

Name:_____ andrewID:_____ Recitation:_____

15-112 F24
Midterm2 version A (80 min)

Read these instructions carefully before starting:

1. Midterm versions are color-coded. You must have a different version (color) of this quiz than the students sitting to your left and right.
2. This exam has two handouts to submit:
 - The main exam where you write your solutions.
 - An additional handout with the FR writeups, which you also must submit!
3. Stop writing and submit the entire exam when instructed by the proctor.
 - Do not unstaple any pages.
 - You must submit the entire exam with all pages intact.
4. Do not discuss the exam with anyone else until after 4pm.
 - This applies to everyone, including students in either lecture.
5. Do not use your own scrap paper.
 - You should not need scrap paper, there is plenty of room for you on the exam.
 - 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 exam. We will not grade anything on your scrap paper.
6. You may not ask questions during the exam.
 - The one exception is for English-language clarifications.
 - If you are unsure how to interpret a problem, just take your best guess.
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) [10 pts, 2 pts each]

For each CT, indicate what the code prints.

Place your answer (and nothing else) in the box below the code.

CT1:

```
def ct1(L):
    s, t, u = set(), set(), set()
    for v in L:
        if v%2 == 0:
            s.add(v)
        elif v in t:
            u.add(v)
        else:
            t.add(v)
    return sorted(s) + sorted(t) + sorted(u)
print(ct1([1,2,2,3,3,3,5]))
```

CT2:

```
def ct2(n):
    if n < 3:
        return n**2
    else:
        return ct2(n-2) + ct2(n-4)
print(ct2(4), ct2(5))
```

CT3:

```
def ct3(L):
    if L == [ ]:
        return [42]
    else:
        i = len(L)//2
        left = L[:i]
        mid = L[i]
        right = L[i+1:]
        return [sum(left) * mid] + ct3(right)
print(ct3([2, 3, 5, 7, 9]))
```

CT4:

```
def ct4(d):
    e = { None: set() }
    for k in d:
        v = d[k]
        if type(v) in [int, str]:
            if v not in e:
                e[v] = set()
        else:
            v = None
            e[v].add(k)
    return e
print(ct4({0:2, 1:2, 3:'four', 5:[6]}))
```

CT5:

The class Foo is used by ct5:

```
class Foo:
    def __init__(self, v):
        if isinstance(v, Foo):
            v = v.v * 2
        self.v = v * 2

    def __repr__(self):
        return f'Foo({self.v})'

def ct5(L):
    return [Foo(v) for v in L]
print(ct5([1, Foo(2)]))
```



True/False [10 pts, 1 pt each]

For each of the following, bubble in True or False (do not bubble in both!).

Assume L and M are both lists with N integers.

Assume S is a set with N integers.

True False 1. Sets can contain tuples.

True False 2. A dictionary d can be a value in itself (so there can be some key k such that $d[k] == d$).

True False 3. When a value v is added to a set s, Python first calls `int(v)` to convert v to an integer.

True False 4. It is faster to search for an element in an unsorted list L with linear search than it is to first sort L and then use binary search.

True False 5. If $\text{set}(L) == \text{set}(M)$, then $L == M$.

True False 6. `L.pop(0)` is generally slower than `L.pop()`.

True False 7. If a recursive function lacks a base case, then a call to that function will generally result in stack overflow.

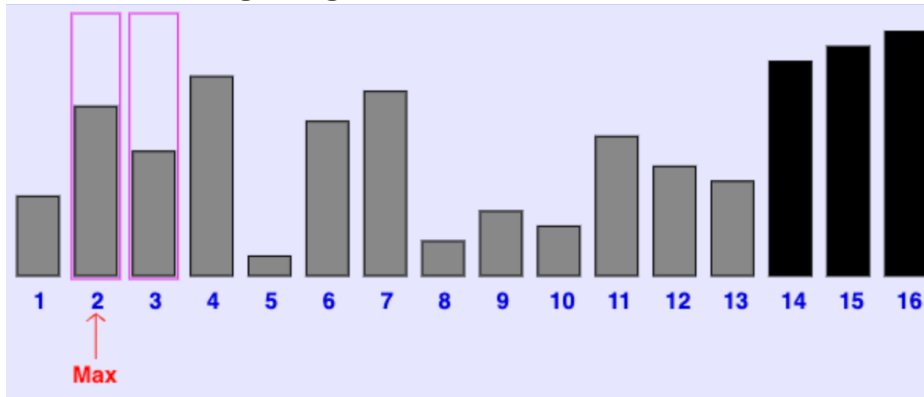
True False 8. Any iterative function can be rewritten to be recursive (without using for or while loops).

