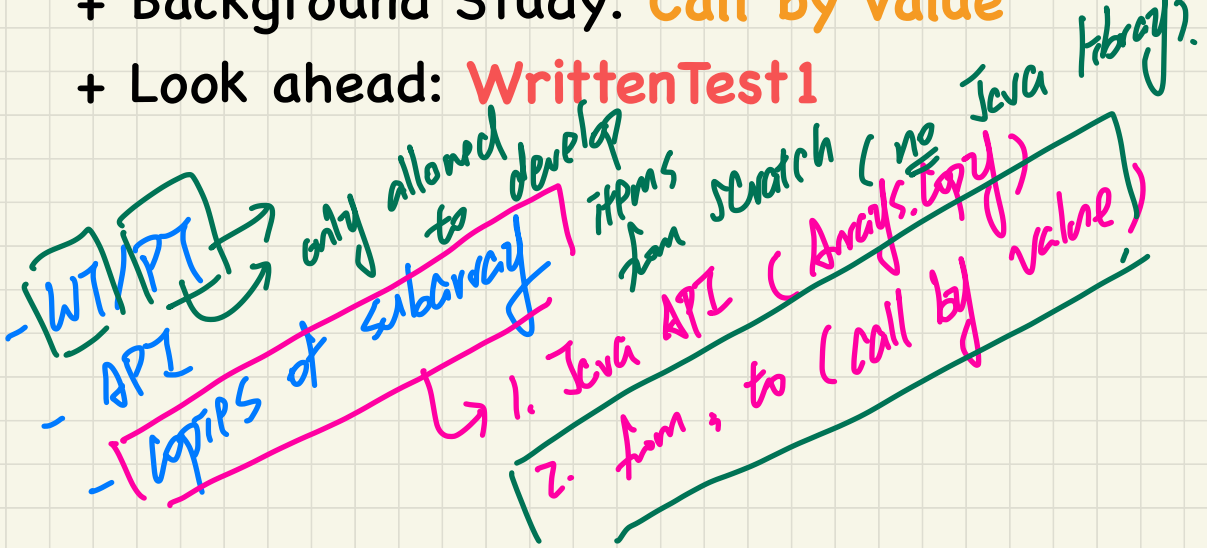
Lecture 3 - Monday, January 16

Announcements

- **Assignment 1** to be released next Monday
  - + Background Study: **Basic Recursion**
  - + Background Study: **Call by Value**
  - + Look ahead: **WrittenTest1**

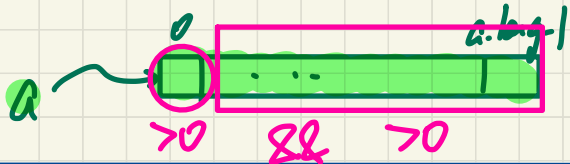


Tracing Recursion

1. Stack (e.g. factorial, fib)

2. tree-like drawing

Tracing Recursion: allPositive



Say a = {}

allPositive(a)

allPH(a,0,-1)

a.length == 0

public

private

recursive helper method

```
boolean allPositive(int[] a) {
  return allPositiveHelper(a, 0, a.length - 1);
}

boolean allPositiveHelper(int[] a, int from, int to) {
  if (from > to) { /* base case 1: empty range */
    return true;
  }
  else if(from == to) { /* base case 2: range of one element */
    return a[from] > 0;
  }
  else { /* recursive case */
    return a[from] > 0 && allPositiveHelper(a, from + 1, to);
  }
}
```

0 > -1

current ele. being positive

conjunction

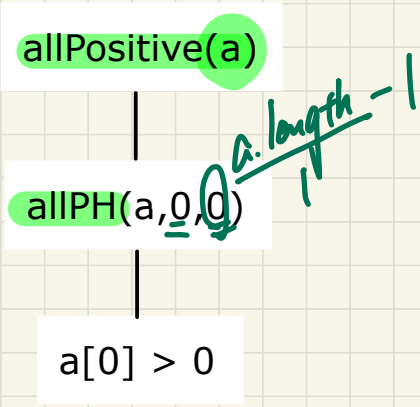
the rest of the array contains all positive.

