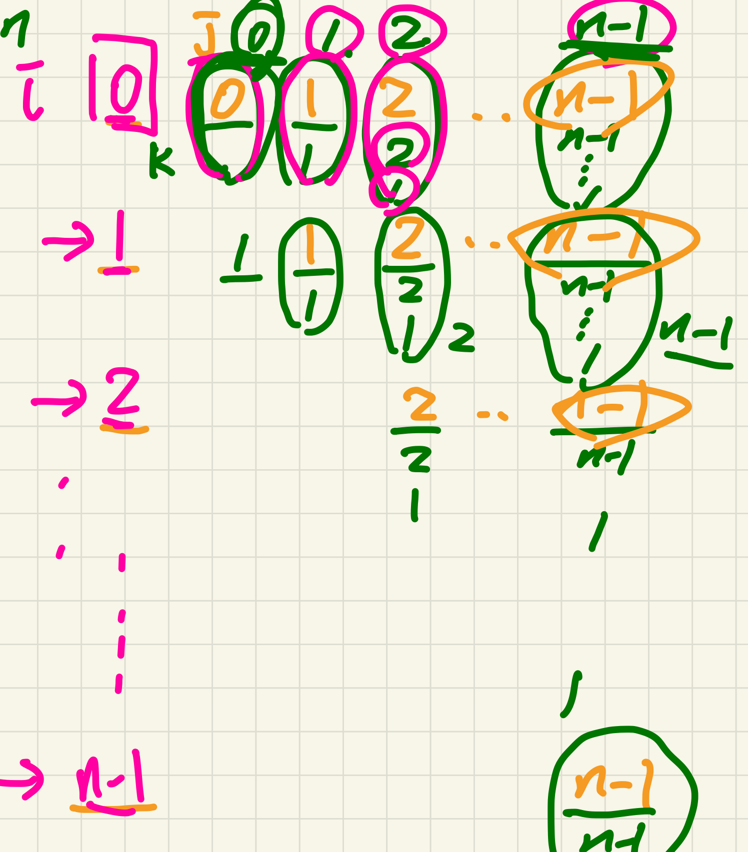
# Exam Review II

Tuesday December 10

$a_1 + a_2 + \cdots a_n = \frac{(a_1 + a_n) * n}{2}$

```
void prog(int[] a, int n)
  for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
      for (int k = j; k > 0; k--) {
        System.out.println(i * j + k);
      }
    }
  }
```



body of loop $O(1)$

$$\sum_{i=0}^{n-1} \frac{(i + (n-1)) * (n-i)}{2}$$

$i=0$: $\frac{(0 + (n-1)) * (n-0)}{2}$

$i=1$: $\frac{(1 + (n-1)) * (n-1)}{2}$

$\vdots$

$i=n-1$: $\frac{((n-1) + (n-1)) * (n-(n-1))}{2}$

$RT = O(\underbrace{1}_{\text{each iteration}} * \# \text{ of iterations})$

Exercise -

```
void prog(int[] a, int n)
  for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
      for (int k = j; k >= 0; k--) {
        System.out.println(i * j + k);
      }
    }
  }
```

k++

k *= 2

< n

Design an algorithm for X

s.t. RT(n · log n)

50%

· RT($n^2$)

# Javadocs.

1. Read Javadocs.
   - @param
   - @return
   - @throws
   - @precondition → assume

```
String m(... ...) {
  ...
}
```

# Unit Testing.

1. Read unit tests.
2. ~~No need to write~~ ~~JUnit tests~~
3. Given problem, come up with test cases.

Option 1.

```
/**
 * @return ...
 * @param x ...
 * @param y ...
 * @throws X ... */
double divide(double x, double y) throws X {
  if (y == 0) { throw new X(...); }
  return x / y;
}
```

defensive about illegal value passed by caller.

when things under your control

```
class MyClass {
  void m1() {
    System.println( divide(input, 0));
  }
  private ... divide(...) {
} }
```

under what circumstance can you call divide.

