# 15-112 Midterm #2a – Fall 2015
**80 minutes**

Name: ________________________________________

Andrew ID: ________________________@andrew.cmu.edu

Section: _______

- **You may not use any books, notes, or electronic devices during this exam.**
- **You may not ask questions about the exam except for language clarifications.**
- **Show your work on the exam (not scratch paper) to receive credit.**
- **If you use scratch paper, you must submit it with your andrew id on it, and we will ignore it.**
- **All code samples run without crashing. Assume any imports are already included as required.**

| DO NOT WRITE IN THIS AREA | | |
|---|---|---|
| Part 1 (CT) | 5 points | |
| Part 2 (RC) | 5 points | |
| Part 3 (SA) | 5 points | |
| Part 4 (DB) | 10 points | |
| Part 5 (FR) | 15 points | |
| Part 6 (FR) | 15 points | |
| Part 7 (FR) | 15 points | |
| Part 8 (FR) | 15 points | |
| Part 9 (FR) | 15 points | |
| Part 10/bonus | 5 points bonus | |
| Total | 100 points | |

1. **[5 pts] Code Tracing**
   Indicate what this will print:

**Statement(s):** | **Prints:**

```
def ct1(L):
  if (len(L) < 2):
    return [ ]
  else:
    f = lambda *args: len(args) + sum(args)
    mid = len(L)//2
    return ct1(L[:mid]) + [f(*L)] + ct1(L[mid:])

print(ct1([1,2,0,4,3])) # hint: prints a list
                        # with 4 values
```

______________________________

2. **[5 pts] Reasoning about code**
   For this function, find values of the parameters so that the function will return True. Place your answers in the box on the right of the function.

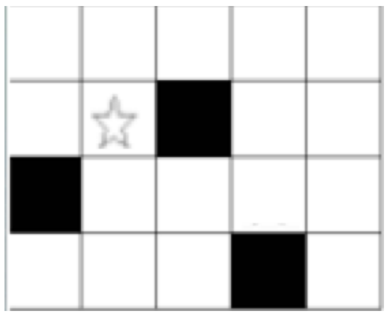
```
def rc1(L):
    assert(L == list(reversed(sorted(L))))
    assert(L != list(reversed(sorted(set(L)))))
    assert(len(L) % 2 == 0)
    assert(max(L) < 10)
    def f(L):
        if (L == [ ]): return [ ]
        else: return [L[0] * L[-1]] + f(L[1:-1])
    return (f(L) == [10,20])
```

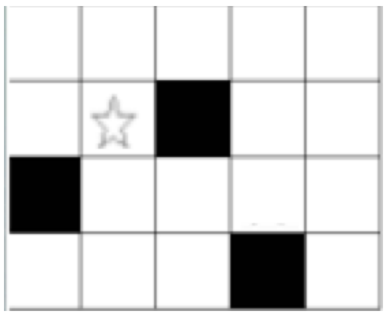
L = ___________________________

3. **[5 pts] Short Answer**
Answer each of the following very briefly.

A. Very briefly, state and prove the worst-case big-oh runtime of radixsort over a list of N non-negative integers all less than 2**32.

B. Say we start with a 4x5 board and fill the black squares as shown below (in the code from our notes, the black squares would be green, and the white squares would be cyan). Then we right-click where the star is to start a floodFill from there. On the left, write the numeric labels for the depths that result in each cell after the floodFill completes. On the right, write the numeric labels for the ordinals.


depths


ordinals

Hint #1: the numbers on the left are depths, so some labels may occur more than once.
Hint #2: the numbers on the right are ordinals, so labels must occur only once each.
Hint #3: the first label is 0, not 1 (so a 0 should be where the star is).
Hint #4: Our floodFill code recursively tries to go up, then down, then left, then right.

4. **[10 pts] Debugging**

You are provided with examples from the course notes. Each has one bug added or one line deleted, however. Find and circle the bug or missing line in the code, being sure to circle just the code that has the bug and no additional code. Then, fill in the box below each function with code to replace the code you circled so these functions work correctly.

```
def powerset(a):
    if (len(a) == 0):
        return [[]]
    else:
        allSubsets = [ ]
        for subset in powerset(a[1:]):
            allSubsets += [[a[0]] + subset]
        return allSubsets
```

```
def quicksort(L):
    if (len(L) < 2):
        return 0
    else:
        first = L[0]  # pivot
        rest = L[1:]
        lo = [x for x in rest if x < first]
        hi = [x for x in rest if x >= first]
        return quicksort(lo) + [first] + quicksort(hi)
```

5. **[15 pts] Free Response: containsCopies(path)**

   Write the recursive function containsCopies(path) that takes a path to a folder, and returns True if any two text files in that folder or any subfolder of it are exact copies of each other, and False otherwise. Note that text files with different names and in different folders may be exact copies.

   While you may use iteration, you must use recursion in a meaningful way (so, for example, you may not use os.walk). Also, for full credit, you must solve this efficiently. In particular, you may not use flatten to create an unnecessary list, and you must use an efficient means of checking for copies.

