15-112
Summer 2019 Final Exam
June 21, 2019

Name:

Andrew ID:

Recitation Section:

- You may not use any books, notes, or electronic devices during this exam.
- Show your work on the exam to receive credit.
- You may use the backs of pages as scratch paper. Nothing written on the back of any pages will be graded.
- You may complete the problems in any order you'd like; you may wish to start with the last three problems, which are worth most of the credit.
- All code samples run without crashing. Assume any imports are already included as required.
- Good luck!

Don't write anything in the table below.

Question	Points	Score
1	10	
2	10	
3	10	
4	15	
5	15	
6	18	
7	22	
Total:	100	

1. (10 points) **Code Tracing**

   Indicate what each will print. Place your answer (and nothing else) in the box below each block of code.

```
def ct(lst):
    d1 = dict()
    for i in range(len(lst)):
        If lst[i] not in d1:
              d1[lst[i]] = 0
        d1[lst[i]] += 1
    print(d1)
    d2 = dict()
    for key in d1:
        d2[d1[key]] = key
    print(d2)
    for key in d2:
        d1[key] = d2[key]*2
    for key in d1:
        print("key:", key, "value:", d1[key])
    return d1.get(4, 1)

lst = [1, 5, 1, 1, 2, 5]
print(ct(lst))
```

2. (10 points) **Reasoning Over Code**

   For each function, find values of the parameters so that the function will return True. Place your answer (and nothing else) in the box below each block of code.

```
def roc(L, depth=0):
        assert(len(set(L)) == len(L))
        assert(len(L) % 4 == 0 and len(L) // 3 == 2)
        assert(L[0] + L[-1] == 10)
        assert(L[0] < L[1])
        if depth < 4:
                L = roc(L[2:] + L[:2], depth + 1)
        return True
```

3. **Short Answer**

Answer each of the following *very briefly*.

(a) (3 points) State the big O of selection sort, and in a couple of words, briefly explain why.

(b) (2 points) In a couple of words, explain why this code will crash.

```
a = (419,420)
a[0] = 420
a[1] = 421
```

(c) (3 points) List 3 style guidelines from the 112 website, and a small code snippet violating the three guidelines (no more than 10 lines of code).

(d) (2 points) In wordsearch, we wrote multiple helper functions in our implementation. Name one of them (name does not have to be exact), and what its purpose was.

4. (15 points) **Big O:** What is the Big-O runtime of each of the following in terms of N?

   Note: The big O's of each line will be graded as well - your overall big O will be graded based on the answers you put for each line. Note that for loops, write the big O based on the amount of times the loop will run.

Built-in Big-O Runtimes					
Sets		Lists			
`len(S)`	$O(1)$	`L[i:j:k]`	$O((j-i)/k)$	`len(lst)`	$O(1)$
`item in S`	$O(1)$	`L.pop(0)`	$O(N)$	`item in lst`	$O(N)$
`S.add(item)`	$O(1)$	`L.append(item)`	$O(1)$	`lst.insert(0, elem)`	$O(N)$
`S.remove(item)`	$O(1)$			`sorted(L)`	$O(NlogN)$

```
def bigOh1(L):                                  # Big O
    # assume L is an NxN (square) 2d list
    N = len(L)                                  # _____
    s = 0                                       # _____
    for row in L:                               # _____
        a = sorted(row)                         # _____
        s += 1                                  # _____
    return s                                    # _____
```

```
def bigOh2(L):                                  # Big O
    for i in range(len(L)):                     # _____
        for j in range(len(L)):                 # _____
            for k in range(len(L)):             # _____
                if i == j == k:                 # _____
                    return False                # _____
    return True
```

```
def bigOh3(S): #S is a set, not a string        # Big O
    L = []                                      # _____
    for elem in S:                              # _____
        L.insert(0, elem)                       # _____
        for i in range(len(L)):                 # _____
            L.append(5)                         # _____
    elem0 = L.pop(0)                            # _____
    elem1 = L.pop(0)                            # _____
    resultL = ([elem0, elem1])*(len(s)**2)      # _____
    return sorted(result)                       # _____
```

```
def bigOh4(L):                          # Big O
    newL = []                           # _____
    while(L != []):                     # _____
        newL.append(L.pop(0))           # _____
        L = L[:len(L)/4]                # _____
        newL = sorted(newL)             # _____
    return newL                         # _____
```