True False 9. When we define a class, we use `__repr__` instead of `__str__` to avoid crashing when we print a list of instances of that class.

True False 10. If `L` is a permutation (that is, some ordering or shuffling) of the integers from 0 to `N-1` (inclusive), then `len(set(range(N)) - set(L)) == 0`.

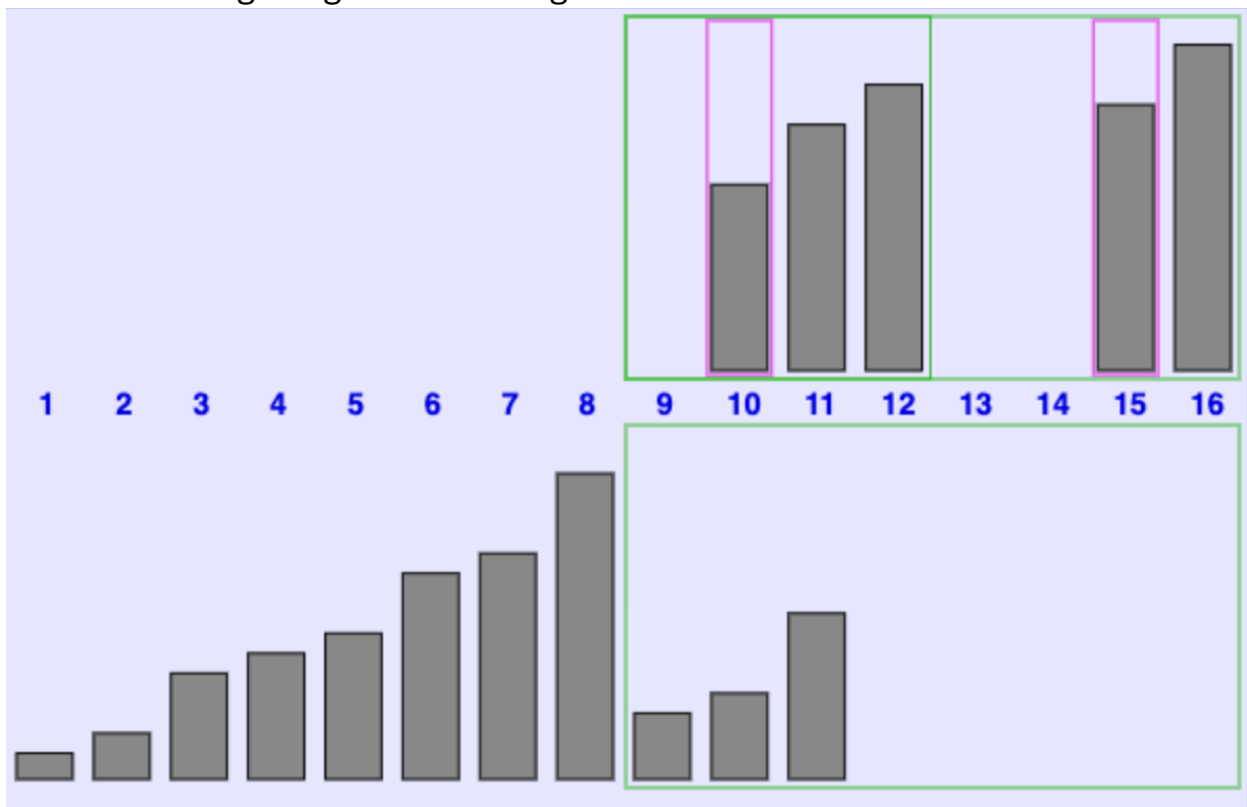
Searching, Sorting, and Hashing [5 pts, 1 pt each]

1. The following image is from selection sort in xSortLab:



What are the indexes of the next two values to be swapped?

