Name:_____ Section:____ AndrewID:_____

**15-112 Fall 2021 Quiz 8b**

*** Up to 20 minutes. * No calculators, no notes, no books, no computers. * Show your work!**

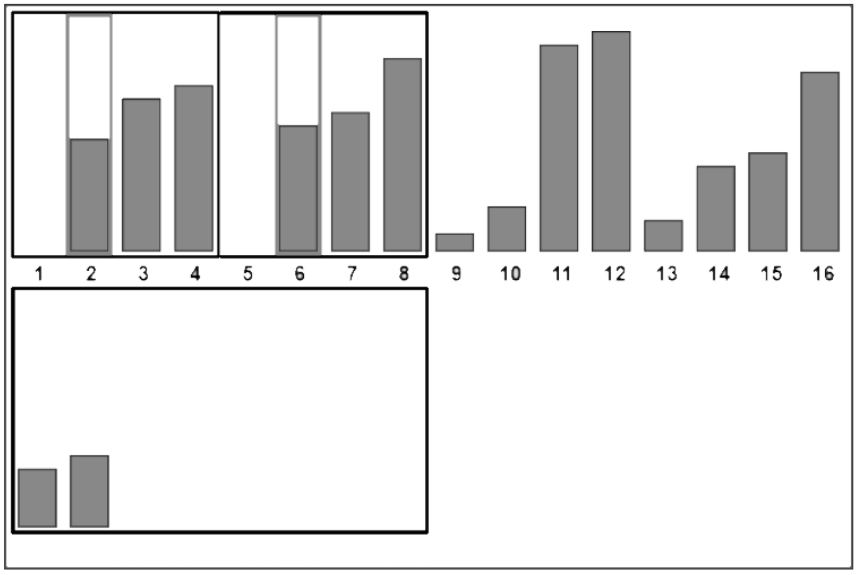
*** No recursion**

**Code Tracing [15pts]:** Indicate what the following code prints. Place your answers (and nothing else) in the box to the right of the code. **Note: If you must print an unordered value, we will accept all valid possibilities**

```
# Note: this prints 2 lines!
def ct1(S):
    x = set()
    y = set()
    z = {}
    for elem in S:
        if elem in x:
            y.add(elem)
        x.add(elem)
    print(f'x:{x}')
    print(f'y:{y}')
    for i in range(len(S)-len(y)):
        if S[i] not in z:
            z[S[i]] = i
        else:
            z[i] = z[S[i]]
    print(f'z: {z}')
S = 'bananas'
ct1(S)
```

**Short Answers [4pts ea, 20pts total]:**

**SA1:** The following is a snapshot of xSortLab running mergeSort. What is the index of the next value to be moved (i.e. copied)? (Use the indices under the bars)



