Questions

1. (a) Suppose you wish to “count down” the numbers from a given $n$ down to 1. You can use a while loop to do this:

   ```java
   countdown(n){
       while (n > 0) {
           print n
           n-- }
   }
   ```

   Write a recursive version of this algorithm.

   (b) Now write a recursive `countUp(n)` algorithm which counts up from 1 to n. This is slightly trickier than it seems.

2. Write a recursive algorithm that prints out a consecutive sequence of elements in an array. The method has three parameters: the array name, and the first and last indices to print from i.e. `displayArray(a, first, last)`

3. Here is an alternative `reverse()` method for the `SLinkedList<E>` class (see online code from linked list exercises), which reverses the order of nodes in a singly linked list. The method calls a recursive helper method `reverseRecursive` which does most of the work.

   Write this helper method. (If solution is different from the given one, then you should add your method to the `SLinkedList` class and test/debug it.)

   ```java
   public void reverse(){
       SNode<E> oldHead = this.head;
       SNode<E> oldTail = this.tail;
       reverseRecursiveHelper(oldHead);
       oldHead.setNext(null);
       SNode<E> tmp = oldHead;
       this.head = oldTail;
       this.tail = tmp;
   }
   ```

4. Consider a simplified version of the game “20 questions”, in which one person has a number in mind from 0 to $2^{20} - 1$. You can easily find that number with your twenty questions e.g. by asking for the $i$th bit of the number. (This is essentially binary search.)

   Here is a slightly different game. Suppose I am thinking of a positive integer $n$ but I don’t give you a bound on the number. It is easy for you figure out the number using $n$ questions, namely question $i$ is “is it the number $i$ ?”. Give a faster algorithm, namely one that runs in time proportional to $\log n$. 
5. Suppose that you are given an array of $n$ different numbers that strictly increase from index 0 to index $m$, and strictly decrease from index $m$ to index $n - 1$, where $n$ is known but $m$ is unknown. Note that there is a unique largest number in such a list, and it is at index $m$. Here are a few examples.

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

Provide the missing pseudocode below of a recursive algorithm that returns the index $m$ of the largest number in the array, in time proportional to $\log n$.

The algorithm is initially called with $\text{low} = 0$, $\text{high} = n-1$.

```java
findM( a, low, high ){ // array is a[], assume low <= high
    if (low == high)
        return low
    else{
        // ADD YOUR CODE HERE (AND ONLY HERE)
    }
}
```

6. Use mathematical induction to prove the following:

For all $n \geq 1$,

$$1 + 3 + 5 + \cdots + (2n - 1) = n^2.$$
Answers

1. (a) `countdown(n){
    if (n > 0) {
        print n
        countdown(n - 1)
    }
}

Did you remember the base case?

(b) `countUp(n){
    if (n > 0) {
        countUp(n - 1)
        print n
    }
}

Did you get the order of the instructions correct?

2. `displayArray( a, first, last){
    print a[first])
    if (first < last)
        displayArray(a, first+1, last)
}

Alternatively, you could do it like this:

`displayArray( a, first, last){
    if (first < last)
        displayArray(a, first, last-1);
        print a[last])
}

3. `public void reverseRecursiveHelper(SNode<E> head){
    SNode<E> tmp = head;
    if (head.next != null){
        head = head.next;
        reverseRecursiveHelper(head);
        head.next = tmp;
    }
}
4. The idea of the algorithm is to guess increasing powers of 2 until the power of 2 is bigger than the number. Then do a binary search (backwards).

\[
\begin{align*}
guess &= 1 \\
answer &= false \\
while (answer == false)\{ \\
    answer &= (guess > n) \quad // \ i.e. \ evaluate \ boolean \ expression \\
    guess &= guess*2 \\
}\} \\
// \ To \ reach \ here, \ we \ must \ have \ guess/2 \leq n < guess \\
binarySearch(n, [guess/2, guess])
\end{align*}
\]

Both the while loop and the binary search take time proportional to \(\log n\).

5. Here are two different solutions.

// SOLUTION 1

\[
\begin{align*}
if (high-low == 1)\{ \\
    if (a[low] < a[high]) \\
        return high \\
    else \\
        return low \\
}\} \\
else\{ \\
    mid &= (low + high)/2 \\
    if (a[mid-1] < a[mid])\{ \\
        return findM(a, mid, high) \\
    } \\
    else \\
        return findM(a, low, mid) \\
\}
\end{align*}
\]

// -------- SOLUTION 2 --------

\[
\begin{align*}
mid &= (low + high)/2 \\
if (a[mid] < a[mid+1])\{ \\
    return findM(a, mid+1, high) \\
} \\
else \\
    return findM(a, low, mid)
\end{align*}
\]
6. The base case of \( n_0 = 1 \) is obvious, since there is only a single term on the left hand side, i.e. \( 1 = 1^2 \). The induction hypothesis is the statement \( P(k) \):

\[
P(k) \equiv "1 + 3 + 5 + \cdots + (2k - 1) = k^2 "
\]

To prove the induction step, we show that if \( P(k) \) is true, then \( P(k + 1) \) must also be true.

\[
\sum_{i=1}^{k+1} (2i - 1) = 2(k + 1) - 1 + \sum_{i=1}^{k} (2i - 1) \\
= 2(k + 1) - 1 + k^2, \quad \text{by the induction hypothesis} \\
= 2k + 1 + k^2 \\
= (k + 1)^2.
\]

Thus, the induction step is also proved, and so we’re done.