Problem: Palindrome

```
boolean isPalindrome(String word) {
  if(word.length() == 0 || word.length() == 1) {
    /* base case */
    return true;
  }
  else {
    /* recursive case */
    char firstChar = word.charAt(0);
    char lastChar = word.charAt(word.length() - 1);
    String middle = word.substring(1, word.length() - 1);
    return
        firstChar == lastChar
        /* See the API of java.lang.String.substring. */
        && isPalindrome(middle);
  }
}
```

Problem: Reverse of a String

```
String reverseOf (String s) {
  if(s.isEmpty()) { /* base case 1 */
    return "";
  }
  else if(s.length() == 1) { /* base case 2 */
    return s;
  }
  else { /* recursive case */
    String tail = s.substring(1, s.length());
    String reverseOfTail = reverseOf(tail);
    char head = s.charAt(0);
    return reverseOfTail + head;
  }
}
```

Problem: Number of Occurrences

```
int occurrencesOf(String s, char c) {
  if(s.isEmpty()) {
    /* Base Case */
    return 0;
  }
  else {
    /* Recursive Case */
    char head = s.charAt(0);
    String tail = s.substring(1, s.length());
    if(head == c) {
      return 1 + occurrencesOf(tail, c);
    }
    else {
      return 0 + occurrencesOf(tail, c);
    }
  }
}
```

Recursion on an Array: Passing new Sub-Arrays

```
void m(int[] a) {
  if(a.length == 0) { /* base case */ }
  else if(a.length == 1) { /* base case */ }
  else {
    int[] sub = new int[a.length - 1];
    for(int i = 1; i < a.length; i ++) { sub[i - 1] = a[i]; }
    m(sub) } }
```

Say a1 = {}, consider m(a1)

Say a2 = {A, B, C}, consider m(a2)

Recursion on an Array: Passing Same Array Reference

```
void m(int[] a, int from, int to) {
  if(from > to) { /* base case */ }
  else if(from == to) { /* base case */ }
  else { m(a, from + 1, to) } }
```

Say a1 = {}, consider m(a1, 0, a1.length - 1)

Say a2 = {A, B, C}, consider m(a2, 0, a2.length - 1)