fullName:_____ andrewID:_____ section:___

15-112 F25
Midterm3 version A
80 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 exam. We will not grade anything on scrap paper.

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

Before and during the exam, 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.

The last page is intentionally blank. You may use it to continue an FR if needed. If you do that, please write "see last page" in your answer so we know to look there.

Do not hardcode your answers. 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.

Do not open this or look inside (even briefly) before the instructors signal for you to begin. When the instructors indicate that time is up, you must *immediately* stop writing, close this document, and hand it in. Do not look at anyone else's exam.

Code Tracing [10 pts, 2 pts each]
Indicate what the following code prints. Place your answer (and nothing else) in the box below each exercise. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

CT1 [2 pts]

```
def ct1(L):
    n = 0
    S = set()
    for i in range(1, len(L)):
        T = tuple(sorted([L[i-1], L[i]]))
        if T in S:
            n += 1
        else:
            S.add(T)
    return S | {n}

print(ct1([1, 2, 3, 2, 0]))
```

CT2 [2 pts]

```
def ct2(d):
    result = dict()
    for k in d:
        delta = d[k] if isinstance(d[k], int) else 1
        i = k + delta
        result[i] = result.get(i, 0) + delta
    return result

print(ct2({1:2, 2:'ack', 3:4}))
```

CT3 [2 pts]

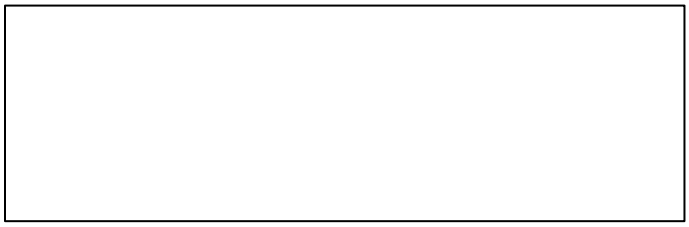
```
def ct3(L, d):
    if len(L) == 0:
        return L
    else:
        first, rest = L[0], L[1:]
        if first%2 == 0:
            return ct3(rest, d)
        else:
            x = first * 10**d
            return [x] + ct3(rest, d+1)

print(ct3([1, 2, 3, 4, 5, 6], 0))
```

CT4 [2 pts]

```
def ct4(t):
    if t.isLeaf():
        return set()
    else:
        result = set()
        for child in t.children:
            if (t.value % 2 != child.value % 2):
                result.add(child.value)
                result = result.union(ct4(child))
        return result

t = Tree(2,
         Tree(3,
              Tree(4),
              Tree(5,
                   Tree(6)
                  ),
              ),
         Tree(7),
         Tree(8,
              Tree(9)
             ),
        )

print(ct4(t))
```

CT5 [2 pts]

```
# The class Bar is used by ct5:
class Bar:
    t = 0

    def __init__(self, h):
        self.h = h
        Bar.t += h

    def f(self, d):
        self.h += d

    def __repr__(self):
        return f'B{self.h}'

def ct5(d):
    L = [Bar(d)]
    while Bar.t < 8:
        d += 1
        L.append(Bar(d))
        for v in L:
            v.f(d)
    return L

print(ct5(3))
print(Bar.t) # don't miss this!
```



True or False [10 pts, 1 pt each]
For each of the following, bubble in True or False (do not bubble in both!). Fill in the bubble completely.
True means "always True". If a statement is only sometimes True, then it is False.
Assume L and M are both lists with N integers (where N > 0).
Assume S is a set with N integers (where N > 0).

TF1. Sets can be keys in dicts.

○True ○False

TF2. Any key in any dict can be added to any set.

○True ○False

TF3. The function `hash(v)` returns a unique integer for each value `v`.

○True ○False

TF4. `L.insert(k, k)` runs in O(1) for any integer value of `k`.

○True ○False

TF5. If `L != M`, then `len(set(L)) != len(set(M))`.

○True ○False

TF6. If U and V are values in the same bucket of a hash table, and we resize the hash table, we are guaranteed that U and V will then be in different buckets.

○True ○False

TF7. Every non-empty tree must have at least one leaf.

