

**Computer  
Science****COMPSCI 105 S2 C - Assignment 2****Due date: Friday, 27<sup>th</sup> October 6 pm**

*100 marks in total = 7.5% of the final grade*

**Assessment**

- Due: Friday, 27<sup>th</sup> October 2016 (6:00 pm)
- Worth: 7.5% of your final mark

**Resources and Submission**

Q1: Resources – `Node.py`, `OrderedList.py`, `LinkedListIterator.py`, and `A2Q1.py`. Submission – `OrderedList.py`.

Q2: Resources – `A2Q2.py`. Submission – `Fibonacci.py` and `Fibonacci_Iterator.py`.

Q3: Resources – `HashTable.py`, `A2Q3.py`, Submission – `HashTable.py`.

**Part I of Assignment****Aims of Part I of Assignment**

Understanding and solving problems using:

- Linked lists
- Recursion
- Hash tables

***Q1. Implementing the `add()`, `search()` and `remove()` functions for an ordered linked list recursively (15 Marks)***

In Lecture 20 we discussed the implementation of an ordered linked list. An ordered linked list maintains items sorted in ascending order from smallest (at the head of the list) to largest. This ordering is maintained even when new items are added to the list. Three of the functions we discussed were the `add()`, `remove()` and `search()` functions, which add, remove and search for an item within an ordered linked list respectively. We implemented all three of these functions iteratively.

In this exercise you will need to alter the implementations of the `add()`, `remove()` and `search()` functions so that they run recursively. I have begun the process for you as follows:

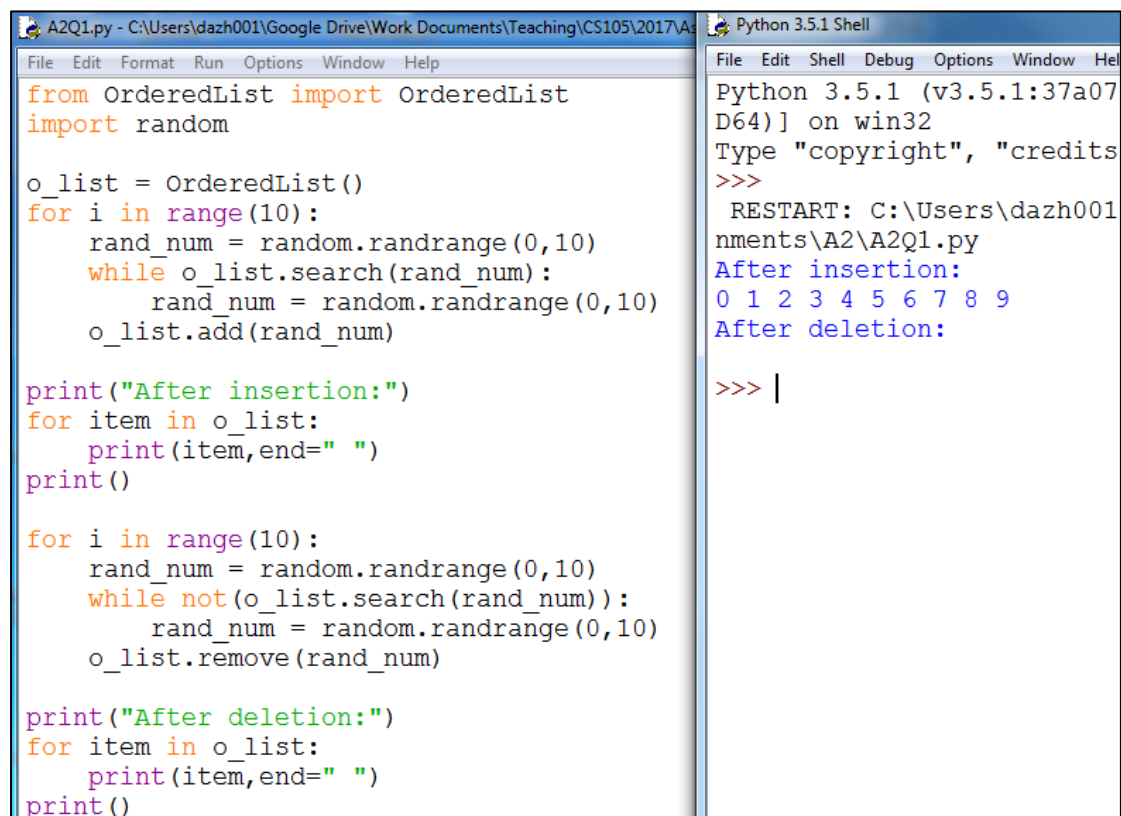
```
def add(self, item):
    new_node = Node(item)
    curr = self.__head
    prev = None
    self.add_recursive(new_node, curr, prev)
```

```
def search(self, item):
    curr = self.__head
    return self.search_recursive(item, curr)
```

```
def remove(self, item):
    curr = self.__head
    prev = None
    self.remove_recursive(item, curr, prev)
```

You will need to implement the `add_recursive()`, `search_recursive()`, and `remove_recursive()` functions, which as their names suggest, **must be recursive functions**. No marks will be given for non-recursive implementations.

