Midterm Exam #2 Solution

CSE2050: Data Structure and Object Oriented Design

Spring 2018

Dr. Wei Wei

First Name:

Last Name:

PeopleSoft ID:

Instructions

1. Put your name and PeopleSoft ID on the exam books.
2. The exam is closed book, closed notes, closed neighbor.
3. You have 75 minutes to complete the exam. Be a smart exam taker – if you get stuck on one problem go on to another problem. Also, don't waste your time giving irrelevant (or not requested) details or material.
4. The total number of points for each question is given in parenthesis. There are 100 points total.
5. Show all your work. Partial credit is possible for an answer, but only if you show the intermediate steps in obtaining the answer.
6. Good luck!

1. (20 points) Write down the outputs of the following code.

(a)

```
def fun(n):
    if n == 1:
        return n
    if n % 2 == 0:
        print(fun(n//2), end = ' ')
        return fun(n//2)
    else:
        print(fun(n-1), end = ' ')
        return fun(n-1)

fun(6)
```

1 1 1 1 1 1 1

(b)

```
def fun1(n):
    if n > 0:
        fun1(n//2)
        print(n, end = ' ')
        fun1(n//2)

fun1(8)
```

1 2 1 4 1 2 1 8 1 2 1 4 1 2 1

(c)

```
def f():
    a = 0
    b = 1
    while True:
        yield a
        a, b = b, a + b

for a in f():
    if a < 20:
        print(a, end = ' ')
    else:
        break
print()
g = f()
for i in range(10):
    print(next(g), end = ' ')
```

0 1 1 2 3 5 8 13
0 1 1 2 3 5 8 13 21 34

(d)

```
class Person:
    def __init__(self, name, ssn):
        self._name = name
        self._ssn = ssn
    def __eq__(self, other):
        return isinstance(other, Person) and self._ssn == other._ssn
    def __hash__(self):
        return hash(self._ssn)
    def __str__(self):
        return str('(' + self._name) + ':' + str(self._ssn) + ')'

p = Person('Foo Bar', 123456789)
q = Person('Fake Name', 123456789)
s = set()
s.add(p)
s.add(q)
for person in s:
    print(person)
```

(Foo Bar:123456789)

2. (20 points) Find the floor value of a key in a sorted list

Given a list of n integers stored in a non-decreasing order, find floor value of input 'key'. If the key is in the list, then key will be returned. Otherwise, the floor value is defined as the largest number in the list that is smaller than the key. If such a number does not exist in the list, we return None.

For example, L = [-1, 2, 3, 5, 6, 8, 9, 10] and key = 7, we should return 6 as outcome.
On the other hand, if L = [1, 2, 3] and key = 0, we should return None.
More examples:
L = [-2, -1, 0, 0, 0, 0, 2, 3, 4, 6], key = 0, and the output is 0.
L = [-3, -1, 0, 2, 4, 5, 6, 7, 8, 12, 20], key = 100, and the output is 20.

Implement the function floorvalue(L, key), so that code like the following can work correctly.
print(floorvalue([-2, -1, 2, 3, 4, 6]), 5)

The running time of the function needs to be O(log n), where n is the size of the input list. To help you get started, below is the code for binary search from the textbook.

```
def bs(L, item):
    left, right = 0, len(L)
    while right - left > 1:
        median = (right + left)//2
        if item < L[median]:
            right = median
        else:
            left = median
    return right > left and L[left] == item
```

Finish the code below.

```
def floorvalue(L, item):
    left, right = 0, len(L)
    while right - left > 1:
        median = (right + left)//2
        if item < L[median]:
            right = median
        else:
            left = median
    if right > left and L[left] <= item: return L[left]
```

3. (20 points) Fill in the missing code to generate the printout on the next page.

```
from random import randrange, seed
import math
class Piece:
    def area(self):
        raise NotImplementedError

    def __lt__(self, other):
        if isinstance(self, Circle) and isinstance(other, Rectangle):
            ## add your code here
            return True
        elif isinstance(self, Circle) and isinstance(other, Circle):
            ## add your code here
            return self.area() > other.area()
        elif isinstance(self, Rectangle) and isinstance(other, Rectangle):
            ## add your code here
            return self.area() < other.area()
        else:
            ## add your code here
            return False
    def __str__(self):
        raise NotImplementedError

class Circle(Piece):
    def __init__(self, radius):
        ## add your code here
        self.radius = radius

    def area(self):
        return math.pi*self.radius*self.radius

    def __str__(self):
        ## add your code here
        return 'C: ' + str(self.radius) + ' '+ str(self.area())


class Rectangle(Piece):
    def __init__(self, length, width):
        ## add your code here
        self.length = length
        self.width = width

    def area(self):
        ## add your code here
        return self.length*self.width

    def __str__(self):
```

```
        ## add your code here
        return 'R: ' + str(self.length) + ' '+ str(self.width) + ' ' + str(self.area())

seed(15)

circles = [Circle(1 + randrange(10))  for i in range(5)]

rectangles = [Rectangle(1 + randrange(10), 1 + randrange(10)) for i in range(5)]

pieces = circles + rectangles

for b in pieces:
    print(b)

print("---------------------------")
pieces.sort()
for b in pieces:
    print(b)
```

```
C: 4 50.26548245743669
C: 1 3.141592653589793
C: 9 254.46900494077323
C: 1 3.141592653589793
C: 3 28.274333882308138
R: 4 1 4
R: 1 3 3
R: 6 4 24
R: 2 6 12
R: 8 6 48
---------------------------
C: 9 254.46900494077323
C: 4 50.26548245743669
C: 3 28.274333882308138
C: 1 3.141592653589793
C: 1 3.141592653589793
R: 1 3 3
R: 4 1 4
R: 2 6 12
R: 6 4 24
R: 8 6 48
```

4. (20 points) Count the number of inversions in a list.

