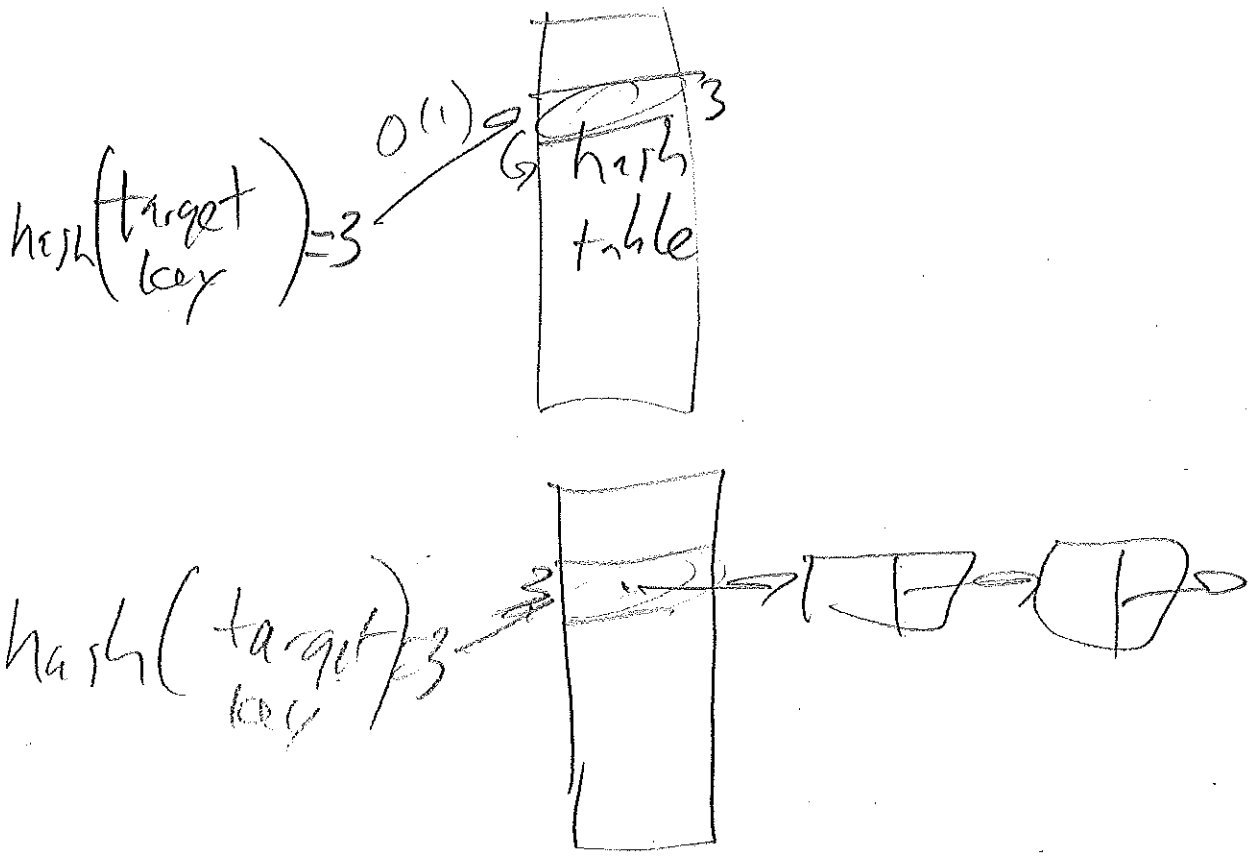
Test 2 will be Thursday March 31 in class. It will be closed-book and notes, except for one 8.5" x 11" sheet of paper containing any notes that you want. (Yes, you can use both the front and back of this piece of paper.) Plus, you can use your Python Summary handout.
The test will cover Chapters 4 and 5. The following topics (and maybe more) with be covered:

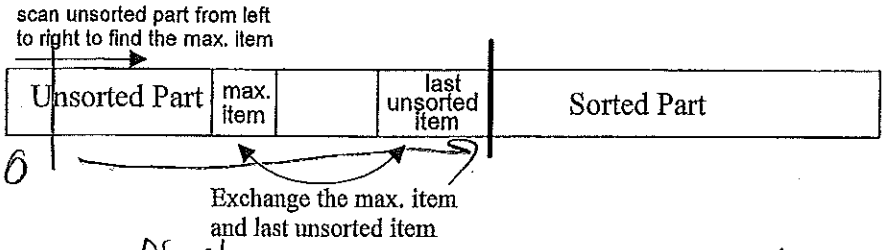
Chapter 4: Recursion
Recursive functions: base-case(s), recursive case(s), tracing recursion via run-time stack or recursion tree, "infinite recursion"
Costs and benefits of recursion
Recursive examples: countDown, OrderedList __str__ method, fibonacci, factorial, binomial coefficient
Divide-and-Conquer technique of solving a problem. Examples: fibonacci, coin-change problem
Backtracking technique of solving a problem: Examples: coin-change problem, maze (textbook)
General concept of dynamic programming solutions for recursive problems that repeatedly solve the same smaller problems over and over. Example fibonacci, coin-change problem, binomial coefficient

Chapter 5: Searching and Sorting
Sequential/Linear search: code and big-oh analysis
Binary Search: code and big-oh analysis
Python List implementation (ListDict) of dictionaries and big-oh analysis
Hashing terminology: hash function, hash table, collision, load factor, chaining/closed-address/external chaining, open-address with some rehashing strategy: linear probing, quadratic probing, primary and secondary clustering
hashing implementation of dictionaries (ChainingDict and OpenAddrHashDict) and their big-oh analysis
General idea of simple sorts
Simple sorts: selection, bubble, insertion sorts and their big-oh analysis
Advanced sorts and their big-oh analysis: heap sort, quick sort and merge sort



Fall 2015 Name: _____

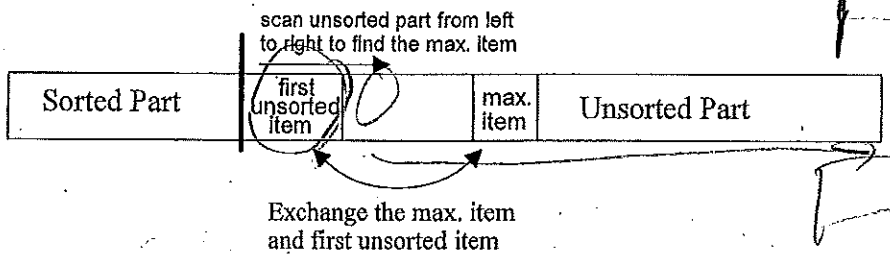
Question 4. In class we developed the following selection sort code which sorts in ascending order (smallest to largest) and builds the sorted part on the right-hand side of the list, i.e.:




```
def selectionSort(aList):
    for lastUnsortedIndex in range(len(aList)-1, 0, -1):
        maxIndex = 0
        for testIndex in range(1, lastUnsortedIndex+1):
            if aList[testIndex] > aList[maxIndex]:
                maxIndex = testIndex
        # exchange the items at maxIndex and lastUnsortedIndex
        temp = aList[lastUnsortedIndex]
        aList[lastUnsortedIndex] = aList[maxIndex]
        aList[maxIndex] = temp
```

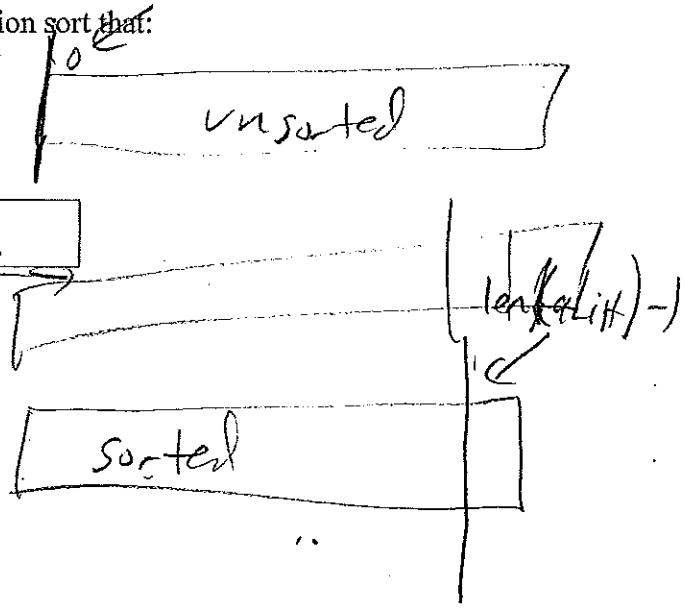
(20 points) For this question write a variation of the above selection sort that:

- sorts in **descending order** (largest to smallest)
- builds the **sorted part on the left-hand side** of the list, i.e.,




```
def selectionSortVariation(myList):
```

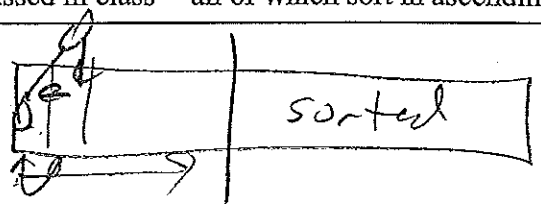
(see modified code above.)



