

## Lecture 3

### Part C

*Loops -  
Stay Condition vs. Exit Condition*

## Stay Condition vs. Exit Condition

When does the loop **exit** (i.e., stop repeating Action 1)?

```
while (p && q) { /* Action 1 */ }
```

↳ repeat Action 1 as long as  $p \&\& q$  evaluates true.

↳ exit from loop:  $! (p \&\& q) \equiv !p \parallel !q$

When does the loop exit (i.e., stop repeating Action 2)?

```
while (p || q) { /* Action 2 */ }
```

↳ repeat Action 2 as long as  $p \parallel q$  evaluates true.

↳ exit from loop:  $! (p \parallel q) \equiv !p \&\& !q$

## Stay Condition vs. Exit Condition: Exercise

infinite loop

Consider the following loop:

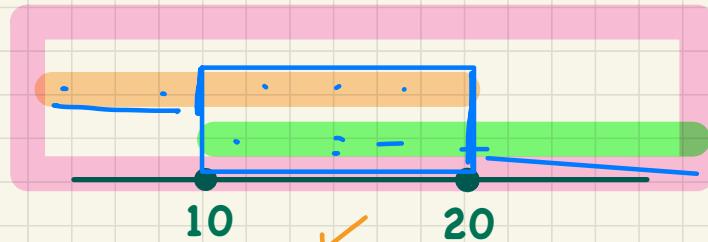
```
int x = input.nextInt();
while (10 <= x || x <= 20) {
    /* body of while loop */
}
```

True.

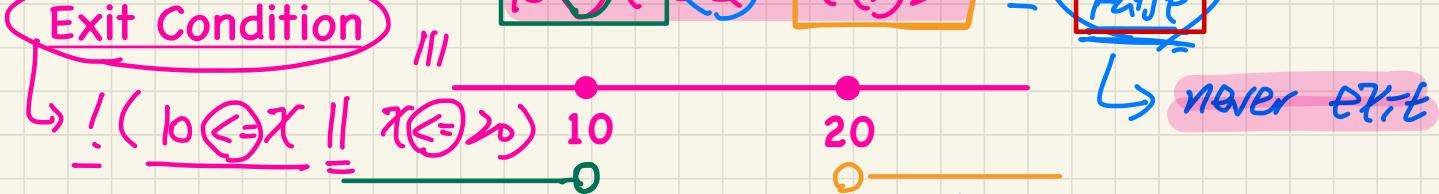
always evaluates to true  
⇒ never exit

- It compiles, but has a logical error. Why?