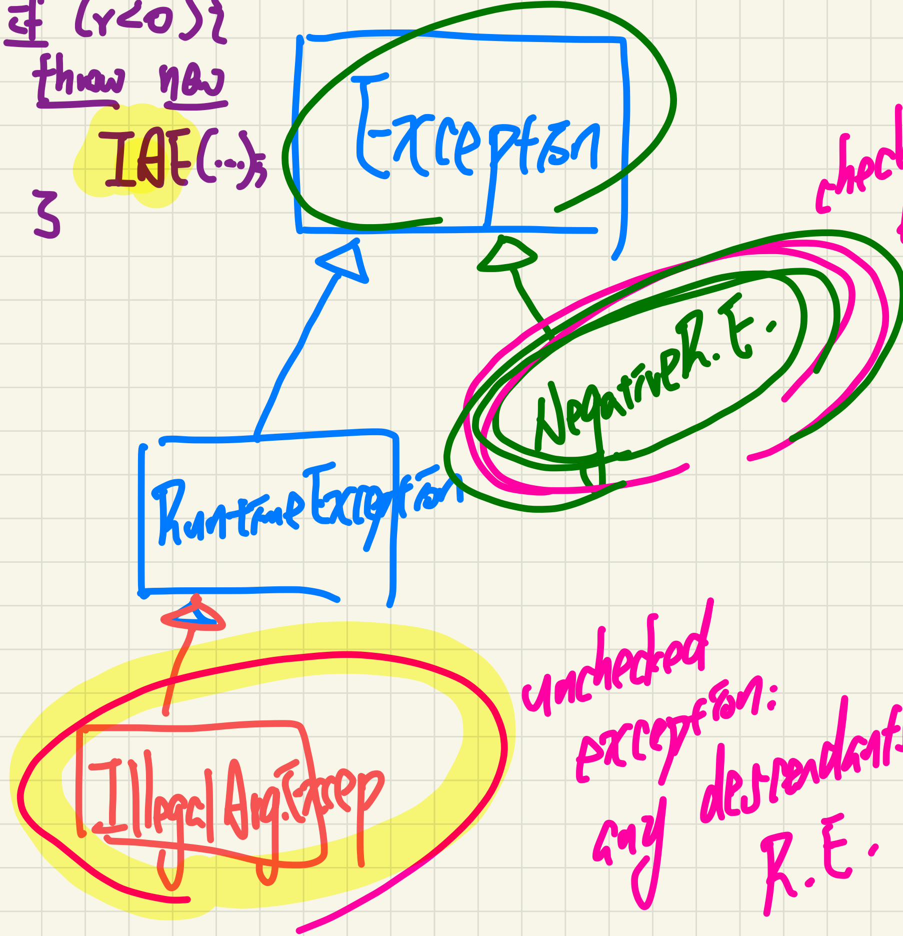
Option 2:

```
/** @return
 * @param x
 * @param y
private
double divide(double x, double y) {
  return x / y;
}
```

assume: $y != 0$.

@precondition $y != 0$

```
void m(int r) {
  if (r<0) {
    throw new IAE(...);
  }
}
```




checked exception: any direct child class of Exception.
(subject to: catch or specify Req.

unchecked exception: any descendants of R.E.

```
void m(int r) throws NRE {
  if (r < 0) {
    throw new NRE(...);
  }
}
```

