Lecture 23 – Nov 27

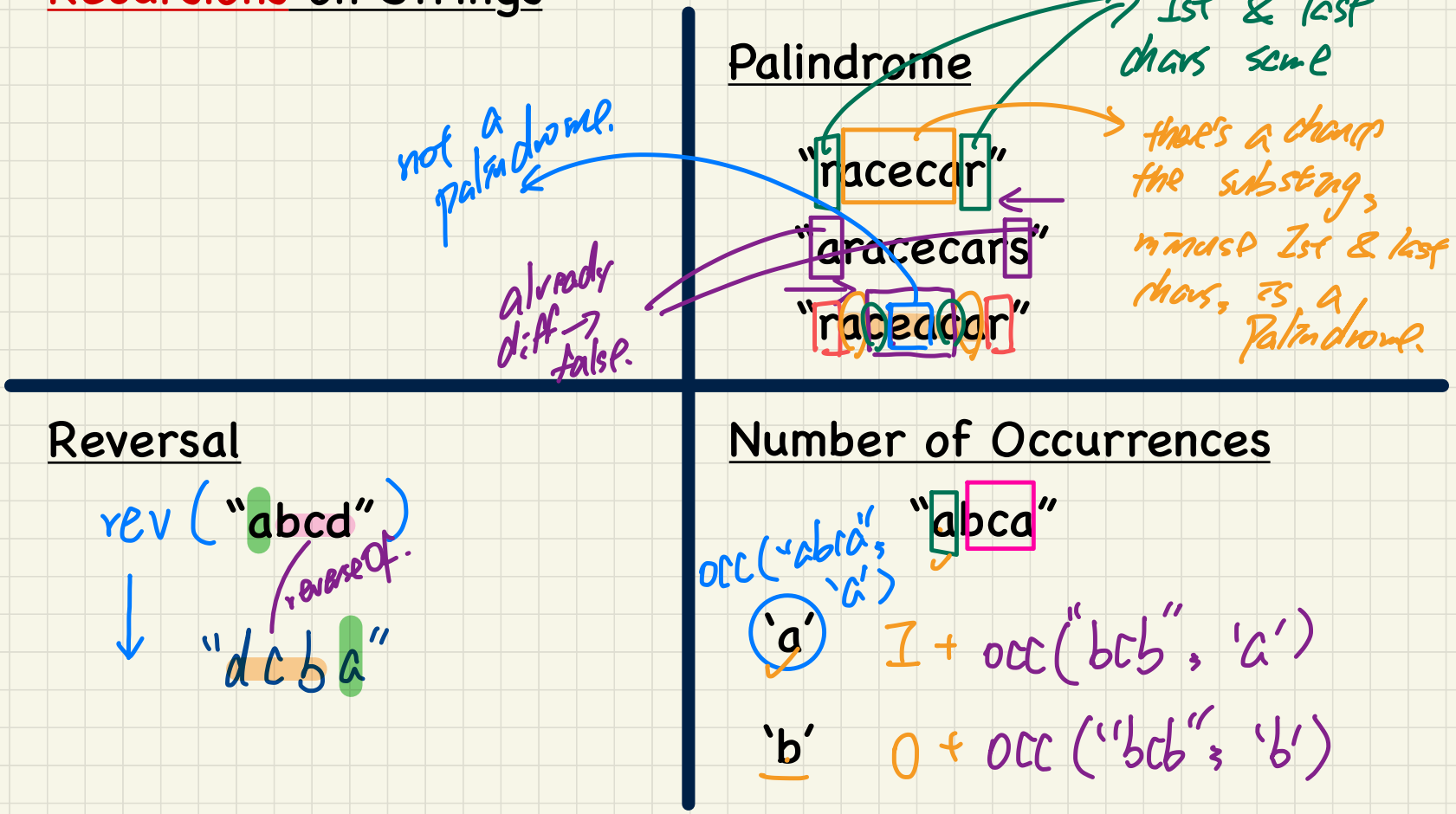
Recursion

Examples: reverseOf, occurrencesOf
Recursion on Arrays: Creating Sub-Arrays
Recursion on Arrays: Call by Value

Announcements/Reminders

- Today's class: notes template posted
- Lab5 released (deadline: Dec 2; grace period until Dec 9)
  + Required background study: abstract classes & interfaces

