Lecture 17

Tuesday Nov. 7

Problem	Sub-Problems	Solution
fac(n)	fac(n-1)	n * fac(n-1)
fib(n)	fib(n-1) fib(n-2)	fib(n-1) + fib(n-2)
isP(c1,s,c2)	isP(s)	c1 == c2 && isP(s)
reverse(c1,s)	reverse(s)	reverse(s) + c1
occOf(c1,s,c2)	occOf(s, c2)	occOf("aaab", 'a') = 1 + occOf("aab", 'a', 2) occOf("baab", 'a') = 0 + occOf("aab", 'a', 2)

if c1 == c2
1 + occOf(s, 'a')
else
0 +

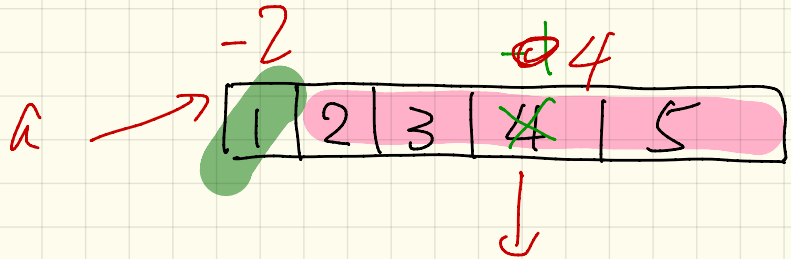
$\forall x : \text{int} \mid x \in \{1, 2, 3\} \cdot x > 2$ F

$\exists x : \text{int} \mid x \in \{1, 2, 3\} \cdot x > 2$ T

(1) $\forall x \mid x \in \emptyset \cdot P(x)$ T ⁓ no counter-ex. e s.t. $e \in \emptyset$ $\hat{\neg P}(e)$

(2) $\exists x \mid x \in \emptyset \cdot P(x)$ F ⁓ no witness e s.t. $e \in \emptyset$ $\hat{P}(e)$

isAllPos(int[] a)



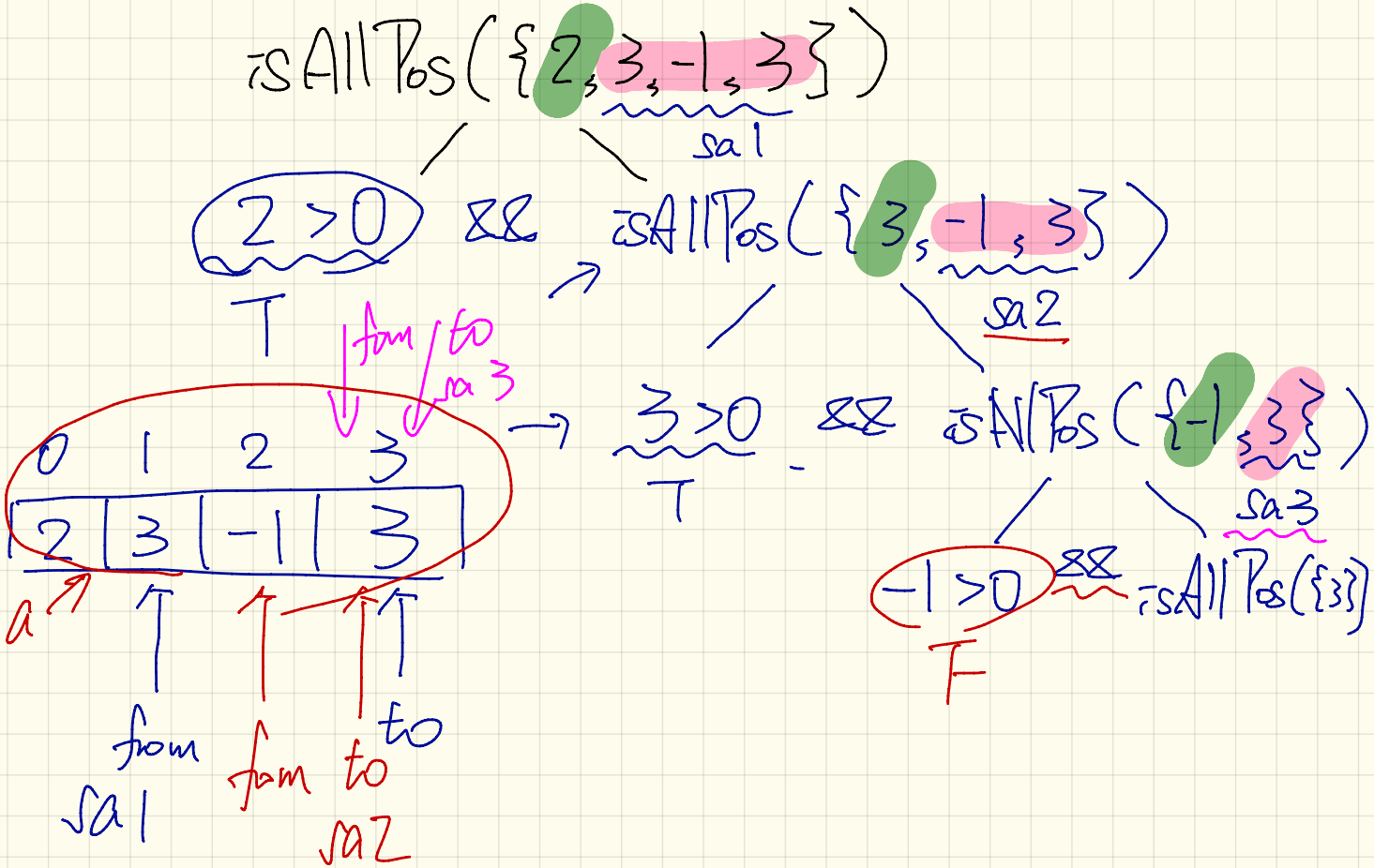
isAllPositive({2, 3, 4, 5})

