fullName: _____ andrewID: _____ section: ____

15-112 F26 Quiz9 Version A
Time: 25 Minutes

You must write your name on this paper and hand this back in immediately after the assessment. If we do not receive it immediately, you will receive a zero on the assessment. Do not unstaple any pages. All pages must be handed in intact.

Do not use your own scrap paper. You should not need it, but if you must absolutely have scrap paper, raise your hand and we will provide some. Write your andrewID clearly on it and hand it in with your quiz. We will not grade anything on scrap paper.

You may not ask questions during the quiz, except for English-language clarification questions. If you are unsure about a problem, take your best guess.

Before and during the quiz, you may not view any other notes, prior work, websites or resources, including any form of AI. You may not use calculators, phones, laptops, or any other devices. You may not communicate with anyone else except for current 112 TAs or faculty during the assessment. All syllabus policies apply.

You may not discuss this test with anyone else, even briefly, in any form, until we have released grades. Failure to abide by these rules may result in an academic integrity violation.

Do not use material we have not yet covered. Thus, do not use recursion. Also: do not use eval() or exec().

For all problems, assume that the required imports (such as 'import math') are present.

Do not open this or look inside (even briefly) before you are ready to begin. Do not spend more than the specified time noted above on this assessment.

Short Answers (on Efficiency) [20 pts, 5 pts each]
Write your answers in each box.

For SA1 and SA2, assume:

- f(L) is a Python function that takes a list L of N integers.
- len(L1) == 1000
- len(L2) == 3000

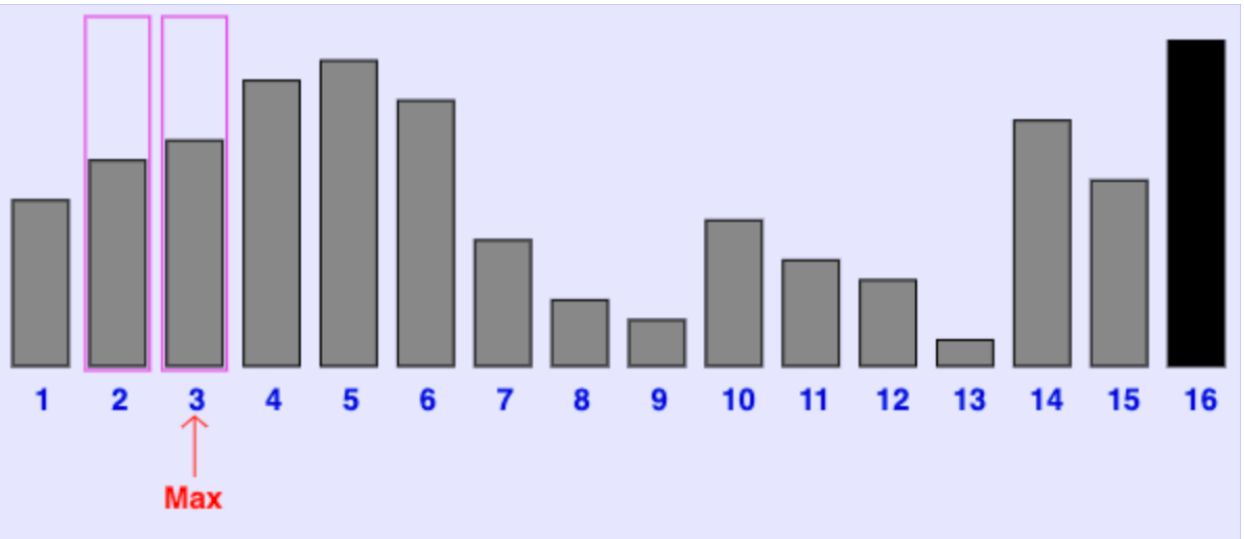
SA1: If f(L1) takes 4 seconds and f(L2) takes 36 seconds, what is the most likely function family that f runs in?