   You may assume that readFile(path) exists and returns the string contents of a text file.

6. **[15 pts] Free Response: Person and Student classes**

Implement the classes Person and Student so the following test code works properly. For full credit, you will have to use OOP properly, including, for example, using super() where appropriate, and only overriding those methods that need to be overridden.

```
p1 = Person("fred", "123-45-6789")
assert(p1.name == "fred")
assert(p1.ssn == "123-45-6789")
assert(str(p1) == "<fred's ssn is 123-45-6789>")

# Note: two Person instances are equal if and only if their ssn’s
# are equal, regardless of whether or not their names are the same
# (so someone could have different names yet be the same person).

p2 = Person("fred", "111-11-1111")
assert(not (p1 == p2)) # same name, but different ssn's
p3 = Person("barney", "123-45-6789")
assert(p1 == p3) # even with different name, same ssn's
assert(not (p1 == 42)) # don't crash here!

# Sets have to work as noted above, so persons with the
# same ssn but different names are in fact the same person

s = set()
assert(Person("wilma", "222-22-2222") not in s)
s.add(Person("wilma", "222-22-2222"))
assert(Person("wilma", "222-22-2222") in s)
assert(Person("betty", "222-22-2222") in s)
assert(Person("wilma", "333-33-3333") not in s)

# A Student is a kind of Person

p4 = Student("betty", "444-44-4444", "CMU")
assert(isinstance(p4, Person))
assert(str(p4) == "<betty's ssn is 444-44-4444>")
assert(p4.getSchool() == "CMU")

# Be sure getSchool works like this:

ok = False
try: print(Person("pebbles", "555-55-5555").getSchool())
except: ok = True
assert(ok)
```

[this page is blank (for Person and Student classes)

7. **[15 pts] Free Response: dotSplot**

Assuming the run() function from this semester's events-example0.py is already written, write the remaining functions required so that when run(400,400) is called, the following animation runs:

- The animation starts with a 10x10 ball bouncing back-and-forth horizontally across the top of a 400x400 canvas.
- A score of 0 is displayed in the middle of the screen.
- Each time the user presses the mouse in the moving ball, the score increases by 1 and the ball moves faster.
- If the user misses the ball, the screen displays "Game Over" in the middle, below the final score, with nothing else drawn, and stays that way, without any other movement or responding to any other events, except when the user presses "r", at which point the game resets to a score of 0 and play starts over.

[this page is blank (for dotSplot))

8. **[15 pts] Free Response: solveWordLadder(wordList, word1, word2)**

For this problem, assume you are given a wordList, which is a list of same-length lowercase-only words. Here is an example wordList:

```
wordList = ["dog", "doc", "dot", "box", "cot", "cat", "dig"]
```

From words in this wordList, we can build a Word Ladder, which is a list of at least 2 words where each word in the ladder occurs only once, and each word differs from the previous word at only one index. For example, ["dog", "dot", "cot"] is a word ladder, since "dog" and "dot" only differ at index 2, and "dot" and "cot" only differ at index 0. Similarly, ["dog", "dot", "dog"] is not a word ladder, since "dog" appears twice.

With this in mind, write the function solveWordLadder(wordList, word1, word2) that takes a wordList and two words, and returns a word ladder that starts with word1 and ends with word2, or None if this is not possible. If more than one word ladder exists, return the shortest possible word ladder. If more than one shortest-possible word ladder exists, you may return any one of these.

You may use iteration, but you must use recursion in a meaningful way. In particular, to solve this, you must use recursive backtracking.

[this page is blank (for solveWordLadder))

9. **[15 pts] Free Response: @withCallCounter decorator**
Write the withCallCounter decorator, and all other Python code that is required in order for the following to work correctly.

```
def testWithCallCounter():
    print("Testing @withCallCounter...", end="")
    @withCallCounter
    def f(x, y): return x*y
    assert(getCallCount(f) == 0)
    assert(f(2,3) == 6)
    assert(getCallCount(f) == 1)
    assert(f(3,2) == 6)
    assert(getCallCount(f) == 2)
    @withCallCounter
    def h(a, b, c, d, e): return a+b+c+d+e
    assert(getCallCount(h) == 0)
    assert(h(1,2,3,4,5) == 1+2+3+4+5)
    assert(getCallCount(h) == 1)
    print("Passed!")
testWithCallCounter()
```

Note that you may use one well-chosen global variable here if it helps.

10. **Bonus/Optional:**

Bonus/Optional: **[2.5 pts] What will this print?**

```
def bonusCT1():
    def f(x): return "f" + str(x if (x < 5) else g(x//2) + g(x//3))
    def g(x): return "g" + str(x if (x < 5) else f(x-1) + f(x-2))
    return f(10)
print(bonusCT1())
```

Bonus/Optional: **[2.5 pts] What will this print?**

```
def bonusCT2():
    f = lambda i: (lambda x: x*10**i * (1-2*(i%2)))
    return(f(2)(sum([f(i)(i) for i in range(5)])))
print(bonusCT2())
```