I have provided a file for you to use to test your ordered linked list implementation - A2Q1.py. You should get the same output regardless of whether you use the ordered linked list implementation discussed in class or the recursive implementation you complete for this assignment. An example of it running is shown below:



```
A2Q1.py - C:\Users\dazh001\Google Drive\Work Documents\Teaching\CS105\2017\As
Python 3.5.1 Shell

File Edit Format Run Options Window Help
from OrderedList import OrderedList
import random

o_list = OrderedList()
for i in range(10):
    rand_num = random.randrange(0,10)
    while o_list.search(rand_num):
        rand_num = random.randrange(0,10)
    o_list.add(rand_num)

print("After insertion:")
for item in o_list:
    print(item, end=" ")
print()

for i in range(10):
    rand_num = random.randrange(0,10)
    while not(o_list.search(rand_num)):
        rand_num = random.randrange(0,10)
    o_list.remove(rand_num)

print("After deletion:")
for item in o_list:
    print(item, end=" ")
print()

Python 3.5.1 (v3.5.1:37a07
D64)] on win32
Type "copyright", "credits
>>>
RESTART: C:\Users\dazh001
nments\A2\A2Q1.py
After insertion:
0 1 2 3 4 5 6 7 8 9
After deletion:

>>> |
```

Note that the assumptions we made for our implementation of the ordered list in lectures still hold true for the recursive implementation here. When you add an item you can assume that it is not already present in the list. When you remove an item you can assume that the item is present in the list.

### Marking Scheme for Question 1

The <code>add_recursive()</code> function is implemented correctly	5 marks
The <code>search_recursive()</code> function is implemented correctly	5 marks
The <code>remove_recursive()</code> function is implemented correctly	5 marks
No docstring at the top of <code>OrderedList.py</code> with student name and ID	-2 marks

### Q2. Implementing a Fibonacci Iterator

(15 Marks)

We discussed the Fibonacci sequence in Lecture 24. It is a sequence of numbers where the first two numbers are both 1, with subsequent numbers being the sum of the previous two numbers in the sequence. The first 10 Fibonacci numbers are shown below:

1 1 2 3 5 8 13 21 34 55

In this exercise you will develop two classes that will enable you to iterate through the sequence of Fibonacci numbers with a `for` loop – `Fibonacci.py` and `Fibonacci_Iterator.py`.

```
A2Q2.py - C:\Users\dazh001\Google Drive\Work Documents\Teaching\CS105\2017\Assignments\A2\A2Q2.py (3.5.1)
File Edit Format Run Options Window Help
from Fibonacci import Fibonacci

def main():
    user_input = input("Please enter a number between 1 and 50: ")
    while not(user_input.isdigit()) or int(user_input) < 1 or \
        int(user_input) > 50:
        user_input = input("Please enter a number between 1 and 50: ")
    n = int(user_input)
    print()
    print("The first",n,"term(s) of the Fibonacci sequence:")
    fib_obj = Fibonacci(n)
    for i in fib_obj:
        print(i, end=" ")
    print()

main()

Python 3.5.1 Shell
File Edit Shell Debug Options Window Help
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:54:25) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more in
formation.
>>>
RESTART: C:\Users\dazh001\Google Drive\Work Documents
\Teaching\CS105\2017\Assignments\A2\A2Q2.py
Please enter a number between 1 and 50: 1

The first 1 term(s) of the Fibonacci sequence:
1
>>>
RESTART: C:\Users\dazh001\Google Drive\Work Documents
\Teaching\CS105\2017\Assignments\A2\A2Q2.py
Please enter a number between 1 and 50: 2

The first 2 term(s) of the Fibonacci sequence:
1 1
>>>
RESTART: C:\Users\dazh001\Google Drive\Work Documents
\Teaching\CS105\2017\Assignments\A2\A2Q2.py
Please enter a number between 1 and 50: 10

The first 10 term(s) of the Fibonacci sequence:
1 1 2 3 5 8 13 21 34 55
Ln: 21 Col: 4
```

I have provided a file for you to test your Fibonacci classes – A2Q2.py. Three examples of this code running are provided in the image on the previous page, printing out the first 1, 2 and 10 numbers in the Fibonacci sequence.

