fullName:_____ andrewID:_____ section:___

15-112 S25
Quiz9 version B

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 9 or beyond unit 6.
    - Do not use 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!

Free Response (FR1): fasterFoo(L) [20 pts]

For this exercise, you are given a function `foo(L)` that takes a list L with N integers. First study the function and **trace through the test cases** to understand what the function is doing.

Then write the function `fasterFoo(L)` which does the same thing as `foo(L)` but runs in O(N).

Answers that do not run in O(N) will not receive any credit.

```
import copy

def foo(L):
    M = copy.copy(L)
    for v in L[::-1]:
        if v**2 not in L:
            M.insert(0, v**2)
    return M

def testFoo():
    print('Testing foo()...', end='')
    assert(foo([2, 5, 1, 4]) == [25, 16, 2, 5, 1, 4])
    assert(foo([2, 3, 5]) == [4, 9, 25, 2, 3, 5])
    print('Passed!')

testFoo()
```

Write your answer on the following page

#Write your answer to FR1 here:

Free Response (FR2): getEvenCountsMap(L) [30 pts]

Write the function `getEvenCountsMap(L)` that takes a possibly-empty list L of integers and returns a dict mapping the counts of each value in L to a set of the values in L with that count, but only if the count is even.

Note: for full credit, **your function must run in O(N)** time where `N = len(L)`.

For example, say `L = [1,5,1,3,3,4,3,3,4]`. We see that:

```
L.count(1) == 2
L.count(3) == 4
L.count(4) == 2
L.count(5) == 1
```

Here, `getEvenCountsMap(L)` returns:

```
{2: {1, 4}, 4: {3}}
```

This is because 1 and 4 have a count of 2, 3 has a count of 4.
We ignore 5 because it has an odd count (1).

Thus:

```
assert(getEvenCountsMap([1, 5, 1, 3, 3, 4, 3, 3, 4])
                        == {2: {1, 4}, 4: {3}})
```

Also:

```
assert(getEvenCountsMap([1, 6, 3, 3, 6, 6, 3])
                         == dict()) # no even counts

assert(getEvenCountsMap([]) == dict()) # empty list
```

Write your answer on the following page

#Begin your answer to FR2 here:

#Continue your answer to FR2 here:

Code Tracing (CT) [20 pts, 10 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):
    L += L
    M = [(L[i] + i) for i in range(len(L))]
    s = set(L)
    t = set(M)
    return (s-t, t|s, s.intersection(t))
print(ct1([1,2,4]))
```

CT2:

```
def ct2(d):
    e = dict()
    for k in d:
        n = len(d[k])
        if n not in e:
            e[n] = {n}
        e[n].add(k)
    return e
print(ct2({1:'ab', 2:'c', 3: 'de'}))
```

True or False (TF) [10 pts, 2 pts each]
For each of the following, bubble in True or False (do not bubble in both!). Fill in the bubble completely.
Assume L and M are both lists with N integers.
Assume S is a set with N integers.

TF1. If S is a non-empty set, then `S[0]` always crashes but `S.pop()` does not crash.

◯True ◯False

TF2. If d is a dict, then `d[d]=42` crashes but `d[42]=d` does not crash.

◯True ◯False

TF3. Binary search will work incorrectly over an unsorted list, but it will not crash.

◯True ◯False

TF4. If `set(L) == set(M)`, then `L == M`.

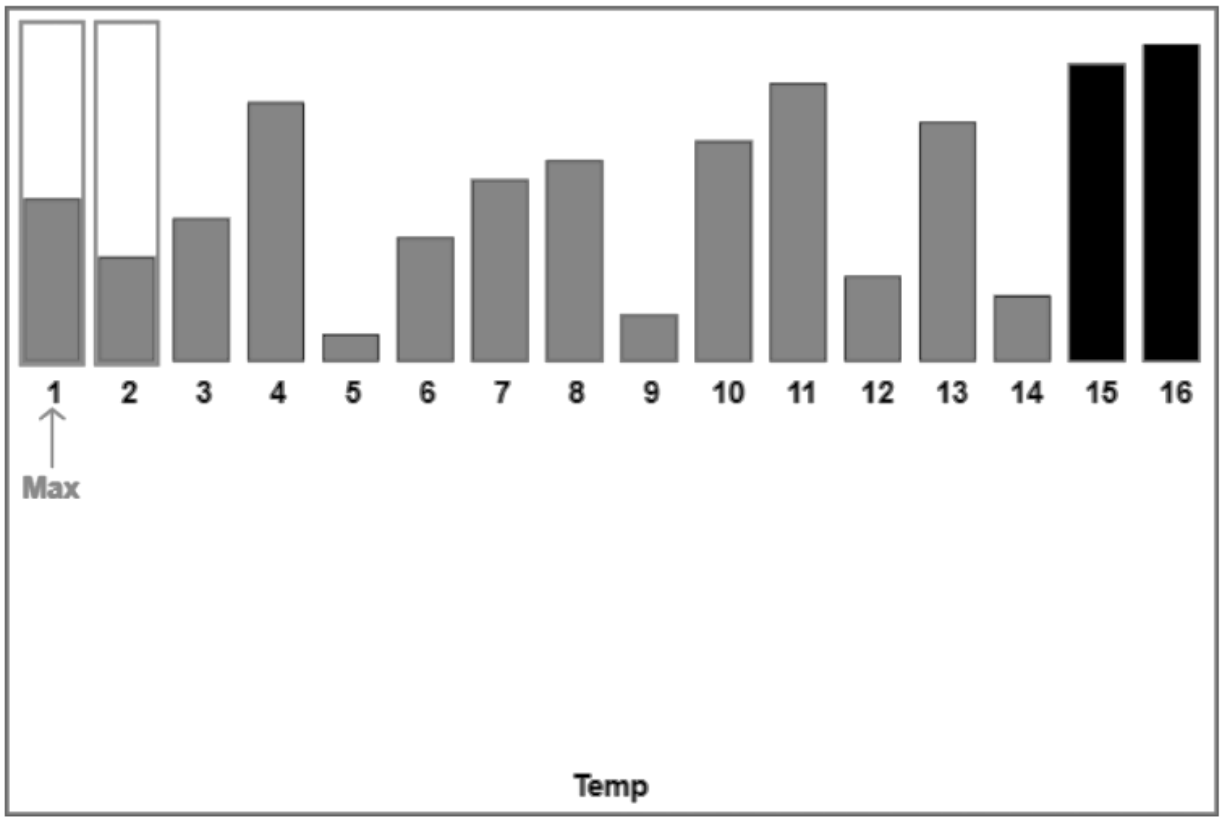
◯True ◯False

TF5. If `len(set(L) - set(M)) > 0`, then L and M must share at least one value.

◯True ◯False

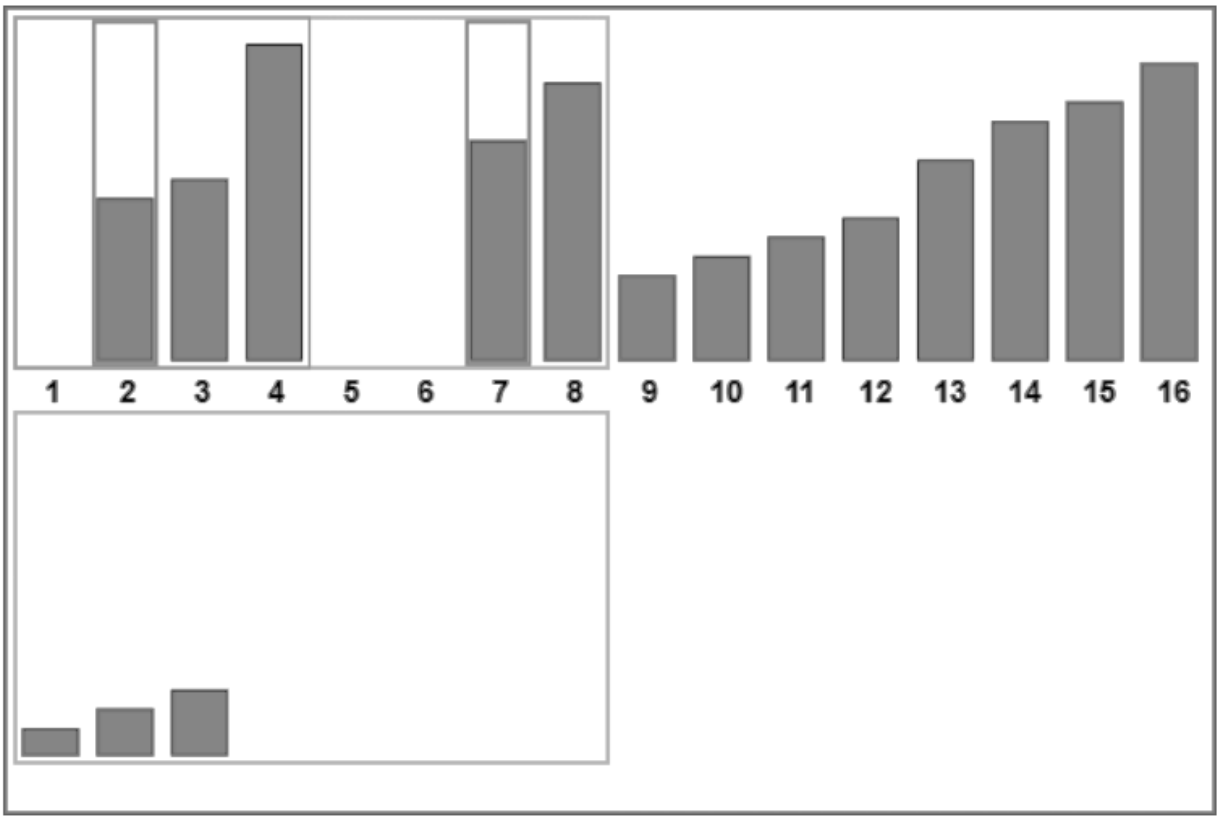
Short Answer (SA) [20 pts, 2 pts each]
The following questions focus on searching, sorting, and hashing. Write your answer (and only your answer) in the corresponding box below.

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



What are the indexes of the next two values to be **swapped**? (Use the indexes in the image even though they start at 1.)

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? (Use the indexes in the image even though they start at 1.)

Note: for the next 2 questions, give an expression in terms of N (e.g. N2 + 1)**

3. When running selection sort over a list of N integers, how many steps are taken on the first pass (where a step is either a compare or a swap)?

4. 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?

5. If selection sort takes 8 seconds to sort 3 million values, how long (to the nearest second) would we expect selection sort to take to sort 9 million values on the same computer? Your answer must be a specific number, so if 10 is the answer, write 10, not log(2**10).

6. Heap sort is an O(NlogN) sort. Say we know these facts:

- Merge sort takes 5 seconds to sort a list L.
- Heap sort takes 15 seconds to sort the same list L.
- Merge sort takes 25 seconds to sort a list M.

Given these facts, how long would we expect heap sort to take to sort the list M? Your answer must be a specific number, so if 10 is the answer, write 10, not log(2**10).

7. Say that we were using this very bad hash function:

```
def hash(v):
    return 42
```

This function hashes every value to the integer 42. While sets would still work with this hash function, they would perform very poorly. For example, adding a value to a set would be O(N) instead of O(1). In just a few words, why so?

Note: for the next 2 questions:

- You may assume that 2**10 == 1,000, and
- Your answer must be a specific number, so if 10 is the answer, write 10, not log(2**10).
- We will accept answers within 2 of the correct answer.

8. For a list with 16,000 values, how many comparisons would be required in the worst case for linear search?

9. For a sorted list with 16,000 values, about how many comparisons would be required in the worst case for binary search?

10. For this question, assume:

1) Hash tables are implemented as described in lecture,
2) Each bucket in a hash table has a max bucket size of 2,
3) Hash tables start with 4 buckets, and
4) Hash(x) == x for any integer x.

Say we have this code:

```
s = set()
s.add(88)
s.add(24)
s.add(24)
s.add(13)
```

With the assumptions above, write the values in each bucket for the hash table that represents this set after running those lines of code:

0:
1:
2:
3:

Bonus Code Tracing (BonusCT) [Optional, 2pts 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 ct1(s):
    s = set(s.split('C'))
    s = set(str(s)) - set(str(set('A set')))
    return ''.join(sorted(s))
print(ct1('ACABCDE'))
```

BonusCT2:

```
def ct2(d):
    r = len(d)
    while any(d.values()):
        for k in d:
            r += d[k]
            d[k] //= 2
    return r
print(ct2({'A':1024, 'Z':16, 16: 4}))
```