○True ○False

TF8. Iterative merge sort is O(NlogN), but recursive merge sort is not O(NlogN).

○True ○False

TF9. When we define a class, we must define both `__eq__` and `__hash__` for sets to work properly with instances of that class.

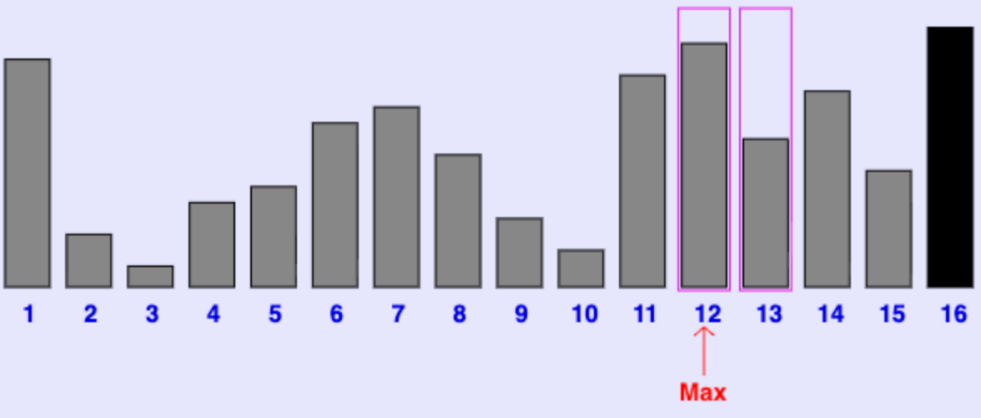
○True ○False

TF10. If `f(L)` runs in O(NlogN) and `g(L)` runs in O(N**2), where `N = len(L)`, then calling `f(L)` will run faster than calling `g(L)` for any list `L`.

○True ○False

Short Answer [5 pts, 1 pt each]
The following questions focus on searching, sorting, and hashing. Write your answer (and only your answer) in the corresponding box below.

SA1. The following image is from selection sort in xSortLab. What are the indexes of the next two values to be swapped?

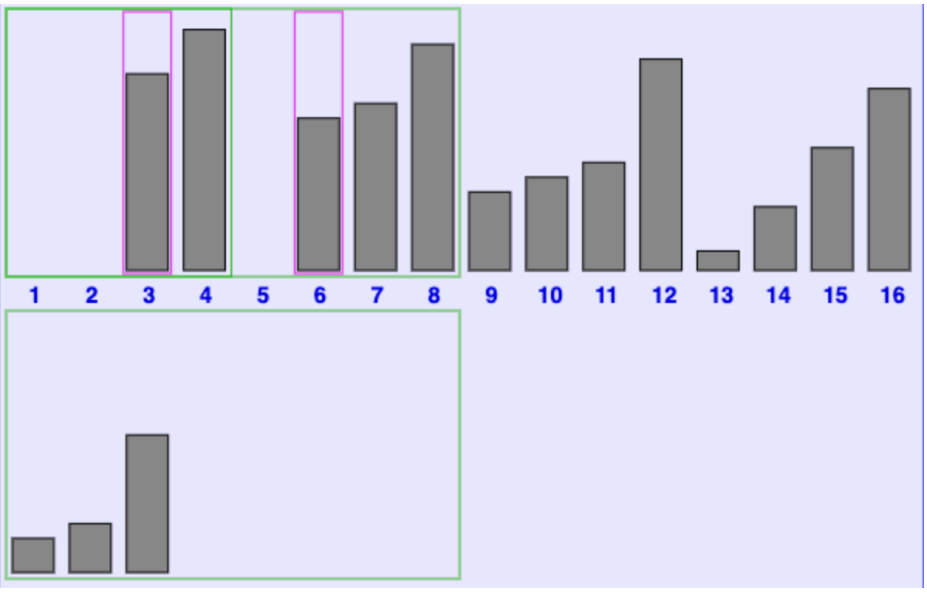


SA2. Note: for this problem, do not use Big O notation. That is, you should include constants where appropriate in your answers. With that, when running merge sort over a list of N integers, where you may assume that N is a power of 2...