Fall 2015 Name: _____

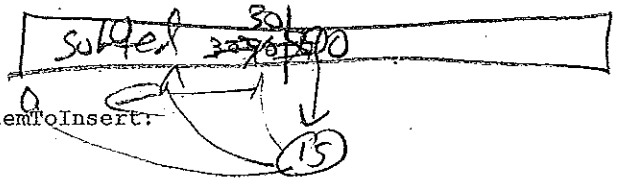
Question 3. (15 points) Consider the following simple sorts discussed in class -- all of which sort in ascending order.

```
def bubbleSort(myList):
    for lastUnsortedIndex in range(len(myList)-1,0,-1):
        alreadySorted = True
        for testIndex in range(lastUnsortedIndex):
            if myList[testIndex] > myList[testIndex+1]:
                temp = myList[testIndex]
                myList[testIndex] = myList[testIndex+1]
                myList[testIndex+1] = temp
                alreadySorted = False
        if alreadySorted:
            return
```

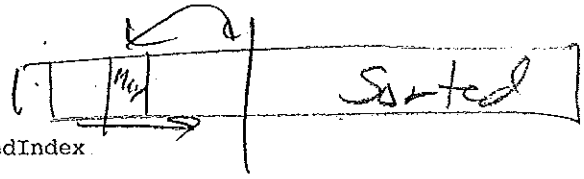



```
def insertionSort(myList):
    for firstUnsortedIndex in range(1,len(myList)):
        itemToInsert = myList[firstUnsortedIndex]
        testIndex = firstUnsortedIndex - 1
        while testIndex >= 0 and myList[testIndex] > itemToInsert:
            myList[testIndex+1] = myList[testIndex]
            testIndex = testIndex - 1

        myList[testIndex + 1] = itemToInsert
```




```
def selectionSort(aList):
    for lastUnsortedIndex in range(len(aList)-1, 0, -1):
        maxIndex = 0
        for testIndex in range(1, lastUnsortedIndex+1):
            if aList[testIndex] > aList[maxIndex]:
                maxIndex = testIndex
        # exchange the items at maxIndex and lastUnsortedIndex
        temp = aList[lastUnsortedIndex]
        aList[lastUnsortedIndex] = aList[maxIndex]
        aList[maxIndex] = temp
```



Timings of Above Sorting Algorithms on 10,000 items (seconds)			
Type of sorting algorithm	Initial Ordering of Items		
	Descending	Ascending	Random order
bubbleSort.py	24.5	0.002	16.5
insertionSort.py	14.2	0.004	7.3
selectionSort.py	7.3	7.7	6.8

a) Explain why bubbleSort on a descending list (24.5 s) takes longer than bubbleSort on a random list (16.5 s).

Descending order cause bubble sort to compare and swap all the way down the unsorted part, but with random order only about half the swaps would be performed (same # of compares).

b) Explain why bubbleSort on a descending list (24.5 s) takes longer than insertionSort on a descending list (14.2 s).

Bubble sorts compares and swaps all the way down unsorted part and insertion sort compares and moves all the way down sorted part. Since each swap takes 3 moves, bubble sort is slower than insert sorts one move to shift.