2. The following image is from merge sort in xSortLab:



What is the index of the next value to be copied to the temporary list?

3: 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?

4. If selection sort takes 1 minute to sort 400 values, how long (to the nearest minute) would we expect selection sort to take to sort 2,000 values on the same computer?

Note: for the next question:

- you may assume that $2^{10} \approx 1,000$, and
- we will accept answers within 2 of the correct answer.

5. For a sorted list with 4 million values, about how many comparisons would be required in the worst case for binary search?

Big O [10 pts, 2.5 pts each]

For each problem, indicate the Big O (in simplified form) of the code.
Place your answer (and nothing else) in the box to the right of the code.

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:

```
s = set()
t = set()
for v in reversed(sorted(L)):
    if v in s:
        t.add(v)
    else:
        s.add(v)
return sorted(s)
```

Big O #2:

```
M = [ ]
for x in range(N):
    for y in range(x, N):
        for z in range(5):
            M.append((x, y, z))
```

Big O #3:

```
total = 0
```

```
k = 2**N
```

```
while N > 0:
```

```
    total += k
```

```
    k //= 2
```

```
    N -= 2
```



Big O #4:

```
while L.count(v) > 0:
```

```
    L.pop(0)
```

```
print(sorted(L))
```



Fractal Multiple Choice [5 pts]

Consider the following code:

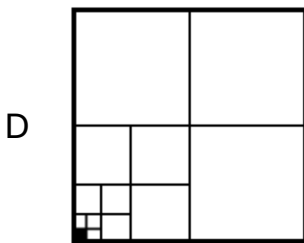
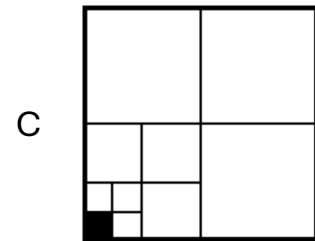
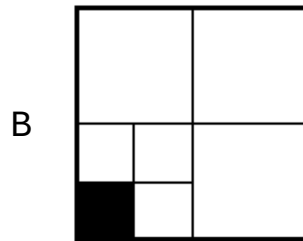
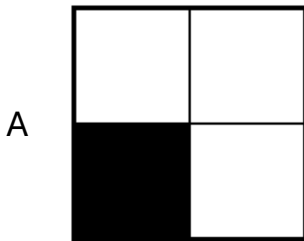
```
from cmu_graphics import *

def drawFractal(level, cx, cy, r):
    if level == 0:
        drawRect(cx-r, cy-r, 2*r, 2*r)
    else:
        drawLine(cx, cy-r, cx, cy+r)
        drawLine(cx-r, cy, cx+r, cy)
        drawFractal(level-1, cx-r/2, cy+r/2, r/2)

def redrawAll(app):
    drawRect(100, 100, 200, 200, fill=None,
            border='black', borderWidth=4)
    drawFractal(3, 200, 200, 100)

runApp()
```

Which one of the following will be drawn when the code above runs?
Assume each outer box is the 200x200 rectangle drawn in the code.
(Circle the letter corresponding to your answer.)



