fullName:______________________ andrewID:________________ section:___

15-112 S25
Quiz2 version A

Read these instructions carefully before starting:

1. Quiz versions are color-coded. You must have a different version (color) of this quiz than the students sitting to your left and right.
2. Stop writing and submit the entire quiz when instructed by the proctor.
    - Do not unstaple any pages.
    - You must submit the entire quiz with all pages intact.
3. Do not discuss the quiz with anyone else until after 5pm.
    - This applies to everyone, including students in either lecture.
4. Do not use your own scrap paper.
    - You should not need scrap paper, there is plenty of room for you on the quiz.
    - However, if you absolutely must use scrap paper, raise your hand and we will provide some. Then, you must write your andrew id clearly on the scrap paper, and hand in the scrap paper with your paper quiz. We will not grade anything on your scrap paper.
5. You may not ask questions during the quiz.
    - The one exception is for English-language clarifications.
    - If you are unsure how to interpret a problem, just take your best guess.
6. Do not use any concepts (including built-in functions) not covered in the notes through week 2 or beyond the loops part of unit 2.
    - Do not use strings, lists, tuples, dictionaries, sets, or recursion.
7. Do not hardcode your solutions.
    - We may test your code using additional test cases.
    - Hardcoding will receive zero points.
8. Assume almostEqual(x, y) and rounded(n) are both supplied for you.
    - You must write all other helper functions you wish to use, unless we specify otherwise.
9. Good luck!

Code Tracing (CT) [30 pts, 15 pts each]
For each CT, indicate what the code prints
Place your answer (and nothing else) in the box below the code.

Note: If floats occur in these CTs, they will have no more than one digit after the decimal point.

CT1:

```
def ct1(n):
    z = 0
    for x in range(1, n, 3):
        z = 10*z + x
    for y in range(n, n-8, -4):
        z = 10*z + y
    return z

print(ct1(9))
```

CT2:

```
def ct2(x):
    d = 1
    while x > 0:
        x -= d
        if x == 17:
            continue
        elif x == 3:
            break
        d *= 2
        print(x, end=' ')
    return 10*x + d

print(ct2(20))
```

Free Response / FR1: sortDigits [35 pts]

Reminder: you must not use strings or lists here.

Write the function `sortDigits(n)` that takes a possibly-negative int `n` and returns an integer with the same sign and same digits as n, only with the digits sorted from largest to smallest.

For example, for `sortDigits(204032)`, the result must have two 0's, two 2's, one 3, and one 4, sorted from largest to smallest. Thus:

```
     assert(sortDigits(204032) == 432200)
```

Here are some more examples:

```
    assert(sortDigits(123) == 321)
    assert(sortDigits(103) == 310)
    assert(sortDigits(133) == 331)
    assert(sortDigits(-204032) == -432200)
    assert(sortDigits(0) == 0)
```

Hints:

1) We found it useful to write the helper function `countDigit(n, targetDigit)` that returns the number of times the `targetDigit` occurs in the integer n.

2) We called our `countDigit` function on each digit in order from 9 to 0 (not the other way).

3) Importantly, we never used any kind of sorting algorithm (so we suggest you avoid using anything like bubblesort or mergesort, even if you happen to know what those are).

Begin your FR1 answer on the next page.

Begin your answer to FR1 here:

Free Response / FR2: nthPrimePower [35 pts]

Note: for this problem, you do not have to write `isPrime(n)`. You may assume it is already written for you.

We will say that a number n is a prime power (a coined term) if:

- n is not prime, and
- n has exactly one prime factor.

For example, consider `isPrimePower(8)`. The only factors of 8 are 1, 2, 4, and 8, and among those, only 2 is prime. As there is only one prime factor, 8 is a prime power. So, isPrimePower(8) returns True. By contrast, `isPrimePower(6)` returns `False` because 6 has two prime factors (2 and 3).

The first several prime powers are:
4, 8, 9, 16, 25, 27, 32, 49, 64, 81, 121, ...

With this in mind, write the function `nthPrimePower(n)` that takes a non-negative int n, and returns the nth prime power, where `nthPrimePower(0)` is 4.

Begin your FR2 answer here or on the next page.

Begin or continue your answer to FR2 here:

Bonus Code Tracing (BonusCT) [Optional, 2 pts each]
Bonus problems are not required.
For these CTs, indicate what the code prints.
Place your answer (and nothing else) in the box below the code.

BonusCT1:
```
def bonusCt1(n):
    # we have not seen strings, but this is bonus, so... :-)
    # This builds a string of Python code containing a
    # series of nested for loops:
    code = 'import math'
    code += '\n' + 'x = 0'
    for i in range(n):
        code += '\n' + '  '*i
        code += f'for _ in range({n}**{i+1}):'
    code += '\n' + '  '*n + f'x += {n}'
    # prints log-base-4 of x:
    code += '\n' + f'print(int(math.log(x, 4)))'
    exec(code) # and this runs the code we just created

bonusCt1(4)
```

BonusCT2:

```
def bonusCt2(n):
    def f(n):
        z = 0
        while n:
            n, d = n//10,n%10
            z += (d > 8)
        return int(bool(z))
    def g(n):
        # This uses recursion, but this is bonus, so...  :-)
        return f(n) + g(n-1) if n else 0
    return g(n)
print(bonusCt2(500))
```