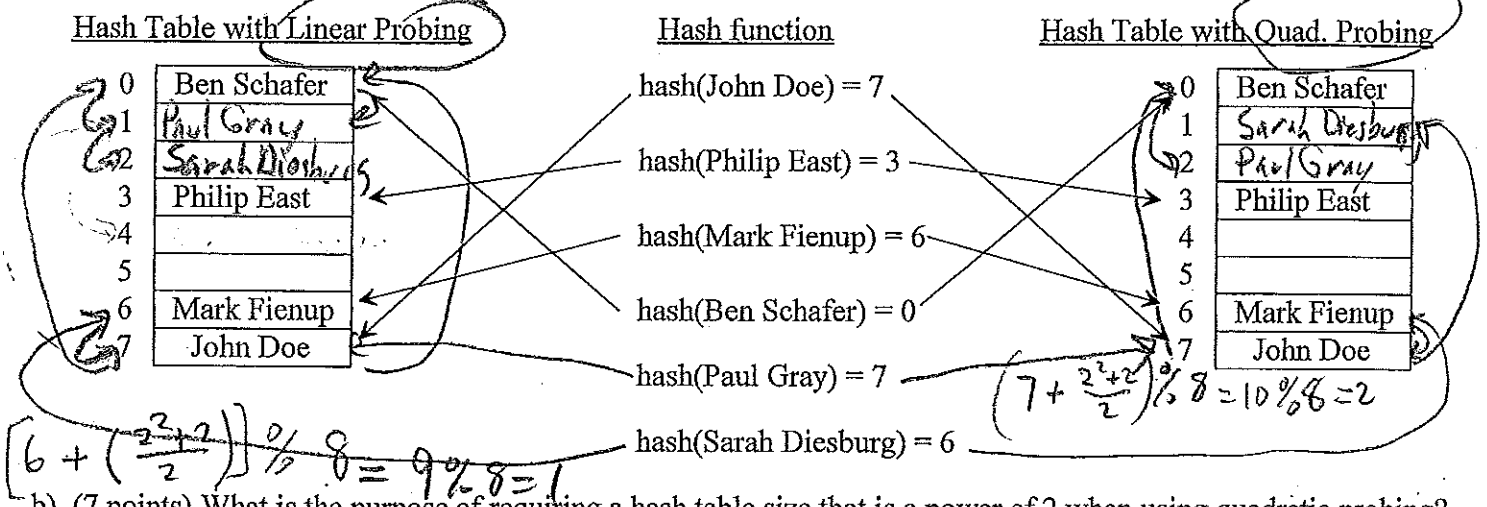
c) Explain why insertionSort on a descending list (14.2 s) takes longer than selectionSort on a descending list (7.3 s).

Insertion on descending compares and moves all the way down sorted part. Selection compares all the way down unsorted part, but only does one swap (3 moves).

Question 5. Recall the quadratic rehashing strategy we discussed for open-address hashing:

Strategy	Description
quadratic probing	Check the square of the attempt-number away for an available slot, i.e., [home address + ((rehash attempt #)^2 +(rehash attempt #))/2] % (hash table size), where the hash table size is a power of 2. Integer division is used above