SA2. If f(L) runs in O(1), and f(L1) takes 13 seconds, how long should f(L2) take?

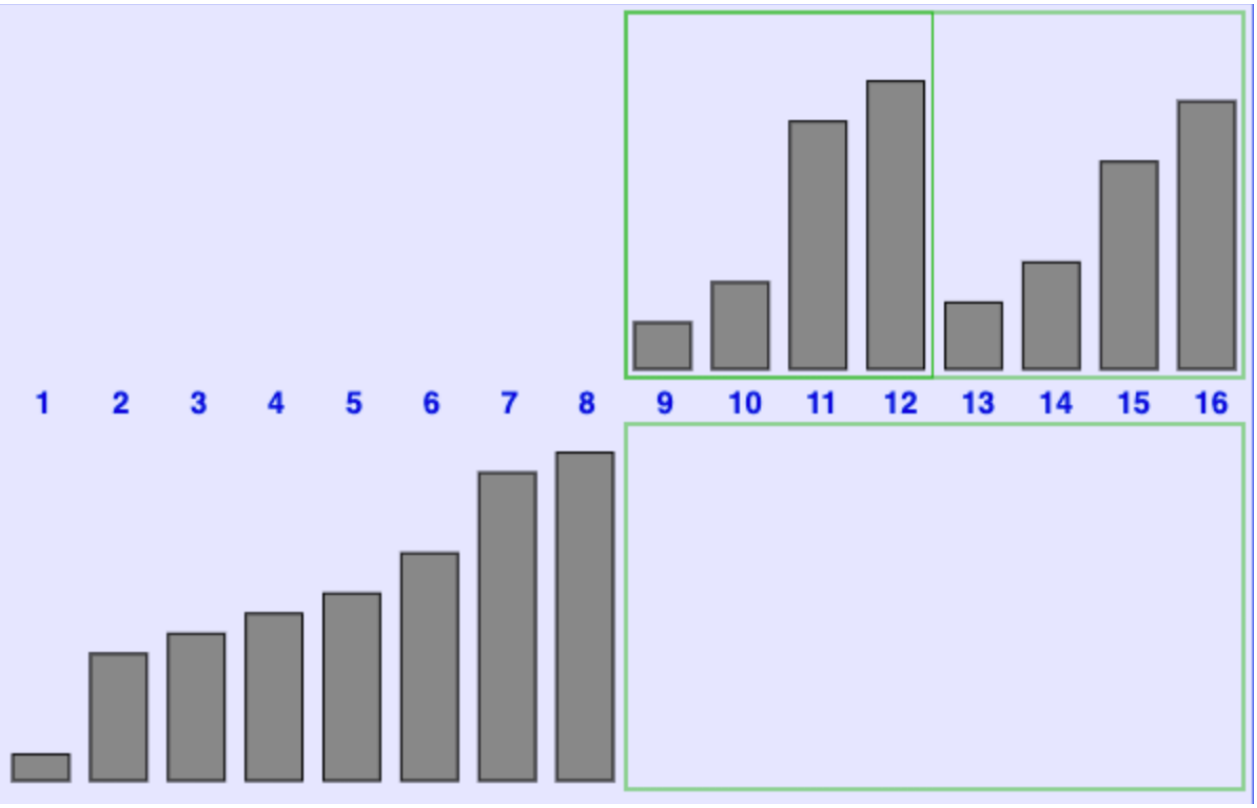
For SA3 and SA4:

- Note that the indexes in xSortLab start at 1 and not 0.

SA3. The following is a snapshot of xSortLab running selection sort. What are the indexes of the next two values to be swapped?



SA4. The following is a snapshot of xSortLab running merge sort. What are the indexes of the next two values to be compared?



True or False [10 pts, 2 pts each]

Indicate True or False for each of the following. Indicate your answer by filling in the dot.