## Exercise.

two counters

$c_1$: min $-1$, max $3$ — init: $0$

inc

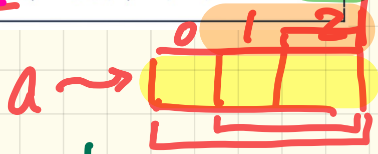
$c_2$: min $4$, max $7$ — init: $4$

dec

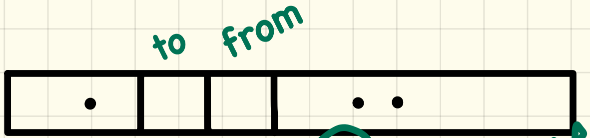
# Correctness Proofs: Ideas

allPosH (a, 0, 2)
↳ allPosH(a, 1, 2)
↳ allPosH(a, 2, 2)
↳ allPosH(a, 3, 2)

```
boolean allPositive(int[] a) { return allPosH (a, 0, a.length - 1); }
boolean allPosH (int[] a, int from, int to) {
  if (from > to) { return true; }
  else if(from == to) { return a[from] > 0; }
  else { return a[from] > 0 && allPosH (a, from + 1, to); } }
```



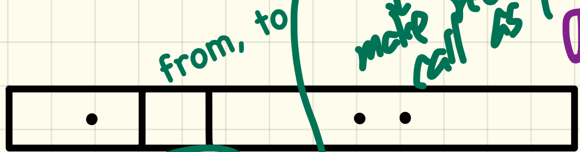

**Base Case:**
**Empty Array**




make recursive call as the I.H.

① Link to the code (line #'s)

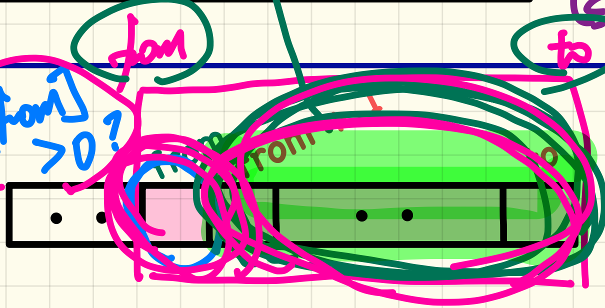
**Base Case:**
**Array of Size 1**




② Argue.

**Recursive Case:**
**Array of size > 1**

a[from] > 0 ?




a[from+1]
a[from+2]
⋮
a[to]

# Correctness Proofs

a → from … a.length - 1 to

```
1  boolean allPositive(int[] a) { return allPosH (a, 0, a.length - 1); }
2  boolean allPosH (int[] a, int from, int to) {
3    if (from > to) { return true; }
4    else if (from == to) { return a[from] > 0; }
5    else { return a[from] > 0 && allPosH (a, from + 1, to); } }
```

I.H. true if a[from+1], a[from+2], .. a[to] are all pos.

- Via mathematical induction, prove that `allPosH` is correct:

  **Base Cases**

  base cases — empty case

  - In an empty array, there is no non-positive number ∴ result is ***true***. **[L3]**
  - In an array of size 1, the only one elements determines the result. **[L4]**

  1-element case

  **Inductive Cases**

  - **Inductive Hypothesis**: `allPosH(a, from + 1, to)` returns ***true*** if a[from + 1], a[from + 2], . . . , a[to] are all positive; ***false*** otherwise.
  - `allPosH(a, from, to)` should return ***true*** if: **1)** a[from] is positive; and **2)** a[from + 1], a[from + 2], . . . , a[to] are all positive.
  - By *I.H.*, result is $a[\mathit{from}] > 0 \land$ `allPosH(a, from + 1, to)`. **[L5]**

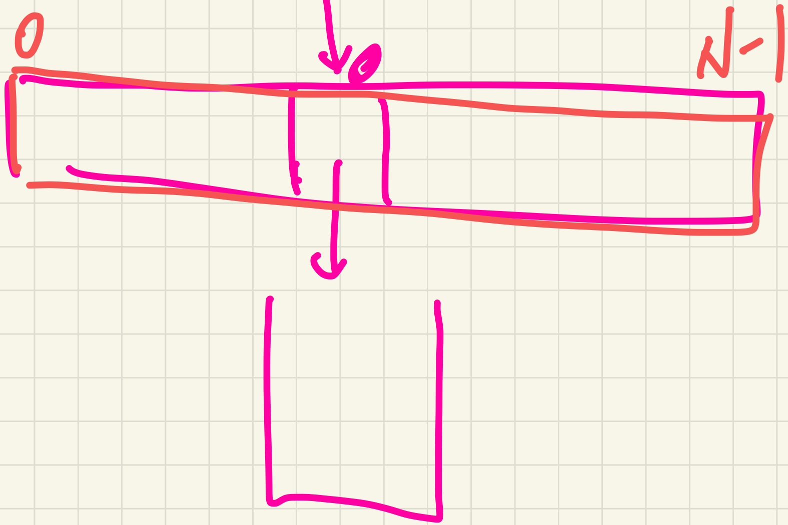
- `allPositive(a)` is correct by invoking `allPosH(a, 0, a.length - 1)`, examining the entire array. **[L1]**