Recursions on Strings



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);
  }
}
```

recursive case.

if(fC == lC){ return isP(middle); } else { return false; }

isP("abcba")

a==a && isP("bcb")

T

b==b && isP("c")

T

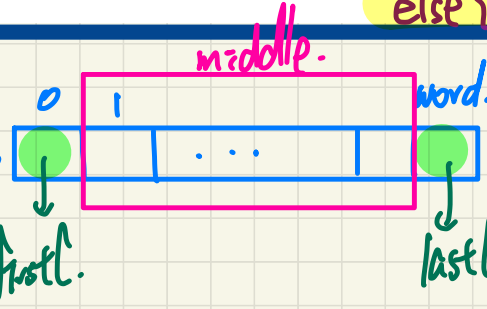
true.

T

T.

1) LHS is true → excl RHS

2) LHS is word → false → skip RHS

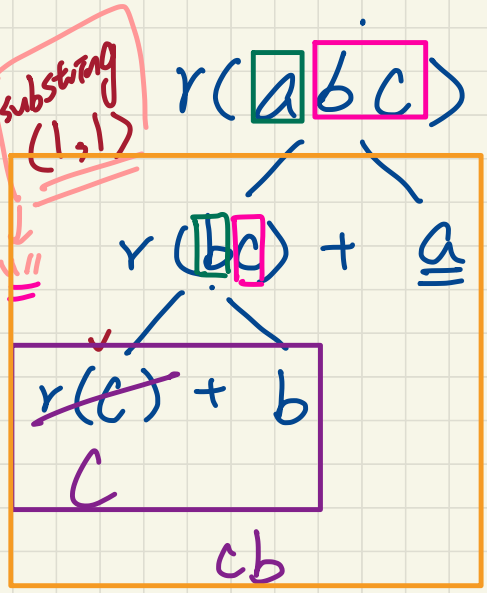


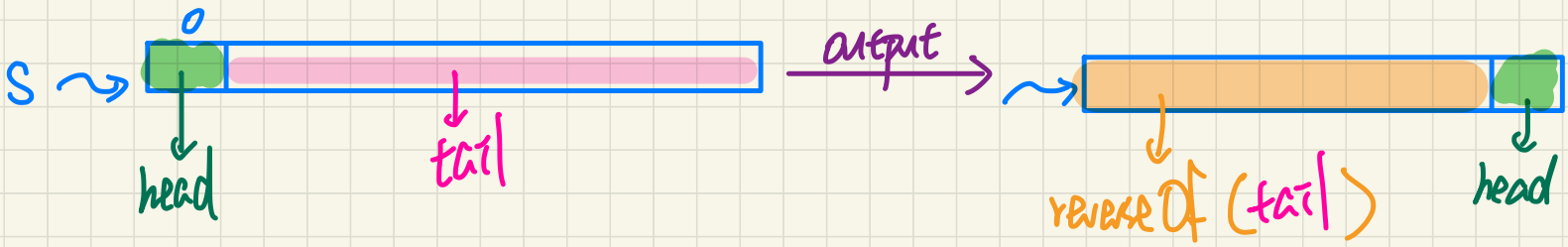
Problem: Reverse of a String

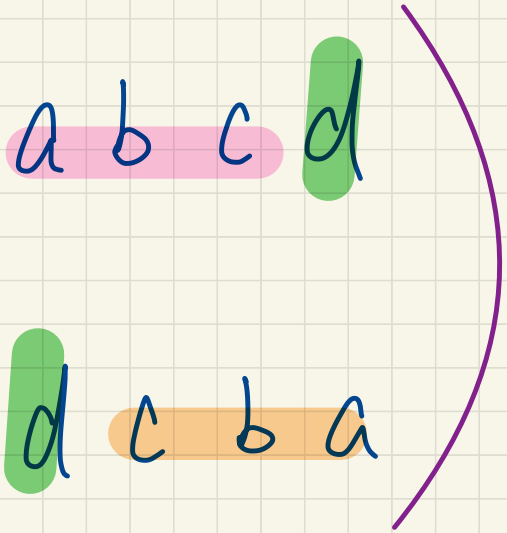
```
String reverseOf (String s) {
  if(s.isEmpty()) { /* base case 1 */
    return "";
  }
  else if(s.length() == 1) { /* base case 2 */
    return s;
  }
  else { /* recursive case */
    String tail = s.substring(1, s.length());
    String reverseOfTail = reverseOf(tail);
    char head = s.charAt(0);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

base cases

what if s.length == 1 "c".substring(1,1)