Stay Condition



Exit Condition

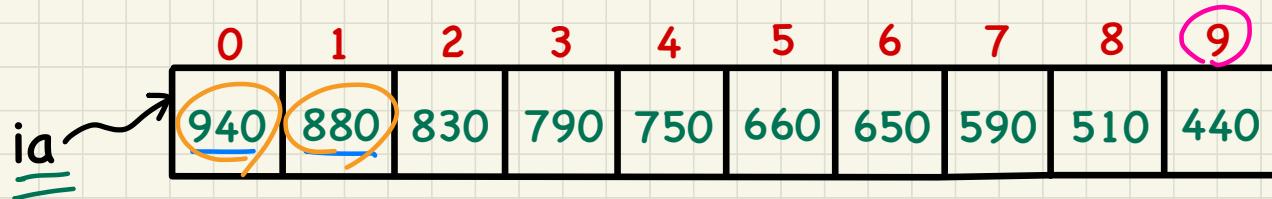


# Lecture 3

## Part D

*Loops -  
Arrays: Declaration and Initialization*

# Initializing an Array of Integers (1)



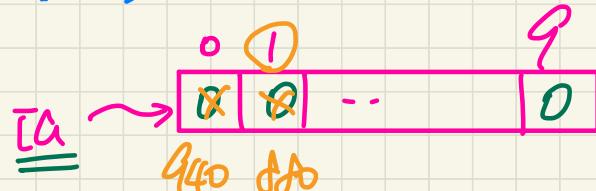
## Approach 1: Initializer

`int [] ia = {940, 880, 830, 790, 750, 660, 650, 590, 510, 440};`

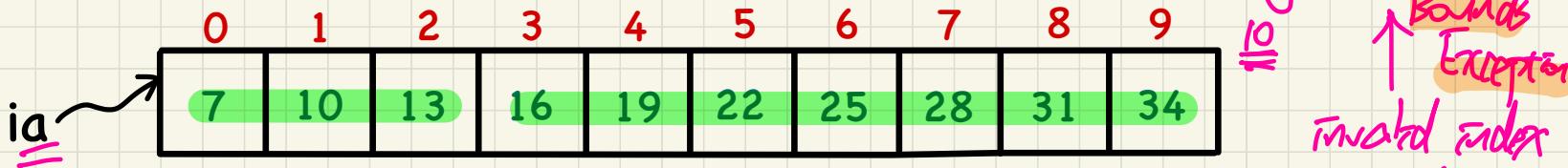
## Approach 2: Discrete Assignments

`int [] ia = new int [10];`

`ia[0] = 940;    ia[1] = 880; ...`



## Initializing an Array of Integers (2)

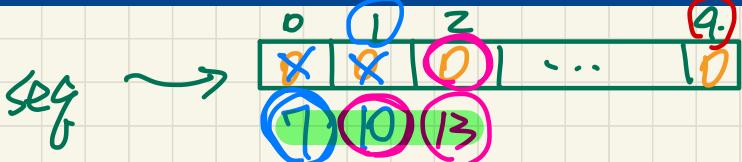


### Approach 3: Patternizing Stored Values

```

int[] seq = new int[10];
seq[0] = 7; i
for(int i = 0; i < seq.length; i++) {
    seq[i] = seq[i - 1] + 3;
}
    
```

Annotations: 'seq' is circled in green. 'i' is circled in blue. '0' is circled in blue. '10' is circled in red. 'seq.length' is circled in red. 'i++' is circled in blue. 'seq[i - 1]' is circled in yellow. 'i' is circled in blue below the loop. '0' is circled in blue below the assignment. '1' is circled in blue below the increment. '2' is circled in pink below the increment. '3' is circled in pink below the increment. '4' is circled in pink below the increment. '5' is circled in pink below the increment. '6' is circled in pink below the increment. '7' is circled in pink below the increment. '8' is circled in pink below the increment. '9' is circled in pink below the increment. '10' is circled in pink below the increment.



Is it?

4th. →  
Xth. →

True.

i	i < seq.length	i - 1	seq[i - 1]
0	True	-1	seq[-1]
1	True	0	seq[0]
2	True	1	seq[1]
3	True	2	seq[2]
4	True	3	seq[3]
5	True	4	seq[4]
6	True	5	seq[5]
7	True	6	seq[6]
8	True	7	seq[7]
9	True	8	seq[8]
10	False	9	seq[9]

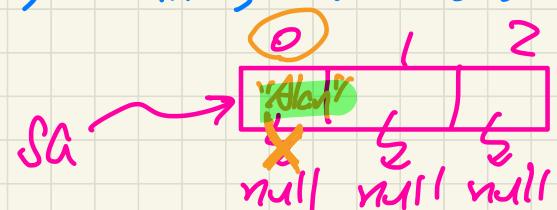
# Initializing an Array of Strings



## Approach 1: Initializer

`String[] sa = {"Alan", "Mark", "Tom"};`

## Approach 2: Discrete Assignments



`String[] sa = new String[3];`

`sa[0] = "Alan";`

## for-Loops vs. while-Loops: Iterating through Arrays

```
int[] a = new int[100];
for(int i = 0; i < a.length; i++) {
    /* Actions to repeat. */
}
```

min index of array

Stay condition    Exit:  $i < a.length$

```
int[] a = new int[100];
int i = 0;
while(i < a.length) {
    /* Actions to repeat. */
    i++;
}
```

$i < a.length$

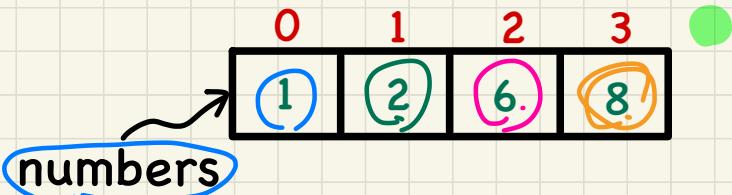
first  
increased  
index

## Lecture 3

### Part E

*Loops and Arrays -  
Computational Problems*

# Computational Problem: Average



## Test Inputs:

int[] numbers = {1, 2, 6, 8};  
int[] numbers = {};

4.25

**Problem:** Given an array `numbers` of integers, how do you print its average?

e.g., Given array {1, 2, 6, 8}, print 4.25.