K (int, strings, object)

~~K.hashCode()~~ → design % N → prime number

11

13.



180 min.

→ 50 multiple choice

↳ < 2 minutes.

5 written questions.

↳ 10 min.

1st round: complete all q's that you can answer

90 min

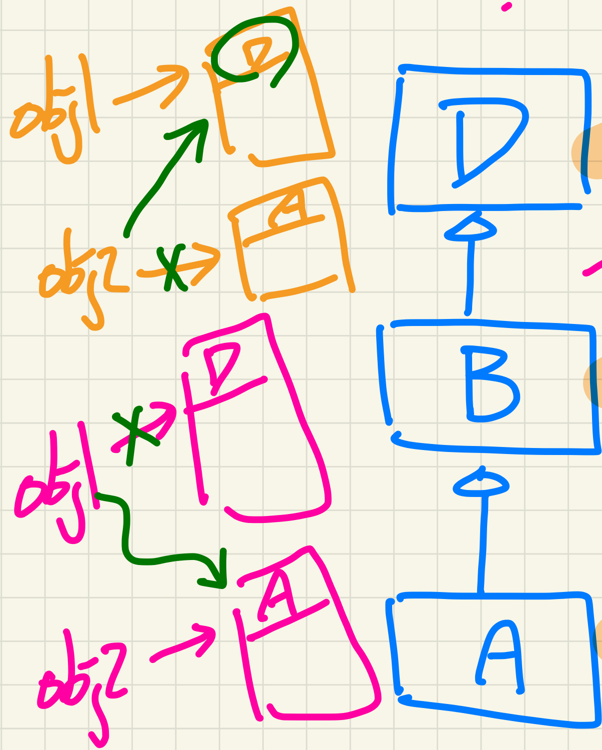
$hc(k) = k \% 11$

$hc(k1) == hc(k2)$

$\underbrace{hc(23)}_{1} == \underbrace{hc(34)}_{1}$ ✓ collision

$hc(23) == hc(\underset{\sim}{23})$ → same value for the same input.

$23 \% 11$ $\qquad$ $23 \% 11$




dm print("D.dm");

dm print("B.dm");

bm print("B.bm");

dm print("A.dm");

am print("A.am");

D obj1 = new D();

obj1.bm(); X

ST: D

obj1.dm(); -> "D.dm"

D obj2 = new A();

obj2.dm(); -> "A.dm"

Scenario 1

obj1 = obj2;

obj1.dm();

"A.dm"

Scenario 2

obj2 = obj1;

obj2.dm();

"D.dm"

```
class App {
  ... main(...) {
    C oc = new C();
    D obj1 = new A();
    oc.add(obj1);  oc.add(obj1);
    B obj2 = new A();
    oc.add(obj2);  oc.add(obj2);
    oc.add(obj2);
    oc.add((A) obj2);
```

ST: B  ST: B

b = obj2   a = obj2
B   B    A   B

```
class C {
  B[] array;
  int noi; /* # of items */

  void add(D d) {
    a[noi] = d;
  }
```

X  X

ST: B  ST: D

b = obj2

```
  void add(B b) {
    a[noi] = b;
  }
  void add(A a) {
    noi++;
  }
    a[noi] = a;
}
```

ST: B  ST: A