with exactly one repeated. Describe a fast algorithm for finding the integer in A that is repeated.**

**def repeated(A):**

**#runs through the array**

**for x in range(0, len(A)):**

**for y in range(x+1, len(A)):**

**if(A[x] == A[y]):**

**print("The duplicate number is: ", A[y])**

**#A is greater than 2**

**#Integers 1 to n-1 and one number repeated**

**print(repeated([1,2,3,3,4,5]))**

**Chapter 6:**

**R-6.1 What values are returned during the following series of stack operations, if executed upon an initially empty stack? push(5), push(3), pop(), push(2), push(8), pop(), pop(), push(9), push(1), pop(), push(7), push(6), pop(), pop(), push(4), pop(), pop().**

**5**

**(9 pushes and 8 pops = 1 value left)**

**R-6.5 Implement a function that reverses a list of elements by pushing them onto a stack in one order, and writing them back to the list in reversed order.**

**#user gives us a list**

**def reverse\_list(listy):**

**stack = []**

**# this makes the stack out of the user list**

**for a in range(len(listy)):**

**#python 3 does not have push function for stacks (it is append)**

**stack.append(listy[a])**

**#**

**for i in range(len(listy)):**

**listy[i] = stack.pop()**

**return listy**

**if \_\_name\_\_ == '\_\_main\_\_':**

**boop = [1,2,3,4,5,6,7,8,9]**

**print(reverse\_list(boop))**

**[9, 8, 7, 6, 5, 4, 3, 2, 1]**

**C-6.23 Suppose you have three nonempty stacks R, S, and T. Describe a sequence of operations that results in S storing all elements originally in T below all of S’s original elements, with both sets of those elements in their original order. The final configuration for R should be the same as its original configuration. For example, if R = [1,2,3], S = [4,5], and T = [6,7,8,9], the final configuration should have R = [1,2,3] and S = [6,7,8,9,4,5].**

R = [1,2,3]

S = [4,5]

T = [6,7,8,9]

n = len(S)

x = len(T)

def stacky(R,S,T):

stack = []

for a in range(len(T)):

stack.append(T[a])

for i in range(len(T)):

T[i] = stack.pop()

for a in range(n):

R.append(S.pop())

for a in range(x):

S.append(T.pop())

for a in range(n):

S.append(R.pop())

return R, S

print(stacky(R,S,T))