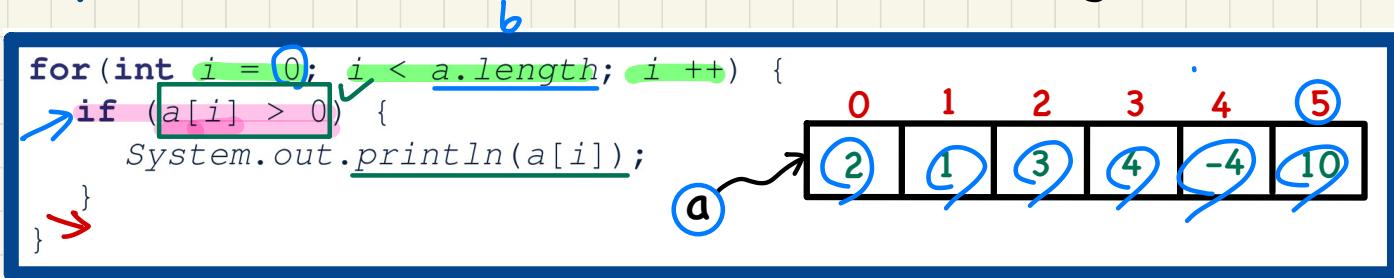
```
int sum = 0; 0 < 0 (F) 4
for(int i = 0; i < numbers.length; i++) {
    sum += numbers[i];
}
double average = (double) sum / numbers.length;
System.out.println("Average is " + average);
```

Annotations on the code:

- Line 1: `int sum = 0;` has a pink circle around `sum`.
- Line 2: `0 < 0 (F) 4` is written above the loop.
- Line 3: `for(int i = 0;` has a pink circle around `i`.
- Line 4: `i < numbers.length;` has a pink circle around `numbers`.
- Line 5: `sum += numbers[i];` has a pink circle around `sum`.
- Line 6: `0.0/0 → division by zero exception.` is written next to the division operation.
- Line 7: `4` is written below the division operation.
- Line 8: `double average = (double) sum / numbers.length;` has a pink circle around `sum`.
- Line 9: `System.out.println("Average is " + average);` has a pink circle around `average`.

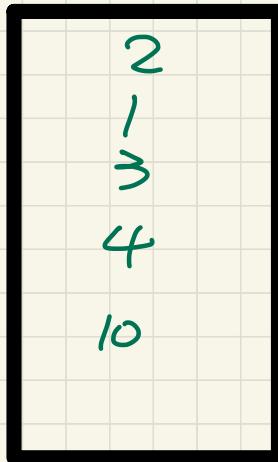
i	Sum
0	1
1	3
2	9
3	17
4	exit.

# Computational Problem: Conditional Printing



i	$i < a.length$	$a[i]$	$a[i] > 0$
0	True.	2	T
1		1	T
2		3	T
3		4	T
4		-4	F
5		10	T

Console

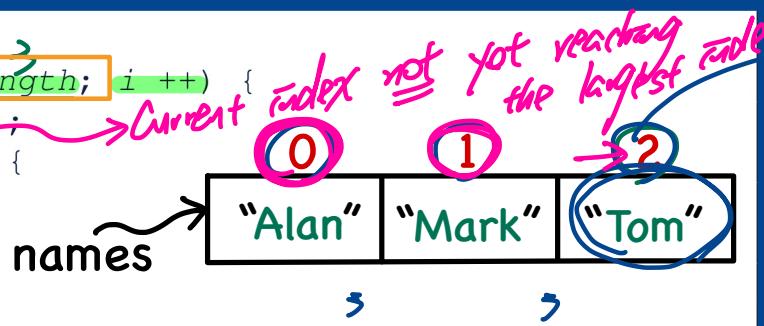


At the end of the 6th iteration  
it's false.

# Computational Problem: Printing Comma-Separated Lists

```
System.out.print("Names: ")  
for(int i = 0; i < names.length; i++) {  
    System.out.print(names[i]);  
    if (i < names.length - 1) {  
        System.out.print(", ");  
    }  
}  
System.out.println(".");
```

Current index  $\neq$  yet reaching the largest end X  
names.length - 1  
largest valid index  
 $\leq$  names.length - 1



i	i < names.length	names[i]	i < names.length - 1
0	True.	'Alan'	T
1		"Mark"	T
2		"Tom"	F
3			F

Console

Names: Alan, Mark, Tom.