F

a[0]>0

&&

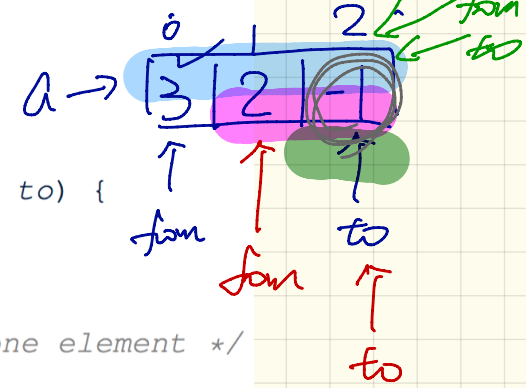
isAllPos({2, 3, 4, 5})

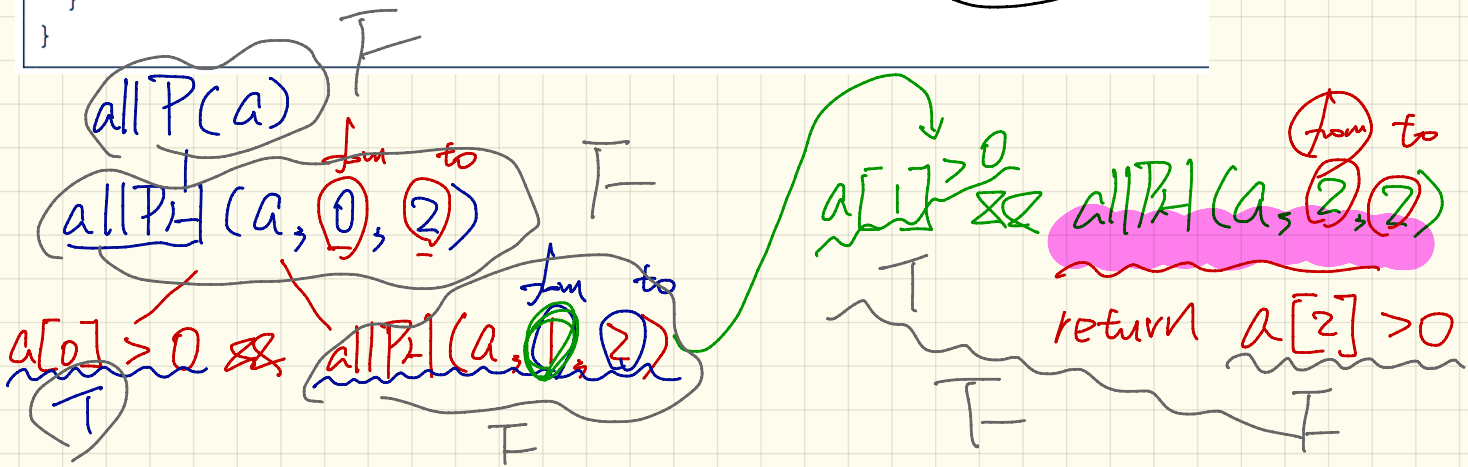

isAllPos({2,3,-1,3})
sa1
2 > 0 && isAllPos({3,-1,3})
T
sa2
from to
sa3
0 1 2 3
2 3 -1 3
3 > 0 && isNIPos({-1,3})
T
sa3
-1 > 0 && isAllPos({3})
F
a
from
sa1
from to
sa2
to

