Recursion (continued)

For each of the functions given below, do the following:

- Trace the function for n = 4 (show activation records on stack, etc.)
- Identify pre- and post-conditions of the function
- Identify variant expression
- Identify threshold
- Prove that there isn't infinite recursion
- Prove that function is correct via inductive reasoning

```
int sumOdds(const int n) {
int result = 1;
  if (n != 1)
    result = sumOdds(n-1) + (2 * n – 1);
  return(result);
}
```

Trace: n = 4 returns 16 (which is 1 + 3 + 5 + 7)

Preconditions: n ≥ 1

Postconditions: sum of 1st n (positive) odd numbers is returned

Variant expression: n

Threshold: 1

No infinite recursion: n decreases by 1 in sumOdds(n-1); when n = 1, just return 1 (no more recursive calls)

Correctness: only terminates w/o recursive call if n = 1 (in which case 1 is the sum of 1st positive odd number); sum of 1st n positive odd numbers is sum of 1st n-1 odd numbers + nth odd number (and nth odd number = (2 * n – 1))

```
void reverse(const String s) {
  if (s.length( )  >  0) {
     reverse(s.substr(1));   // substr(1) gives substring that begins at s[1] to end of string
     cout << s[ 0 ];
  }
}
```

Trace: when s = "ABC", outputs 'C', 'B', 'A'

Preconditions: s is a string of 0 or more chars

Postconditions: chars of s are output in reverse order

Variant expression: s.length( )

Threshold: 0

No infinite recursion: length of s decreases by 1 in reverse(s.substr(1)); when length = 0, just return (no more recursive calls)

Correctness: only terminates w/o recursive call if empty string (in which case doesn't output anything); outputting reverse of s accomplished by outputting reverse of substring that comes after 1^{st} char in s, followed by outputting 1^{st} char of s

Runtime Analysis of Recursive Functions

Express runtime function T(n) as a **recurrence equation**, then solve to find big-O

<u>Example</u>: sumOdds(n)

$$T(n) = \begin{cases} c & \text{if } n \le 1 \\ T(n-1) + c & \text{if } n > 1 \end{cases}$$

So for n > 1,

$$\begin{aligned} T(n) &= T(n-1) + c \\ &= (T(n-2) + c) + c \\ &= ((T(n-3) + c) + c) + c \\ &= \dots \\ &= T(1) + ((n-1) * c) \\ &= c + ((n-1) * c) \\ &= cn \end{aligned}$$

which is O(n)