1. OOP stands for Object-Oriented Programming.

○ True ○ False

2. Properties are also called attributes or fields.

○ True ○ False

3. Instances of our classes are immutable.

○ True ○ False

4. __hash__ is required if we plan to use instances as values (and not keys) in a dict.

○ True ○ False

5. OOP generally makes code run faster.

○ True ○ False

Code Tracing [10 pts, 5 pts each]

Indicate what the following code prints. Place your answer (and nothing else) in the box. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

 # CT1:

```
class A:
    def __init__(self, x):
        self.x = x

    def __repr__(self):
        return f'A({self.x})'

    def __add__(self, other):
        if isinstance(other, int):
            return A(10 * self.x + other) # careful!
        else:
            return A(self.x + other.x)

def ct1(x1, x2, x3):
    a1 = A(x1)
    a2 = A(x2)
    a3 = a1 + a2
    return a3 + x3

print(ct1(2, 3, 4))
```



```
# CT2:
class B:
    def __init__(self, x):
        self._x = x

    def __repr__(self):
        return f'B({self.x}, {self.y})'

    @property
    def x(self):
        return self._x

    @x.setter
    def x(self, value):
        if value % 2 == 1:
            raise ValueError('ack')
        self._x = value

    @property
    def y(self):
        return self._x * 2 # careful!

def ct2(x1, x2, x3):
    b = B(x1)
    print(b)
    try:
        b.x = x2
        print(b)
    except:
        print('oops')
    try:
        b.x = x3
        print(b)
    except:
        print('oops')
ct2(2, 5, 8)
```



FR: Breed class [60 pts]

Write the Breed class so that the following test code works. Do not hardcode to these examples. Any similar examples must also work properly.

```
def testBreedClass():
    b1 = Breed('Beagle', 20)
    assert(str(b1) == 'Beagle (about 20 pounds)')
    assert(str([b1]) == '[Beagle (about 20 pounds)]')

    b2 = Breed('Boxer', 40)
    assert(str(b2) == 'Boxer (about 40 pounds)')

    b3 = b1.crossbreed(b2)
    # The name of a crossbreed is the hyphenated combination of the
    # two breeds' names.
    # The weight is the rounded average of the two breeds' weights.
    assert(str(b3) == 'Beagle-Boxer (about 30 pounds)')

    # We will say that two breeds are equal only if both their
    # name and weight are the same, so:
    assert(b1 == Breed('Beagle', 20))
    assert(b1 != Breed('Beagle', 25))
    assert(b1 != 'do not crash here')

    # Be sure to make sets work!
    s = set()
    assert(Breed('Beagle', 20) not in s)
    s.add(Breed('Beagle', 20))
    assert(Breed('Beagle', 20) in s)

    # Remember: two breeds are equal only if both their
    # name and weight are the same, so:
    assert(Breed('Beagle', 15) not in s)

    print('Passed!')
```

Write your solution on the next page.

BonusCT [2 pts, 1 pt each]
These CTs are optional, and intended to be very challenging. They are worth very few points. Indicate what the following code prints. Place your answers (and nothing else) in the boxes below or to the right of each function. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

bonusCT1: [1pt]

```
class Q:
    def __init__(self):
        self.L = [1] * 2
    def next(self):
        self.L.append(self.L[-1] + self.L.pop(0))
        return self.L[-1]
def bonusCt1(lo, hi):
    q = Q()
    L = [ ]
    while True:
        k = q.next()
        if k > hi: break
        if k > lo: L.append(k)
    return L
print(bonusCt1(20, 40))
```

bonusCT2: [1pt]

```
class Y:
    def __init__(self, x):
        if type(x) == type:
            # hint: str(str) == "<class 'str'>"
            x = len(str(x)) - len(str(str))
        elif isinstance(x, list):
            x = [Y(y).x for y in x] + [Y(type(y)).x for y in x]
        elif not isinstance(x, int):
            x = len(str(x))
        self.x = x
def bonusCt2(L):
    return Y(L).x
print(bonusCt2([ 2.1, 'three', 4]))
```