Data Structures - Test 2

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

- $a^0 = 1$, for $n = 0$
- $a^1 = a$, for $n = 1$
- $a^n = a^{n/2}a^{n/2}$, for even $n > 1$
- $a^n = a^{(n-1)/2}a^{(n-1)/2}a$, for odd $n > 1$

(a) (12 points) Complete the below recursive function

```python
def powerOf(a, n):
    if n == 0:
        return 1
    elif n == 1:
        return a
    elif (n % 2) == 0:
        return powerOf(a, n // 2) * powerOf(a, n // 2)
    else:
        return powerOf(a, (n-1)//2) * powerOf(a, (n-1)//2) * a
```

(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.

- Don't call `powerOf` twice with same parameters
- Change
- Dynamic programming

$$8 \times 8 = 64$$
Question 2. (10 points) Consider the following insertion sort code which sorts in ascending order.

```python
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 insertion sort to have the overall worst-case performance and what is the overall worst-case $O(\cdot)$ notation?  

Initially in descending order -- away inserts at index 0

$$1+2+3+\cdots+(n-3)+(n-2)+(n-1) = n \times \frac{(n-1)}{2} \in O(n^2)$$

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

$c) What initial arrangement of items causes the insertion sort to have the overall best-case performance and what is the overall best-case $O(\cdot)$ notation? 

2. Initially in ascending order already. -- while loop never entered.

$$O(n)$$

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

$O(n)$

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)

```python
def selectionSort(myList):
    for FirstUnsortedIndex in range(len(myList)-1):
        maxIndex = FirstUnsortedIndex

        for testIndex in range(FirstUnsortedIndex+1, len(myList)):
            if myList[testIndex] > myList[maxIndex]:
                maxIndex = testIndex
                temp = myList[FirstUnsortedIndex]
                myList[FirstUnsortedIndex] = myList[maxIndex]
                myList[maxIndex] = temp
```
Question 4. (20 points) Recall the common rehashing strategies we discussed for open-address hashing:

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear probing</td>
<td>Check next spot (counting circularly) for the first available slot, i.e.,</td>
</tr>
<tr>
<td></td>
<td>(home address + (rehash attempt #)) % (hash table size)</td>
</tr>
<tr>
<td>quadratic probing</td>
<td>Check the square of the attempt-number away for an available slot, i.e.,</td>
</tr>
<tr>
<td></td>
<td>[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</td>
</tr>
</tbody>
</table>

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())

**General idea of Heap sort:**

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

**Unsorted list with n items**

<table>
<thead>
<tr>
<th>myList</th>
</tr>
</thead>
</table>

**Sorted list with n items**

<table>
<thead>
<tr>
<th>myList</th>
</tr>
</thead>
</table>

a) What is the overall $O(\cdot)$ for heap sort? $O(n \log_2 n)$

b) Explain your $O(\cdot)$ answer for part (a).

Step 1 is $O(1)$

Step 2 does $n$ calls to insert where each insert is at most $\log_2 n$ loops since that’s the height of the heap. Thus, $n \times \log_2 n$.

Step 3 is similar to Step 2: $n$ calls to delMin where each is $O(\log_2 n) \Rightarrow n \log_2 n$.