5. (15 points) **Free Response: Fish and Shark classes**

Write the classes Fish and Shark so that the following test code runs without errors. Do not hardcode against the values used in the testcases, though you can assume the testcases cover the needed functionality. You must use proper object-oriented design, including good inheritance, or you will lose points. The test cases continue on page 8 - you can write your solution on page 8 and 9.

```
# a fish has a name, a species, and a location
nemo = Fish("Nemo", "clownfish", "anemone")
assert(str(nemo) == "Nemo is a clownfish in the anemone.")
dory = Fish("Dory", "tang fish", "coral reef")
assert(str(dory) == "Dory is a tang fish in the coral reef.")
marlin = Fish("Marlin", "clownfish", "anemone")
assert(str(marlin) == "Marlin is a clownfish in the anemone.")
assert(marlin != nemo)
assert(dory != nemo)

# fish can swim and get lost
assert(nemo.isLost() == False)
nemo.swim()
assert(str(nemo) == "Nemo is a clownfish in the open ocean.")
assert(nemo.isLost() == True)
dory.swim()
assert(dory.isLost() == True)
assert(marlin.isLost() == False)

# fish travel in schools
school = set()
assert(nemo not in school)
school.add(nemo)
assert(dory not in school)
school.add(dory)
assert(marlin not in school)

# sharks are a species of fish
bruce = Shark("Bruce", "caves")
assert(isinstance(bruce, Fish))
assert(str(bruce) == "Bruce is a shark in the caves.")

changedBruce = Shark("Bruce", "caves")
changedBruce.swim()
assert(str(changedBruce) == "Bruce is a shark in the open ocean.")
assert(changedBruce.isLost() == nemo.isLost())
assert(changedBruce.meetFish() == "Fish are friends not food!")
school.add(changedBruce)
```

```
assert(changedBruce != bruce)
assert(bruce not in school)
assert(changedBruce in school)
print("done!")
```

Additional Space 1 for Answer to Question 6

6. (18 points) **Free Response: completeKingsTour - the comeback of isKingsTour**

Background: Recall that on a board, a king can move to any adjacent square in any of the possible 8 directions. Just like from homework 6, a valid kings tour is a series of legal king moves such that each square is visited exactly once. We can represent a King's tour as a 2D List where each number represents the order in which the square was visited, from 1 to N^2. For example,

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

In this case, the first list is a valid kings tour, since every consecutive number is adjacent to each other, and each number from 1 to 3^2 appears exactly once. However, the second list isn't a kings tour, since 7 is not adjacent to 8. Furthermore, we'll say that a board represents a partial King's tour, if it represents a kings tour of the numbers from 1 up to some number less than N^2. Unfilled squares will be represented by 0.

With this in mind, write the function completeKingsTour(board), which takes in a board representing a partial King's tour, and returns a board containing the complete King's tour, or None if no such board exists. If multiple King's tours exist, you may return any of them. You can only fill in positions that have 0's in the initial board - you may not fill in positions that have numbers to begin with. Two example test cases are shown below:

```
board1 =  [ [ 3, 2, 1 ],
            [ 0, 4, 0 ],
            [ 0, 0, 0 ] ]

# completeKingsTour(board) returns any valid solution,
# could return the left board in the above example.

board2 = [  [ 0, 0, 1 ],
            [ 0, 2, 5 ],
            [ 3, 4, 0 ] ]

assert(completeKingsTour(board2) == None)
```

board1 is a partial King's tour up until 4, since 1 through 4 appear adjacent to each other, and the rest of the numbers are 0. board2 is also a partial Kings tour, but no solutions exist, so completeKingsTour will return None.

You must use recursive backtracking in your solution.

Key assumptions/hints:

- You may assume that a isPartialKingsTour(board, number) function is written for you, which takes in a board and a number, and returns whether or not the board represents a partial King's tour up until that number.
- You may assume the passed in board is a valid partial King's tour.

- Your solution should work for any sized square board.

Additional Space 1 for Answer to Question 6

Additional Space 2 for Answer to Question 6

7. (22 points) **Free Response: optionally OOPy Animation** Using our animation framework and assuming that run() is already written, write init(data), mousePressed(event, data), keyPressed(event, data), and redrawAll(data) for an animation with the following specification:

   - There are two types of dots: Speeding Dots and Indecisive Dots
   - Speeding Dots:

     Black dots with radius 20 with a random initial speed

     Appear in a random position on the screen every five seconds

     Move horizontally left and right constantly

     Wraparound at the edges
   - Indecisive Dots:

     Red dots with radius 10 with speed 10

     Appear in a random position every ten seconds

     Move vertically up and down constantly

     Rebound from edges (i.e. when an indecisive dot hits an edge it bounces back in the direction it came)
   - When the user presses s, the speeding dots increase their speed by 5 up to a max speed of 50; if a speeding dot is already moving at speed 50, it should remain at that speed
   - When the user presses c, the indecisive dots switch direction (i.e. if a dot was moving up, it will now move down)

   Make reasonable assumptions for anything not specified here. Do not hardcode values for the width or height of the canvas. We recommend that, to save time writing, you abbreviate canvas, event, and data: use c, e and d, respectively. You should use short variable names.

   You may use either regular animation or OOPy animation for this problem.

Additional Space 1 for Answer to Question 7

Additional Space 2 for Answer to Question 7