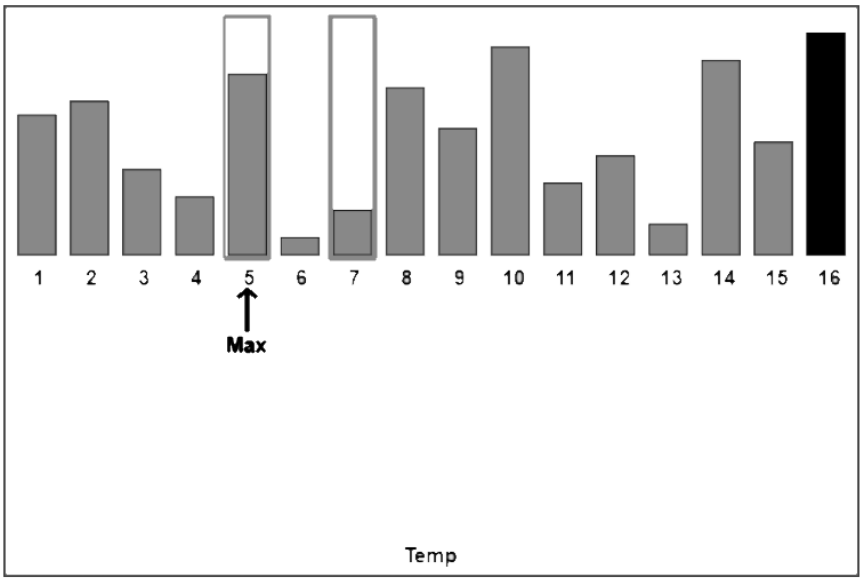

**SA2:** Arrange the following 6 Big-O efficiencies in order from fastest to slowest, given very large inputs:

`O(N+1000)` `O(3N**0.5)` `O(2**N)` `O(7000+logN)` `O(20**10)` `O(N**2)`

**<Slowest_____Fastest>**

**SA3:** If f(N) runs in O(N**2), and f(2000) takes 2 seconds, how long would we expect f(6000) take?

**SA4:** The following is a snapshot of xSortLab running selectionSort. What are the indexes of the next two values to be swapped? (Use the indices under the bars)




**SA5: Fill in the squares to indicate whether each statement is TRUE or FALSE:**

☐**TRUE** ☐**FALSE** **Mergesort must complete approximately N passes before a list is sorted**

☐**TRUE** ☐**FALSE** **On each pass, mergesort will need to perform approximately logN comparisons**

☐**TRUE** ☐**FALSE** **Mergesort is somewhat faster than selection sort, but in the same Big O function family**

☐**TRUE** ☐**FALSE** **There is no comparison sorting algorithm in a faster function family than mergesort (assuming no prior knowledge about the unsorted items and no quantum computers)**

☐**TRUE** ☐**FALSE** **Mergesort runs in O(NlogN)**

**Free Response: CreditCounter Class [65 pts]**

Write the CreditCounter class, which keeps track of some number of students, and how many CMU credits they have earned. Your solution should pass the following test cases. Be sure not to hardcode, so that any similar test code also passes.

```
def testCreditCounterClass():
    print('Testing CreditCounter class...', end='')
    # Create a CreditCounter with these initial counts
    sb1 = CreditCounter({'Alice':80, 'Bob':42})
    assert(sb1.getCredits('Alice') == 80)
    assert(sb1.getCredits('Bob') == 42)
    assert(sb1.getCredits('Chee') == None)
    assert(sb1.getMostCredits() == { 'Alice' }) # Set of names w/ the most credits

    sb1.addScore('Bob', 40) # Bob just earned 40 more credits!
    assert(sb1.getCredits('Bob') == 82) # Now he has 82 credits
    assert(sb1.getMostCredits() == { 'Bob' }) # Bob has 82, Alice has 80

    sb1.addScore('Chee', 64)   # Chee wasn't there before, but now has 64 credits
    assert(sb1.getCredits('Chee') == 64)
    sb1.addScore('Chee', 18)
    assert(sb1.getCredits('Chee') == 82)
    assert(sb1.getMostCredits() == { 'Bob', 'Chee' }) # Bob and Chee have 82

    assert(sb1.getAll() == { 'Alice':80, 'Bob':82, 'Chee':82 })

    sb2 = sb1.getCopy() # This is a copy of sb1, where changes to the copy
                        # do not affect the original, and vice versa
    assert(sb2.getAll() == { 'Alice':80, 'Bob':82, 'Chee':82 })
    sb2.addScore('Alice', 10) # Alice now has 90 in sb2, but still has 80 in sb1
    assert(sb2.getMostCredits() == { 'Alice' })
    assert(sb1.getMostCredits() == { 'Bob', 'Chee' })
    print('Passed!')

testCreditCounterClass()
```

**(You can write your answer on the next page for more space.)**

**(You may answer the FR on this page.)**

**(You may answer the FR on this page.)**

**Bonus/Optional: Code Tracing [+2pts]**

Indicate what this prints. Place your answer (and nothing else) in the box.

```
class A(object):
    def __init__(self, x): self.x = x
    def B(self, x): self.x += self.B(x); return x + self.x
    A = B
    def B(self, x): self.x += 2*x; return x + 2*self.x
class B(A):
    def A(self, x):
        return super().A(x) + super().B(x)
def bonusCt(a): return (a.A(3), a.x)
print(bonusCt(A(4)))
print(bonusCt(B(4)))
```