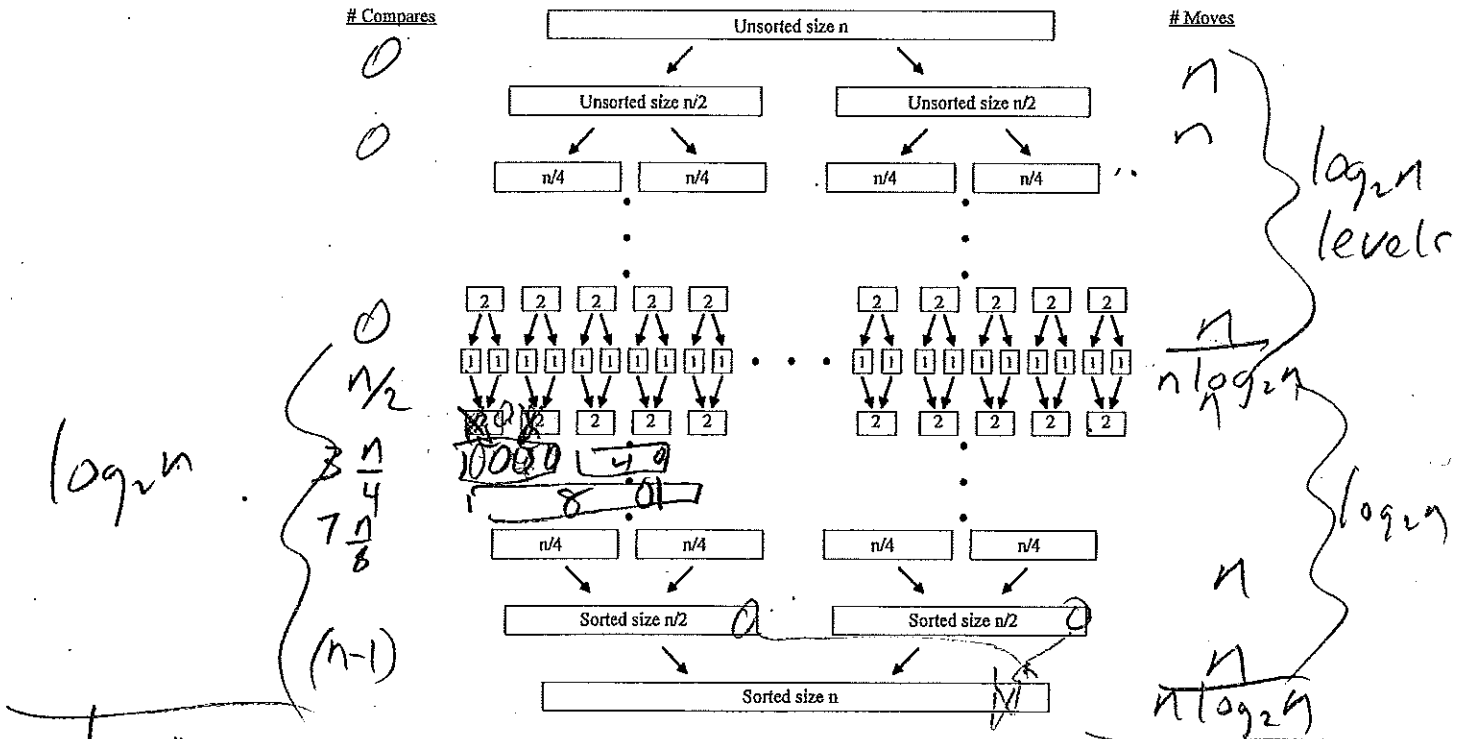
a) (8 points) Insert "Paul Gray" and then "Sarah Diesburg" using Linear (on left) and Quadratic (on right) probing.



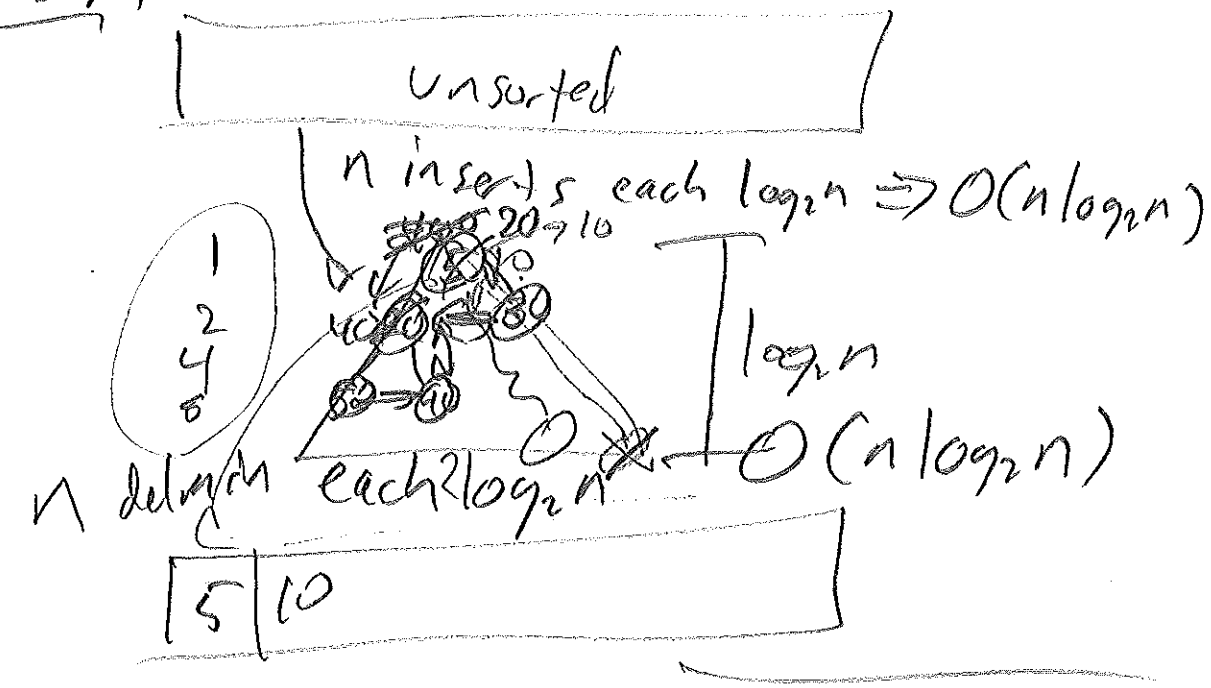
b) (7 points) What is the purpose of requiring a hash table size that is a power of 2 when using quadratic probing?

