Spring 2013 Name: ______________________

Data Structures - Test 2

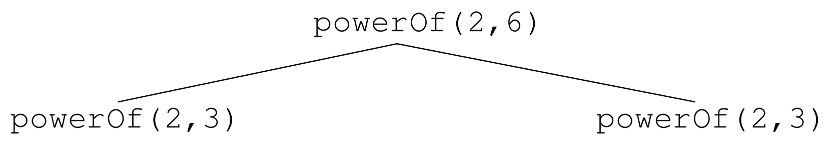
Question 1. Write a recursive Python function to calculate a^n (where n is an integer) based on the formulas:

$$
\begin{array}{ll}
a^0 = 1, & \text{for n = 0} \\
a^1 = a, & \text{for n = 1} \\
a^n = a^{n/2}a^{n/2}, & \text{for even n > 1} \\
a^n = a^{(n-1)/2}a^{(n-1)/2}a, & \text{for odd n > 1}
\end{array}
$$

a) (12 points) Complete the below `powerOf` **recursive** function

```
def powerOf(a, n):
```

b) (8 points) For the above recursive `powerOf` function, complete the calling-tree for `powerOf` (2, 6).



c) (5 points) Suggest a way to speedup the above `powerOf` function.

Question 2. (10 points.) Consider the following insertion sort code which sorts in ascending order.

```
def insertionSort(myList):
    """Rearranges the items in myList so they are in ascending order"""

    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

        # Insert the itemToInsert at the correct spot
        myList[testIndex + 1] = itemToInsert
```

a) What initial arrangement of items causes the overall worst-case performance of insertion sort?

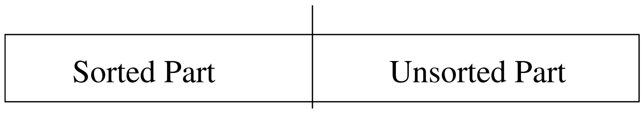
b) What is the worst-case $O(\ )$ notation for insertion sort?

c) What initial arrangement of items causes the overall best-case performance of insertion sort?

d) What is the best-case $O(\ )$ notation for insertion sort?

Question 3. (25 points) Write a variation of selection sort that:

- sorts in descending order (largest to smallest)
- builds the sorted part on the left-hand side of the list by having each pass of the outer loop do the following:



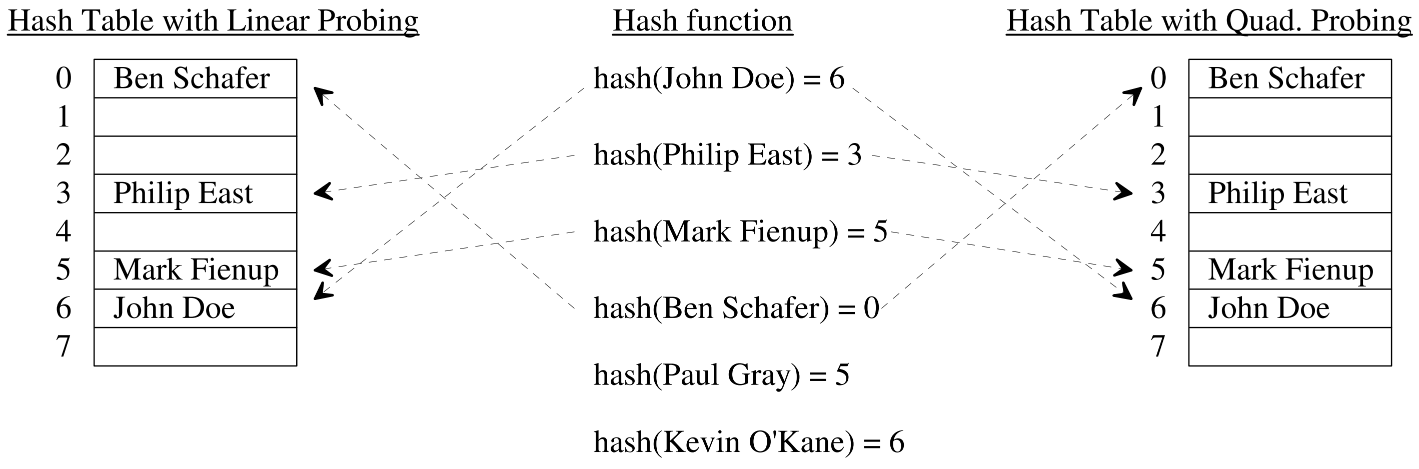
1) Inner loop: that scans the unsorted part to find
   the index of the largest item in the unsorted part
2) Swap the first item in the unsorted part with
   the largest item in the unsorted part that was found in (1)

```
def selectionSort(myList):
```

Question 4. (20 points) Recall the common rehashing strategies we discussed for open-address hashing:

Strategy	Description
linear probing	Check next spot (counting circularly) for the first available slot, i.e., (home address + (rehash attempt #)) % (hash table size)
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) Insert "Paul Gray" and then "Kevin O'Kane" using Linear (on left) and Quadratic (on right) probing.

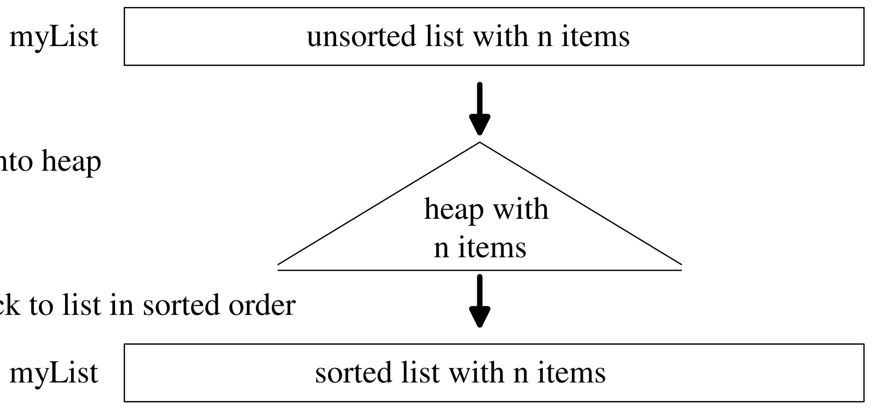


b) Explain why the average/expected search time for hashing is O(1).

Question 5. (20 points) Heap sort uses a min-heap to sort a list. (`BinHeap` methods: `BinHeap(), insert(item), delMin(), isEmpty(), size()`)

<u>Generl idea of Heap sort:</u>

1. Create an empty heap
2. Insert all n list items into heap
3. delMin heap items back to list in sorted order



a) What is the overall *O*( ) for heap sort?

b) Explain your *O*( ) answer for part (a).