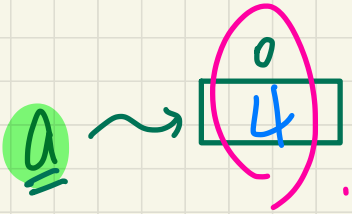
a → 1

a.length == 0

Tracing Recursion: allPositive

Say a = {4}





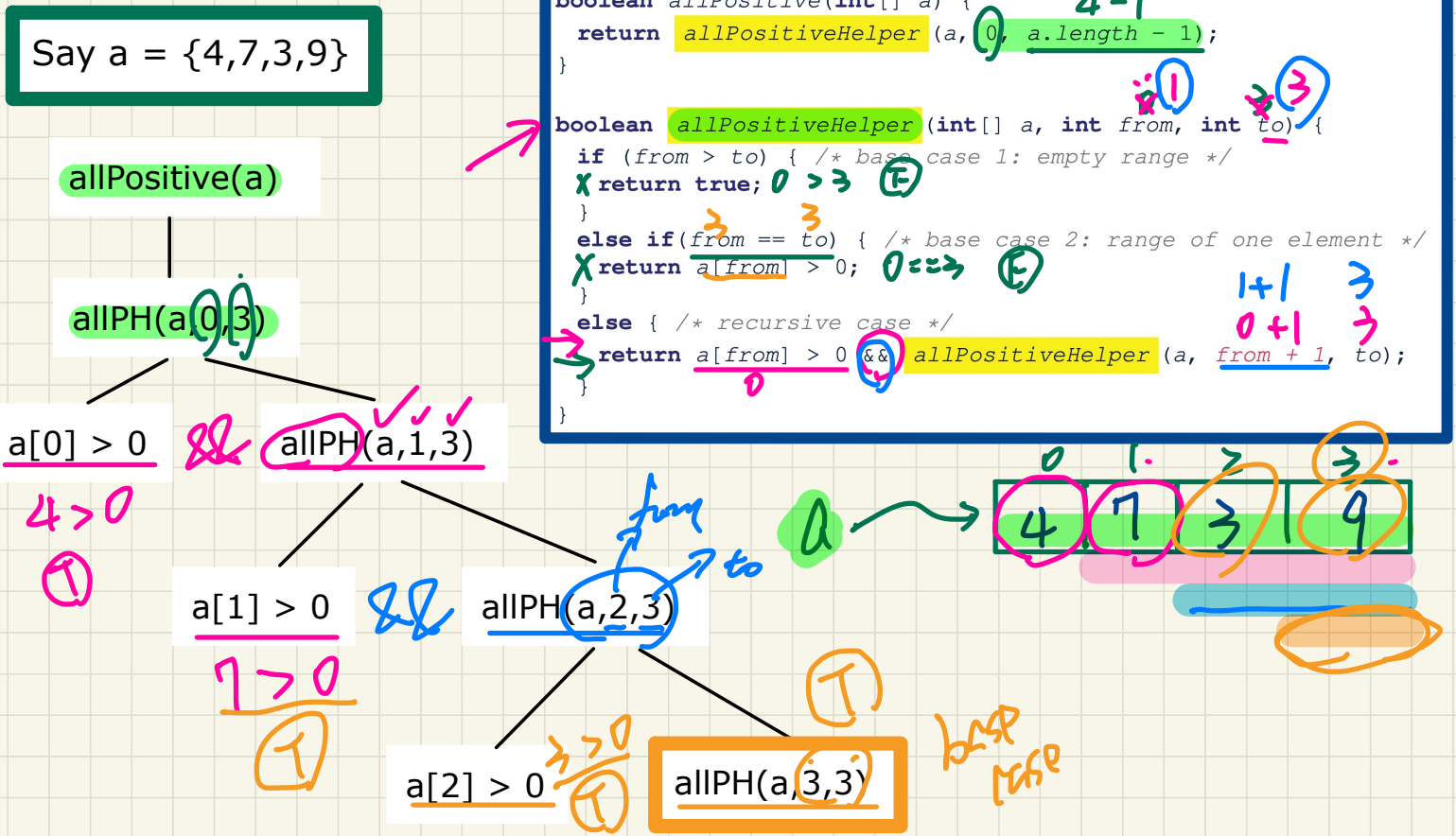

```boolean allPositive(int[] a) {
  return allPositiveHelper(a, 0, a.length - 1);
}

boolean allPositiveHelper(int[] a, int from, int to) {
  if (from > to) { /* base case 1: empty range */
    return true;
  }
  else if(from == to) { /* base case 2: range of one element */
    return a[from] > 0;
  }
  else { /* recursive case */
    return a[from] > 0 && allPositiveHelper(a, from + 1, to);
  }
}
```

Tracing Recursion: allPositive

Bring an example of tracing via stack.