So quadratic probing hits every slot in hash table before repeating.

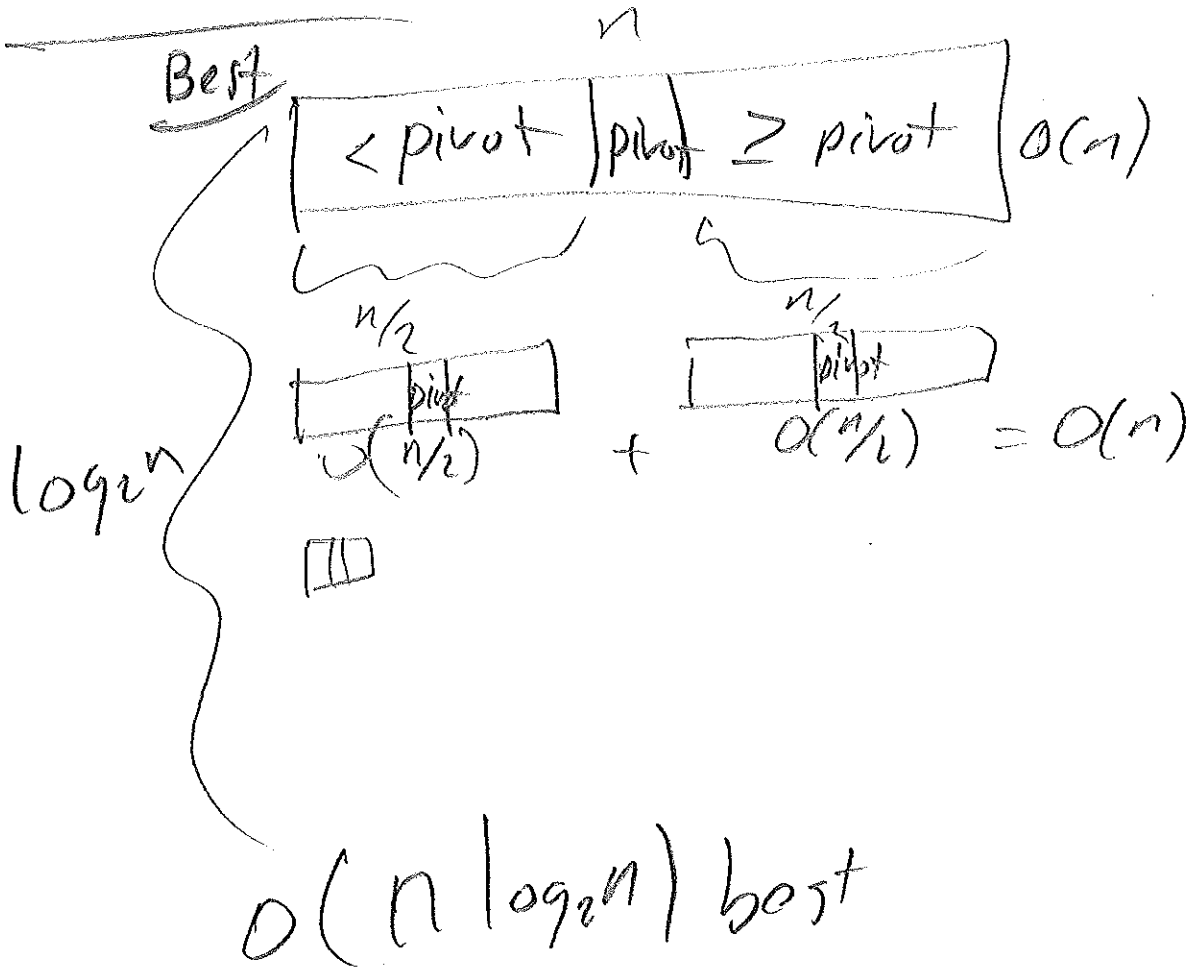
Question 6. (15 points) Use the below diagram to ***explain*** the worst-case big-oh notation of merge sort. Assume "n" items to sort.