In this problem, we will count the number of inversions in a list. We will do so by making minor changes to the following mergesort code.

For index i and j, where i < j, if we have L[j] > L[i], then we say this is an inversion in list L. We will count the number of inversions in a list. For example, the number of inversions in the list L = [4, 3, 2, 1] is 6; the number of inversions in the following list L = [1, 2, 3, 4] is 0, and the number of inversions in the following list L = [1, 2, 4, 3] is 1.

To help you get started, below are the code for mergecount.

```
def mergesort(L):
    # Base Case:
    if len(L) < 2:
        return

    # Divide!
    mid = len(L)//2
    A = L[:mid]
    B = L[mid:]

    # Conquer!
    mergesort(A)
    mergesort(B)

    # Combine!
    merge(A, B, L)

def merge(A, B, L):
    i = 0 # index into A
    j = 0 # index into B
    while i < len(A) and j < len(B):
        if A[i] <= B[j]:
            L[i+j] = A[i]
            i = i + 1
        else:
            L[i+j] = B[j]
            j = j + 1
    # Add any remaining elements once one list is empty
    L[i+j:] = A[i:] + B[j:]
```

```
### Observe how we change the code from above to return the number of inversions
def sortandcount(L):
    # Base Case:
    if len(L) < 2:
        return 0
    # Divide!
    mid = len(L)//2
    A = L[:mid]
    B = L[mid:]

    # Conquer!
    count_a = sortandcount(A)
    count_b = sortandcount(B)

    # Combine!
    count = mergeandcount(A, B, L)
    return count_a + count_b + count

def mergeandcount(A, B, L):
    i = 0 # index into A
    j = 0 # index into B
    count = 0
    while i < len(A) and j < len(B):
        if A[i] <= B[j]:
            L[i+j] = A[i]
            i = i + 1
        else:
            ### add one line of code here

             count += len(A) - i

            L[i+j] = B[j]
            j = j + 1
    # Add any remaining elements once one list is empty
    L[i+j:] = A[i:] + B[j:]
    return count
```

Describe the running time of sortandcount(L) in terms of big-O notations, where n is the number of items in the list L.

O(n log n)

5. (20 points) Write down the output of the following code. Note there is a print statement in the _bucket method. Also hash(key) returns key itself if key is an integer.

```
class Entry:
    def __init__(self, key, value):
        self.key = key
        self.value = value

    def __str__(self):
        return str(self.key) + ' : ' + str(self.value)

class Mapping:
        # Child class needs to implement this!
    def get(self, key):
        raise NotImplementedError

    # Child class needs to implement this!
    def put(self, key, value):
        raise NotImplementedError

    # Child class needs to implement this!
    def __len__(self):
        raise NotImplementedError

    # Child class needs to implement this!
    def _entryiter(self):
        raise NotImplementedError

    def __iter__(self):
        return (e.key for e in self._entryiter())

    def values(self):
        return (e.value for e in self._entryiter())

    def items(self):
        return ((e.key, e.value) for e in self._entryiter())

    def __contains__(self, key):
        #print(self, "contains", key)
        try:
            return (self.get(key) is not None)
        except KeyError:
            return False

    def __getitem__(self, key):
        return self.get(key)

    def __setitem__(self, key, value):
```

```
        self.put(key, value)

    def __str__(self):
        return "{%s}" % (", ".join([str(e) for e in self._entryiter()]))

class ListMapping(Mapping):
    def __init__(self):
        self._entries = []

    def put(self, key, value):
        e = self._entry(key)
        if e is not None:
            e.value = value
        else:
            self._entries.append(Entry(key, value))

    def get(self, key):
        e = self._entry(key)
        if e is not None:
            return e.value
        else:
            raise KeyError

    def _entry(self, key):
        for e in self._entries:
            if e.key == key:
                return e
        return None

    def _entryiter(self):
        return (e for e in self._entries)

    def __len__(self):
        return len(self._entries)

class HashMapping(Mapping):
    def __init__(self, size = 2):
        self._size = size
        self._buckets = [ListMapping() for i in range(self._size)]
        self._length = 0

    def _entryiter(self):
        return (e for b in self._buckets for e in b._entryiter())

    def get(self, key):
        b = self._bucket(key)
        return b[key]
```

```
    def put(self, key, value):
        b = self._bucket(key)
        if key not in b:
            self._length += 1
        b[key] = value

        # Check if we need more buckets.
        if self._length > self._size:
            self._double()

    def __len__(self):
        return self._length

    def _bucket(self, key):
        print(key, "to", hash(key) %self._size)
        return self._buckets[hash(key) % self._size]

    def _double(self):
        # Save the old buckets
        oldbuckets = self._buckets
        # Reinitialize with more buckets.
        self.__init__(self._size * 2)
        for bucket in oldbuckets:
            for key, value in bucket.items():
                self[key] = value

map1 = HashMapping()
print(3 in map1)
for i in range(4):
    map1[i] = i

print(map1)
print(3 in map1)
```

3 to 1
False
0 to 0
1 to 1
2 to 0
0 to 0
2 to 2
1 to 1
3 to 3
{0 : 0, 1 : 1, 2 : 2, 3 : 3}
3 to 3
True