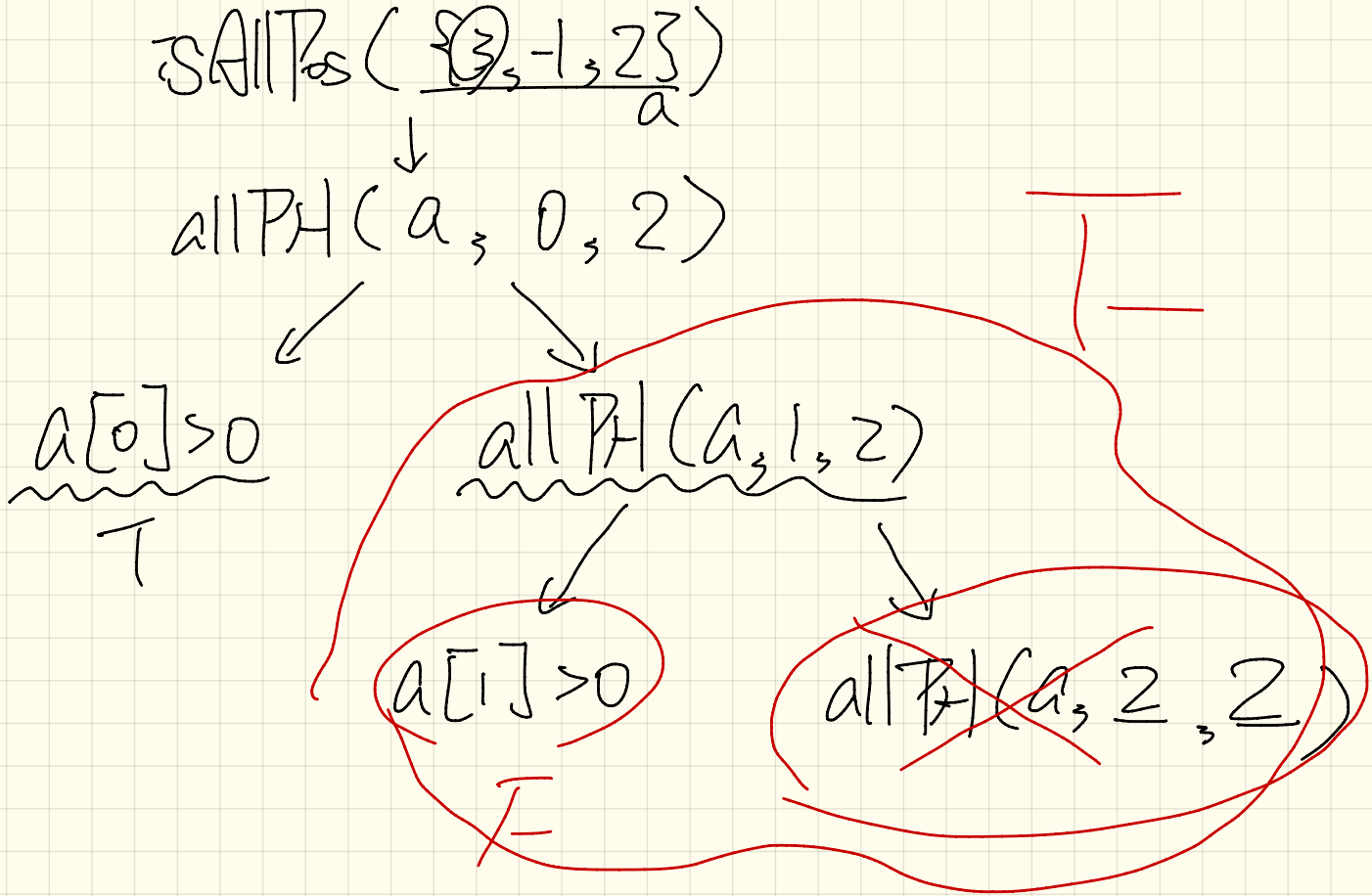
```
boolean allPositive(int[] a) {
  return allPositiveHelper(a, 0, a.length - 1);
}

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);
  }
}
```

empty array.






allPos({3,-1,2}, a)
allPH(a, 0, 2)
a[0]>0
T
allPH(a, 1, 2)
a[1]>0
allPH(a, 2, 2)

int[] @ = new int[0];

allPositive(@)

allPH(a, 0, -1)

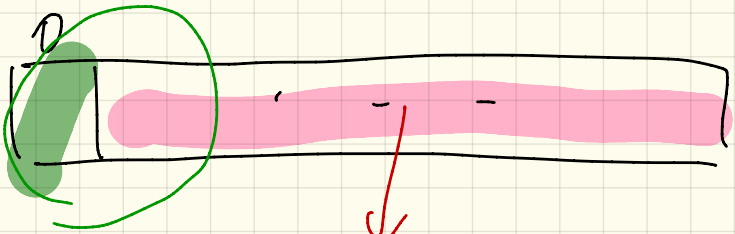
from > to

```
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);
  }
}
```

boolean isSorted(int[] a)

{} → T



a[0] ≤ a[i] && isSorted( )

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

boolean isSortedHelper(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 true;
  }
  else {
    return a[from] <= a[from + 1]
      && isSortedHelper(a, from + 1, to);
  }
}
```