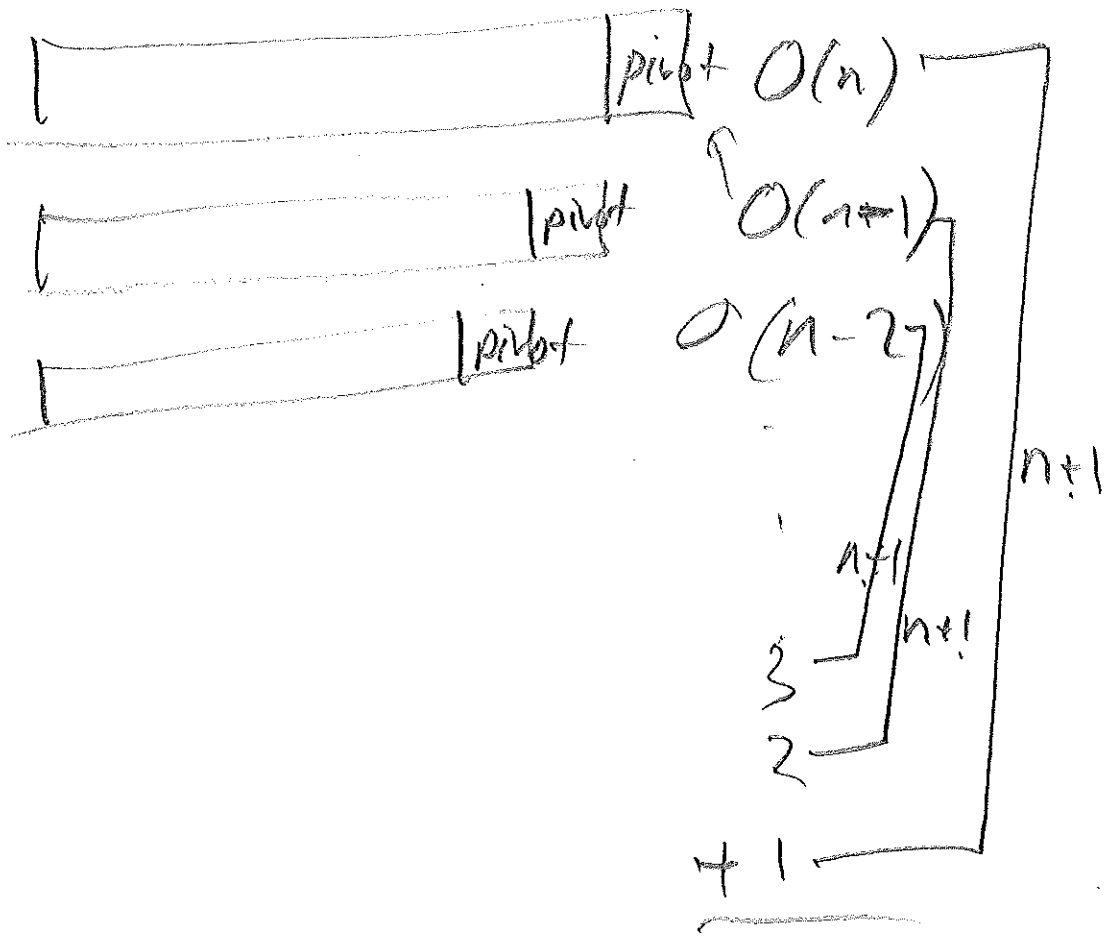
Heap sort



Quicksort



Quicksort worst case



$$\frac{n}{2}\underbrace{(n+1)+(n+1)+\cdots+(n+1)}_{n/2 \text{ pairs}}$$

$$= \frac{n}{2}(n+1) = \frac{n^2}{2}$$

$O(n^2)$ worst case

average $O(n \log n)$

Fall 2015

Name: _____

Data Structures - Test 2

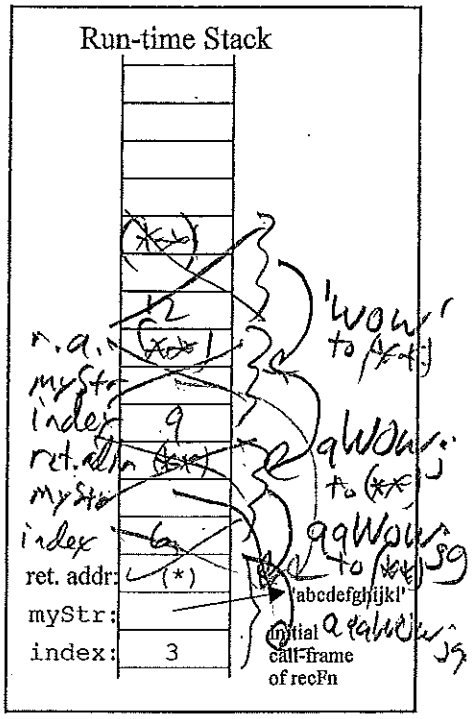
Question 1. (10 points) What is printed by the following program?

```
def recFn(myStr, index):
    print(index)
    if index >= len(myStr):
        return "WOW"
    else:
        return myStr[0] + recFn(myStr, index + 3) + myStr[index]

print("result =",recFn("abcdefghijkl", 3))
```

Output:

```
3
6
9
result = aaaWOWjgd
```



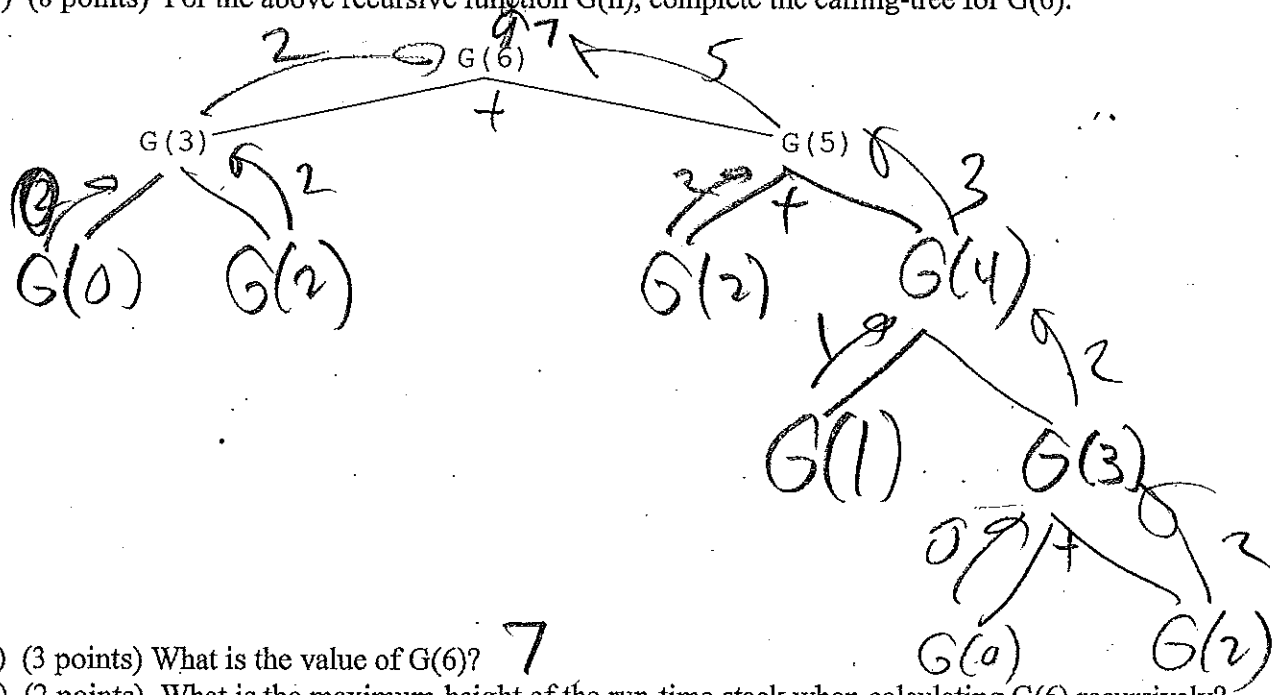
Question 2. a) (12 points) Write a recursive Python function to compute the following mathematical function, G(n):

G(n) = n for all values of n ≤ 2 (e.g., G(1) value is 1)

G(n) = G(n-3) + G(n-1) for all values of n > 2.

```
def G(n):
    if n <= 2:
        return n
    else:
        return G(n-3) + G(n-1)
```

b) (8 points) For the above recursive function G(n), complete the calling-tree for G(6).



c) (3 points) What is the value of G(6)? 7

d) (2 points) What is the maximum height of the run-time stack when calculating G(6) recursively? 5