Say a = {4,7,3,9}

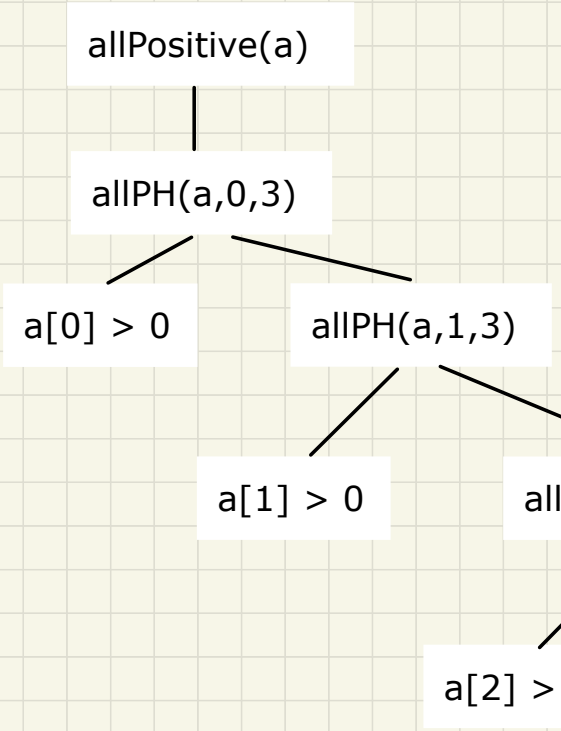
```
boolean allPositive(int[] a) {
  return allPositiveHelper(a, 0, a.length - 1);
}

boolean allPositiveHelper(int[] a, int from, int to) {
  if (from > to) { /* base case 1: empty range */
    return true;
  }
  else if(from == to) { /* base case 2: range of one element */
    return a[from] > 0;
  }
  else { /* recursive case */
    return a[from] > 0 && allPositiveHelper(a, from + 1, to);
  }
}
```



Tracing Recursion: allPositive

Exercise

Say a = {5,3,-2,9}




```
boolean allPositive(int[] a) {
  return allPositiveHelper(a, 0, a.length - 1);
}

boolean allPositiveHelper(int[] a, int from, int to) {
  if (from > to) { /* base case 1: empty range */
    return true;
  }
  else if(from == to) { /* base case 2: range of one element */
    return a[from] > 0;
  }
  else { /* recursive case */
    return a[from] > 0 && allPositiveHelper(a, from + 1, to);
  }
}
```

Lecture

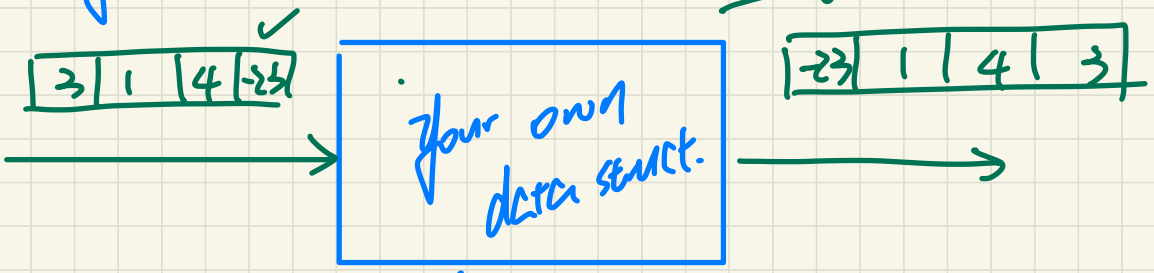
Asymptotic Analysis of Algorithms

Measuring Running Time via Experiments

- Concurrent algorithms

- Arrays vs. linked lists

Sorting - distributed alg.



Sorting

1. insertion sort
2. selection sort
3. merge sort
4. quick sort
5. heap sort → PS: heap.

Arrays

linked lists

SLL

DLL

Example Experiment

space inefficiency ⇒ degraded RT

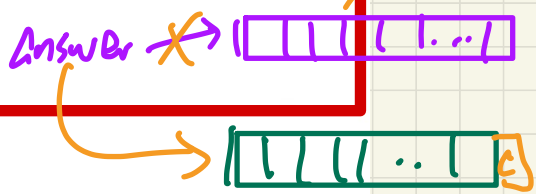
Computational Problem:

- **Input**: A character *c* and an integer *n*
- **Output**: A string consisting of *n* repetitions of character *c*
  e.g., Given input `'*'` and `15`, output `***************`.

Algorithm 1 using *String* Concatenations:

```
public static String repeat1(char c, int n) {
  String answer = "";
  for (int i = 0; i < n; i ++) { answer += c; }
  return answer; }
```

answer = answer + c



Algorithm 2 using *StringBuilder* `append`'s:

```
public static String repeat2(char c, int n) {
  StringBuilder sb = new StringBuilder();
  for (int i = 0; i < n; i ++) { sb.append(c); }
  return sb.toString(); }
```