A) How many steps per pass are required (where a step is either a compare or a copy)?

B) How many total passes are required?

SA3. The following image is from merge sort in xSortLab. What is the index of the next value to be copied to the temporary list?



SA4. If selection sort takes 2 seconds to sort 300 values, how long (to the nearest second) would we expect selection sort to take to sort 1200 values on the same computer?

Note: for the next question:
* You may assume that 2**10 == 1,000, and
* We will accept answers within 2 of the correct answer.

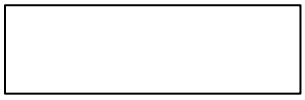
SA5. For a sorted list with 16 billion values, about how many comparisons would be required in the worst case for binary search?

Big O [10 pts, 2 pts each]
What is the worst-case Big O runtime of each of the following? All answers must be simplified (e.g. O(n) instead of O(n+5)).

Assume L is a list with N integers.
Assume S is a set with N integers.
Assume N is a positive integer.

Big O 1:

```
M = [ ]
for v in sorted(S):
    M.insert(0, v**3)
```



Big O 2:

```
i = len(L)-1
M = [ ]
while i > 0:
    M.append(L[i])
    i //= 2
```

Big O 3:

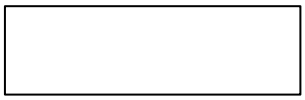
```
d = dict()
for k in range(42):
    i = k % len(L)
    d[k] = L[i]
```

Big O 4:

```
M = [ ]
i = 1
while i**2 < N:
    M.append(L.pop())
    i += 1
```

Big O 5:

```
# Assume M is a 2d NxN list of integers
target = sorted(M[0])
found = False
for row in M:
    if row == target:
      found = True
print(found)
```



FR1: maxInteriorNode [10 pts]

Background: a node in a tree is an "interior" node if it is not a leaf. With that, write the function maxInteriorNode(tree) that takes a tree and returns the largest interior node value in the tree. The function should return None if there are no interior nodes. For example:

```
t1 = Tree(1,
          Tree(2),
          Tree(3,
               Tree(4)))
assert(maxInteriorNode(t1) == 3) # since 4 is a leaf...

t2 = Tree(1,
          Tree(2),
          Tree(3))
assert(maxInteriorNode(t2) == 1)

t3 = Tree(1)
assert(maxInteriorNode(t3) == None)
```

FR2: makeIndexTuples [10 pts]
Note: for this problem, you must not use iteration (for or while loops).
Also, for full credit, your function must not mutate L.

Without using iteration, write the function makeIndexTuples(L) that takes a list L and non-mutatingly returns a new list where each value in L is replaced by a tuple of that value and its index in L.

For example:

```
L = ['cat', 'cow', 'dog']
assert(makeIndexTuples(L) ==
       [('cat', 0),('cow', 1), ('dog', 2)])
assert(L == ['cat', 'cow', 'dog']) # non-mutating
```

Hint: you may wish to use a wrapper function here.

FR3: getTopScorers [15 pts]
Background: we can represent student scores on quizzes in a dict like so (with some extra whitespace added for clarity):

```
    scores = {
        'Ann': { 'quiz1': 90, 'quiz2': 80, 'quiz3': 85 },
        'Ben': { 'quiz1': 70,              'quiz3': 85 },
        'Cam': {              'quiz2': 90              },
        'Del': {                                       },
    }
```

In this example, we see that:
* Ann scored 90 on quiz1, 80 on quiz2 and 85 on quiz3
* Ben scored 70 on quiz1, 85 on quiz3, and did not take quiz2
* Cam scored 90 on quiz2, and did not take quiz1 or quiz3
* Del did not take any quiz

With that, write the function getTopScorers(scores) that takes a dict of student scores as above and returns a dict mapping each quiz that appears in the scores to a set of the students who got the top score on that quiz.

For example, with the scores above:
* The best score on quiz1 is 90, by Ann.
* The best score on quiz2 is 90, by Cam.
* The best score on quiz3 is 85, by both Ann and Ben.