E None of these

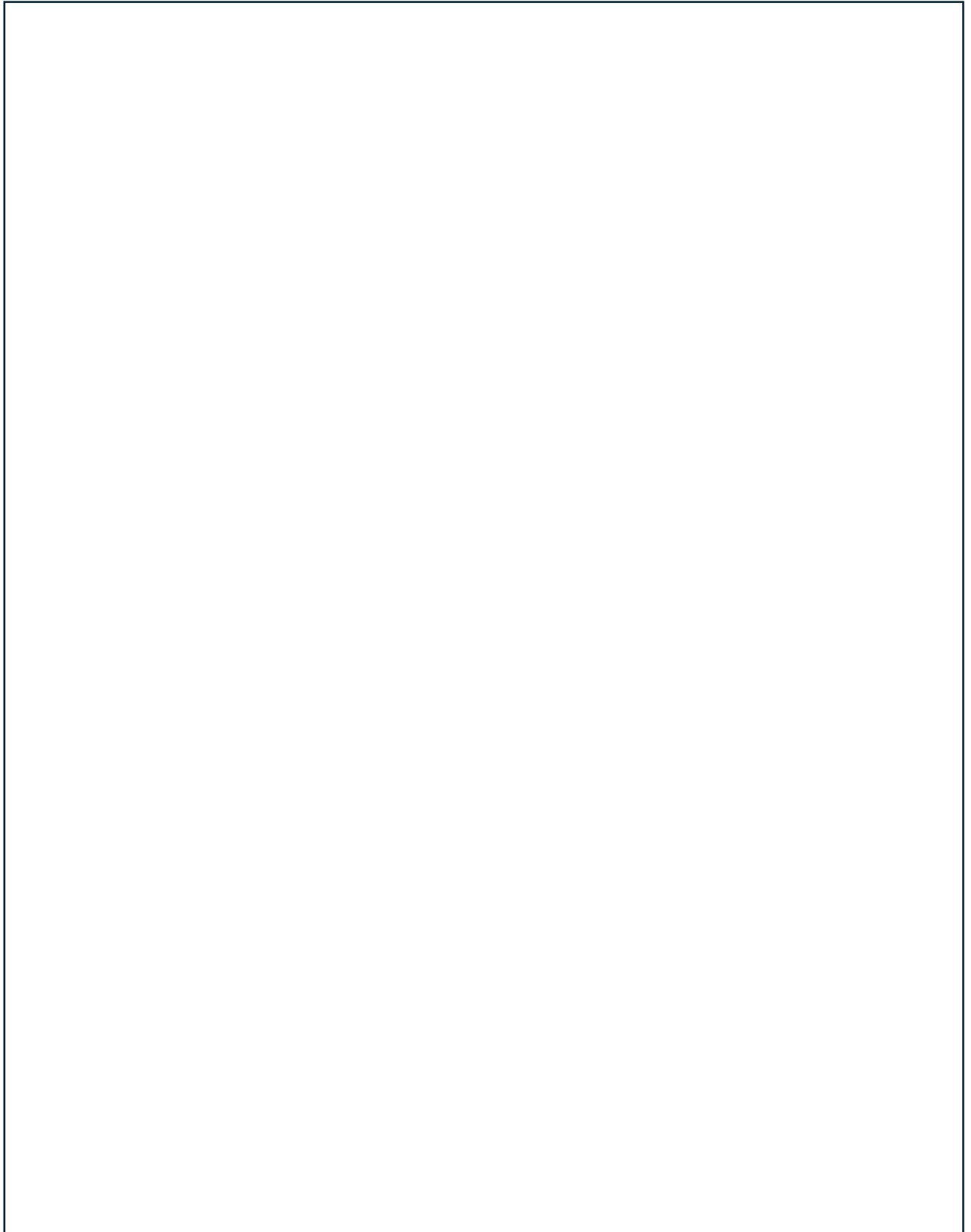
Free Response / FR1: getColorMap [15 pts]

See the FR Handout for the writeup. Write your answer to FR1 here:

Continue your answer to FR1 here:

Free Response / FR2: nthPerfect [15 pts]

See the FR Handout for the writeup. Write your answer to FR2 here:

A large, empty rectangular box with a thin black border, intended for the student to write their answer to the free response question. The box occupies most of the page below the instructions.

Continue your answer to FR2 here:

Free Response / FR3: makeFactorish [15 pts]

See the FR Handout for the writeup. Write your answer to FR3 here:

Continue your answer to FR3 here:

Free Response / FR4: File and Folder Classes [15 pts]

See the FR Handout for the writeup. Write your answer to FR4 here:

Continue your answer to FR4 here:

Bonus Code Tracing (BonusCT) [Optional, 4 pts, 2 pts each]

Bonus problems are not required.

For each CT, indicate what the code prints.

Place your answer (and nothing else) in the box below the code.

BonusCT1:

```
def bonusCt1(x):
    def f(x):
        try: return x + f(x/2)
        except: return 0
    def g(x):
        return x and f(x) + g(x//2)
    return round(g(x))
print(bonusCt1(10))
```

BonusCT2:

```
def bonusCt2(n):
    def f(): return random.randrange(n)/n
    def g(n): return [f()*2 + f()*2 for _ in range(n)]
    def h(n): return sum([v <= 1 for v in g(n)])/n
    return round(math.pi/h(n))
print(bonusCt2(10**6))
```