15-112 Midterm #1a – 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)	10 points	
Part 2 (RC)	10 points	
Part 3 (Big-Oh)	10 points	
Part 4 (SA)	10 points	
Part 5 (FR)	20 points	
Part 6 (FR)	20 points	
Part 7 (FR)	20 points	
Part 8/bonus	5 points bonus	
Total	100 points	

1. **[10 pts] Code Tracing**

Indicate what each will print:

Statement(s):	**Prints:**

`def ct1(L):`
`  s = ""`
`  M = L[-2:2:-2]`
`  for v in M:`
`  s += str(v)`
`  print(s, end="")`
`  M[0] += len(M)`
`  print("\t%d:%s" % (L[-2],str(M[:2])))`
 	
`ct1(list(range(7)))`
	_____

`def ct2(a):`
`  b = a[0:len(a)]`
`  c = copy.copy(b)`
`  d = copy.deepcopy(c)`
`  a[0][0] = 3`
`  c[0] = [4]`
`  c[1][0] = 5`
`  d[1][0] = 6`
`  return [a,b,c,d]`
 	
`for m in ct2([[1],[2]]):`
`  print("%d%d" %`
`  (m[0][0], m[1][0]), end="")`
	_____

2. **[10 pts] Reasoning about code**
For each function, find values of the parameters so that the function will return True. Place your answers in the box on the right of each function.

```
def rc1(n):
    r = s = 0
    if (n > 3600): return False
    while (n > 0):
        (d, n) = (n%10, n//10)
        if (d % 2 == 0):
            r = 10*r + d
        else:
            s = 10*s + d
    return (r,s) == (68, 53)
```

n = _____

```
def rc2(L):
    (rows, cols) = (len(L), len(L[0]))
    d = { 0 : 0 }
    for row in range(rows):
        for col in range(cols):
            key = L[row][col]
            if (key == 0):
                d[0] += 1
            else:
                d[key] = 10*row+col
    return (d == { 0:5, 42:12, 13:3 })
```

L = _____

3. **[10 pts] Big-Oh**
   State the Big-Oh for each of the following functions:

Function:	**Big-Oh:**

`def bigOh1(L):`
`  # assume L is a 1d list`
`  s = N = len(L)`
`  while (s > 0):`
`  for i in range(255):`
`  print(max(L[0:len(L)]))`
`  s //= 4`
	_____

`def bigOh2(L):`
`  # assume L is an NxN (square) 2d list`
`  N = len(L)`
`  s = 0`
`  for row in L:`
`  a = sorted(row)`
`  s += a[N//2]`
`  return s`
	_____

`def bigOh3(L):`
`  # assume L is a 1d list`
`  N = len(L)`
`  for i in range(N):`
`  for j in range(i, N):`
`  if (L[i] > L[j]):`
`  (L[i], L[j]) = (L[j], L[i])`
	_____

`def bigOh4(n):`
`  # assume n is an integer`
`  j = 5`
`  while (j**2 < n):`
`  j += 3`
	_____

4. **[10 pts] Short Answer**
   Answer each of the following very briefly.

   A. In our proof that mergesort is O(nlogn), where does the logn come from?

   B. Why is the builtin sort so much faster than any sorts we might write?

   C. Give two Python expressions that show how different value types affect the semantics of an operator using those values.

   D. Why in our solution to the locker problem did we make the list of lockers of size (lockers+1) and not (lockers)?

   E. Give an example of a list comprehension that evaluates to a list of all the perfect squares in increasing order from 121 (11**2) to 12321 (111**2), inclusive.

5. **[20 pts] Free Response: fasterF(L)**

Background: for this problem, we will consider the following function:

```
def f(L):
    N = len(L)
    a = copy.copy(L)
    s = N
    while (a != [ ]):
        m = max(a)
        if ((m != s) or (a.count(m) != 1)):
            return False
        a.remove(m)
        s -= 1
    return True
```

A. Write in just a few words what this function does in general (that is, what must be True, in general, for a list L in order for f(L) to return True?). Be very brief.

B. Write what the big-oh of this function is.

C. Write the Python function fasterF(L) that works identically to f(L) but runs in a faster function family. For full credit, your solution must run in O(N), where L has N elements in it, though partial credit will be given for slower solutions. You may use the next page as needed.

[this page is blank (for fasterF)]

6. **[20 pts] Free Response: isCountish(n) and nthCountish(n)**

Background: We will say that a number is "countish" (a coined term) if it contains zero 0's (not counting leading 0's), one 1, two 2's, etc, up until the largest digit in the number. So, for example, 312233 is countish, as is -312233, but 3122332 is not (it has an extra 2).

With this in mind, write the function isCountish(n) that takes a possibly-negative integer n and returns True if it is countish and False otherwise. Also write the function nthCountish(n) that takes a non-negative int and returns the nth countish number (starting from n=0).

Important: For full credit, your isCountish(n) function must run in O(D) time, where D is the number of digits in n, though partial credit will be given for slower solutions.

[this page is blank (for isCountish and nthCountish))

7. **[20 pts] Free Response: closestWordSearches(wordSearchListMap, wordSearch)**

Note: while this problem uses wordSearch boards, no actual word searching occurs in solving this problem.

Background: in our worked example, we defined a wordSearch board to be a 2d list of letters. Here, we will consider many such wordSearch boards, and we will assume they are all the same size, NxN for the same value of N. Given two such wordSearch boards b1 and b2, we can define the "distance" between them to be the number of (row,col) locations where b1[row][col] differs from b2[row][col].

For this problem, you will also be provided with a wordSearchListMap. Each key in the map is someone's name, in lowercase, such as "fred" or "wilma". Each value in the map is a 1d list of wordSearch boards (which themselves are 2d lists) that the given person has written.

With this in mind, write the function closestWordSearches(wordSearchListMap, wordSearch) that takes a wordSearchListMap, as just described, and also a single wordSearch board, and finds the wordSearch board in the map that is the closest to the given wordSearch board, according to the distance metric defined above. Your function should return the name of the person who wrote that wordSearch board. However, there could be a tie, in which case your function should return the set of all the names of all the people who wrote the closest wordSearch boards.

[this page is blank (for closestWordSearches)]

[the top of this page is blank (for closestWordSearches)]

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

```
def bonusCT1(n):
    L = [ ]
    for x in range(-n,n+1):
        for y in range(-n,n+1):
            if ((x*y != 0) and (x%y == y%x)):
                L.append(x+y)
    return (len(L), sum(L))
print(bonusCT1(100))
```

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

```
def bonusCT2():
    def g(n):
        f = (lambda q: q+1//q) if (n%2 == 0) else (lambda q: 2*q+1//q)
        def h(z):
            try: return f(z-1)
            except: return f(100)
        return sum([h(z) for z in range(4)])
    return sorted([g(n) for n in range(100)])[-3:]
print(bonusCT2())
```