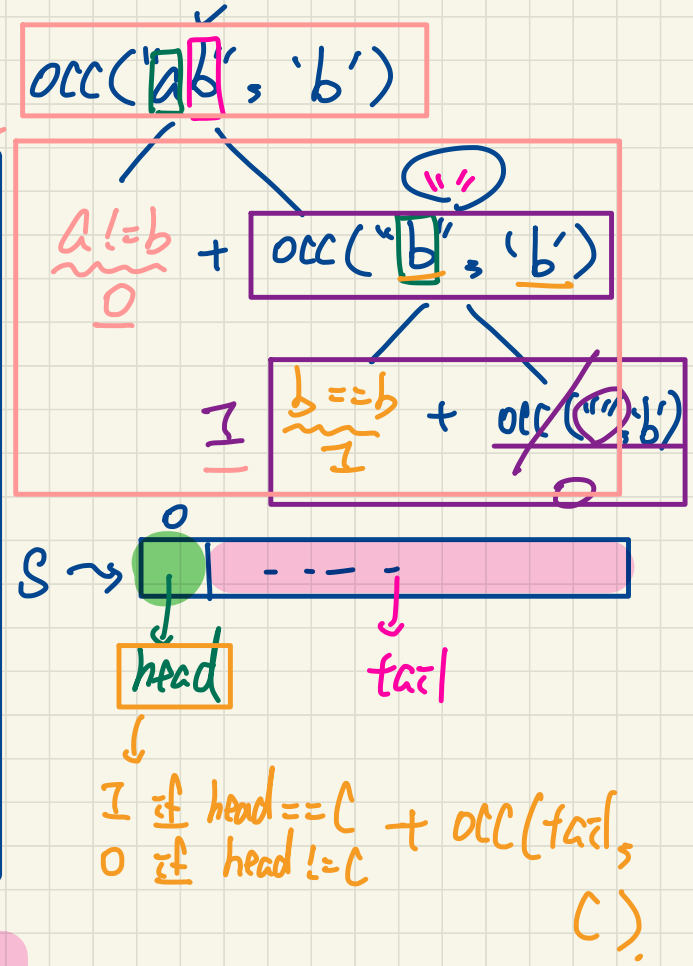
alternative approach for implementing reverseOf.

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);
    }
  }
}
```

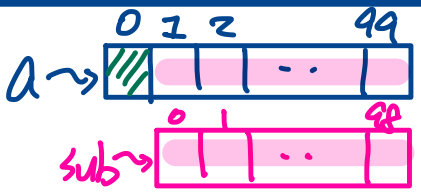
base.

return head == c ? 1 + occ(tail, c) : occ(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) } }
```

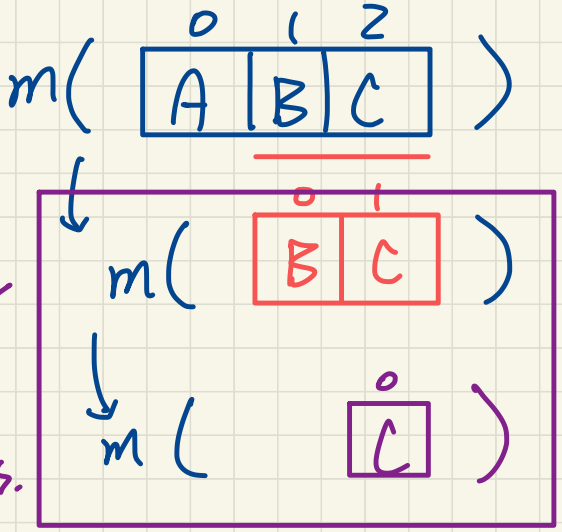


Say a1 = {}, consider m(a1)

↳ base case #1.

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

loop time & space wasted to create sub-arrays between R.C.s.



Recursion on an Array: Passing Same Array Reference

"sub-arrays"

```
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) } }
```

3, 2
2, 2

sub-array of size 0

↳ sub-array of size 1

range of indices to inspect in array 'a'

recursive call

* call by value : passing the ref. of 'a' around.

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

a1.length == 0

1st index

last index

from > to

empty range of indices to look at

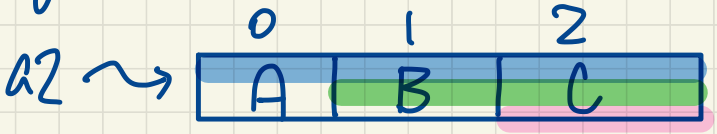
↳ empty sub-array

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

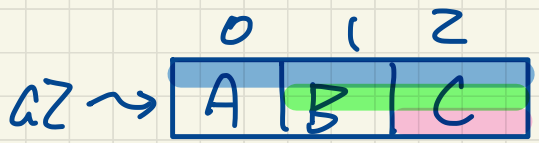
0 1 2

a2.length == 3

2



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



m(a2, 0, 2)
↳ input array of length 3

m(a2, 1, 2)
↳ sub-array of length 2

m(a2, 2, 2)
↳ base case (as if sub-array of length 1).

As recursive calls are made, narrow down the intervals.