Thus, for the scores above:

```
    assert(getTopScorers(scores) ==
           { 'quiz1': {'Ann'},
             'quiz2': {'Cam'},
             'quiz3': {'Ann', 'Ben'}
           })
```

You may assume that there is at least one quiz in the scores dict.

Write your answer on the following page

FR4: Fraction Class [15 pts]
Write the Fraction class so the following test code passes.

Notes:

- You may assume all numerators and denominators are positive integers.
- You can use `gcd(x, y)` here without writing it.
- Do not hardcode any test cases. Your solution must work in general.

However, you do not have to implement any methods that are not being tested here.

```
f1 = Fraction(10, 6) # 10/6 == 5/3
assert(f1.num == 5)
assert(f1.den == 3)
assert(str(f1) == '5/3')
assert(str([f1]) == '[5/3]')

f2 = f1.getInverse() # inverse of 5/3 is 3/5
assert(str(f2) == '3/5')
assert(str([f2]) == '[3/5]')
assert(f2.num == 3)
assert(f2.den == 5)

assert(f1.nearestInt() == 2) # the nearest int to 5/3 is 2
assert(f2.nearestInt() == 1) # the nearest int to 3/5 is 1

assert(Fraction(5, 3) == Fraction(10, 6))
assert(Fraction(5, 3) != Fraction(3, 5))
assert(Fraction(5, 3) != 'do not crash here')

s = set()
assert(Fraction(5, 3) not in s)
s.add(Fraction(5, 3))
assert(Fraction(5, 3) in s)
```

Write your answer on the following page

FR5: makeRP [15 pts]

Notes:

- For this problem you must use backtracking properly to receive credit (even if there are other ways to solve the problem).
- You must use recursion here, but you are also free to use iteration if it helps.
- You may not use `math.gcd()` here, but you may write your own `gcd()` if you wish. Also, if you write `gcd()`, it can be either iterative or recursive.
- Your solution must not mutate the original list L.

Background: `if gcd(x, y) == 1`, we say that x and y are "relatively prime". Note that this does not mean that x or y are prime, just that they have no common divisors greater than 1.

More background: given a list L of positive integers, we will say that L is "RP" (short for "relatively prime") if each value in L (after the first value) is relatively prime with the previous value.

For example:

```
L1 = [2, 5, 4, 15, 14]
```

This list is RP because all of these consecutive values in L are relatively prime:

* 2 and 5
* 5 and 4
* 4 and 15
* 15 and 14

As another example:

```
L2 = [5, 4, 15, 14, 2]
```

This list is not RP because:

* 14 and 2 share the divisor 2

With that, write the function makeRP(L) that takes a list L of positive integers and uses backtracking to return the list M that contains all the same values as L only ordered so that M is RP. Return None if no such list exists.

Note there will be multiple correct answers, since if M is correct, then list(reversed(M)) is also correct. You only have to return any one correct answer, if such a list exists.

In the example above, makeRP(L2) could return several possible results, including:

* [2, 5, 4, 15, 14]
* [14, 15, 4, 5, 2]
* and others

Again, you must use backtracking properly to receive full credit. Also, you may not assume `gcd(x, y)` is already written.

You may continue your solution on the next page.

BonusCT [4 pts, 2 pts 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. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

bonusCT1 [2 pts]

```
def bonusCt1(s):
    s = ''.join(sorted(set(s.upper()))).strip()
    c = min(s)
    e = eval(str({3}).replace(str(ord('f')-ord('c')),str()))
    for i in range(len(s)):
        d = chr(ord(s[::-1][i]) - ord(c) + ord('c'))
        d = d if d <= 'e' else 'e'
        e[d] = i
    return e
print(bonusCt1('This is it'))
```

bonusCT2 [2 pts]

```
def bonusCt2(a, b, c):
    def f(n): return n if n < 1 else f(n-1) + 2*n - 1
    def g(n, k): return n if f(n) > 10**k else g(2*n, k)
    return g(g(a, b) + g(b, a) + c, 4) + c
print(bonusCt2(3, 2, 1))
```

You may continue an FR on this page if needed.