### Marking Scheme for Question 2

Iteration handled by two classes – Fibonacci.py and Fibonacci_Iterator.py	5 marks
Correct sequence is printed when $n = 1$ and $n = 2$	5 marks
Correct sequence is printed when $n > 2$	5 marks
No docstring at the top of both classes submitted	-2 marks

<b>Q3. Implementing a hash table that uses double hashing</b>	<b>(25 Marks)</b>
---	-------------------

In Lectures 26 and 27 we discussed the implementation of a hash table that used linear probing for collision resolution. In this exercise you will modify this hash table class so that it uses double hashing for collision resolution. The hash table will also increase its size whenever its load factor is 0.75 or more. The hash table will contain key-data pairs. You can assume that keys will be integer values and data strings.

Your hash table class will be called `HashTable`. The constructor for this class has no parameter. Five instance variables will be initialized:

- `size` – the size of the hash table
- `count` – the number of items in the hash table
- `slots` – a python list with `size` number of elements.
- `data` – a python list with `size` number of elements.
- `deleted` – a string consisting of the null character `"\0"`.

You will need to modify the `put()` and `delete()` functions discussed in class. You will also need to implement a number of your own functions. This should be done in steps as listed below:

1. Modify the `put()` and `delete()` functions so that the variable `count` is altered appropriately whenever a key-data pair is added to or deleted from the hash table. Change the `__len__()` function so that it returns this count.
2. Your hash table will use the remainder method as discussed in lectures:  $key \% size$ . For collision resolution you will need to implement a second hash function:  $size - (key \% (size - 1) + 1)$ . You will need to alter the implementation of the `rehash()` function to use this second hash function.
3. You will need to implement a function that calculates the load factor of the hash table. Remember that the load factor is calculated using the formula:  
 $\lambda = count / size$ .
4. To increase the size of the hash table you want to find a prime number that approximately doubles its current size. For example if your hash table has a size of 7, you will be looking at increasing the size of the table to 13. You will need to implement a function that finds the largest prime number in the range  $n + 1$  to  $2n$ .
5. You will need to implement a `resize()` function that increases the size of the hash table appropriately and rehashes all existing key-data pairs into the appropriate slots in the larger hash table. You will need to modify the `put()` function to use this `resize()` function appropriately.

## COMPSCI 105 S2 C - Assignment Two

5 of 7

I have provided a file for you to test your hash table class – A2Q3.py. An example of this program running is shown below.

```
RESTART: C:/Users/dazh001/Google Drive/Work Documents/Teaching/CS105/2017/Assignments/A2/A2Q3.py
Test 1
-----
Insertion:
key: 0 data: aardvark
{0:aardvark, , , , , } count: 1
key: 1 data: bull
{0:aardvark, 1:bull, , , , } count: 2
key: 2 data: cat
{0:aardvark, 1:bull, 2:cat, , , , } count: 3
key: 3 data: dog
{0:aardvark, 1:bull, 2:cat, 3:dog, , , , } count: 4
key: 4 data: elephant
{0:aardvark, 1:bull, 2:cat, 3:dog, 4:elephant, , , , } count: 5
key: 5 data: frog
{0:aardvark, 1:bull, 2:cat, 3:dog, 4:elephant, 5:frog, , , , , } count: 6
key: 6 data: goat
{0:aardvark, 1:bull, 2:cat, 3:dog, 4:elephant, 5:frog, 6:goat, , , , , } count: 7
key: 7 data: horse
{0:aardvark, 1:bull, 2:cat, 3:dog, 4:elephant, 5:frog, 6:goat, 7:horse, , , , , } count: 8

Deletion:
{, , , , , , , , , , } count: 0

Test 2
-----
Insertion:
key: 602 data: aardvark
{602:aardvark, , , , , } count: 1
key: 450 data: bull
{602:aardvark, , 450:bull, , , , } count: 2
key: 748 data: cat
{602:aardvark, , 450:bull, , , , 748:cat} count: 3
key: 689 data: dog
{602:aardvark, , 450:bull, 689:dog, , , , 748:cat} count: 4
key: 658 data: elephant
{602:aardvark, , 450:bull, 689:dog, 658:elephant, , , , 748:cat} count: 5
key: 247 data: frog
{689:dog, , , , 602:aardvark, 247:frog, , , , 748:cat, 450:bull, , , , 658:elephant, , , } count: 6
key: 869 data: goat
{689:dog, , , , 602:aardvark, 247:frog, , , , 748:cat, 450:bull, , , , 658:elephant, 869:goat, , } count: 7
key: 779 data: horse
{689:dog, , , , 602:aardvark, 247:frog, , , , 748:cat, 450:bull, , , , 658:elephant, 869:goat, 779:horse} count: 8
```

### Marking Scheme for Question 3

Step 1	3 marks
Step 2	4 marks
Step 3	3 marks
Step 4	6 marks
Step 5	9 marks
No docstring at the top of HashTable.py with student name and ID	-2 marks