## Computational Problem: Printing Backwards

**Problem:** Given an array `numbers` of integers, how do you print its contents backwards?

e.g., Given array {1, 2, 3, 4}, print 4 3 2 1.

**Solution 1:** Change bounds and updates of loop counter.

```
for(int i = numbers.length - 1; i >= 0; i--) {
    System.out.println(numbers[i]);
}
```

$i \geq 0$

loop

$i < ns.length$

T

T

T

T

T

T

T

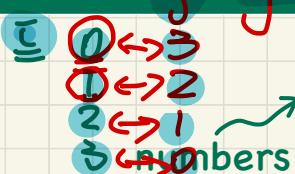
**Solution 2:** Change indexing.

```
for(int i = 0; i < numbers.length; i++) {
    System.out.println(numbers[names.length - i - 1]);
}
```

$i < 0$

T

T

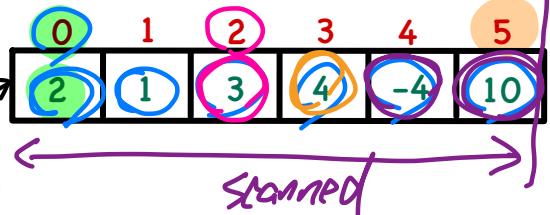


$(ns.length - 1)!$

# Computational Problem: Finding Maximum

```
1 int max = a[0];  
2 for(int i = 0; i < a.length; i++) {  
3     if a[i] > max) { max = a[i]; }  
4 }  
5 System.out.println("Maximum is " + max);
```

*→ Current element > max so far.*



i	a[i]	a[i] > max	update max?	max
0	-	-	-	2
1	2	2>2 false	N	2
2	1	1>2 false	N	2
3	3	3>2 true	Y	3
4	4	4>3 true	Y	4
5	-4	-4>4 false	N	4
6	10	10>4 true	Y	10

Console

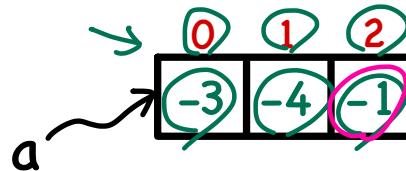
Max is 10

# Computational Problem: Finding Maximum

Q: What if we change the initialization in L1 to `int max = 0`?

Exercise 1

```
1 int max = a[0]; ①
2 for(int i = 0; i < a.length; i++) {
3     if (a[i] > max) { max = a[i]; } ②
4 }
5 System.out.println("Maximum is " + max);
```



i	<code>i &lt; a.length</code>	<code>a[i]</code>	<code>a[i] &gt; max</code>
0	T	-3	-3 > 0 F
1	T	-4	-4 > 0 F
2	T	-1	-1 > 0 F

Console

Max is 0

# Computational Problem: Finding Maximum

$$\leq > \leq \equiv \text{F}$$

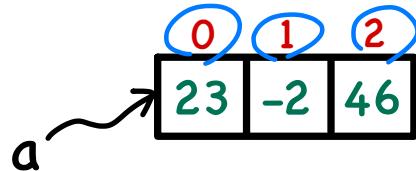
Q: What if we change the initialization in L2 to `int i = 1`?

Exercise 2

```

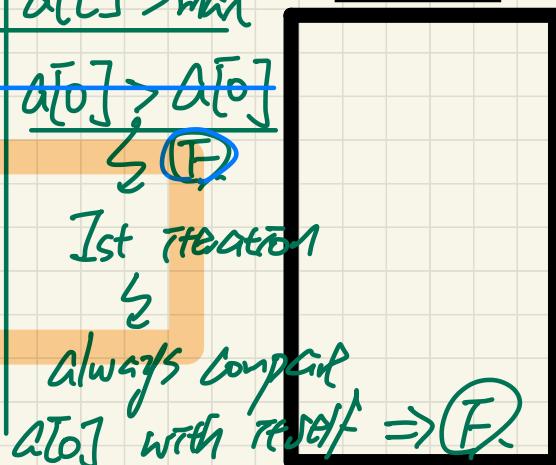
1 int max = a[0]; z3
2 for(int i = 0; i < a.length; i++) {
3     if (a[i] > max) { max = a[i]; } X
4 }
5 System.out.println("Maximum is " + max);

```

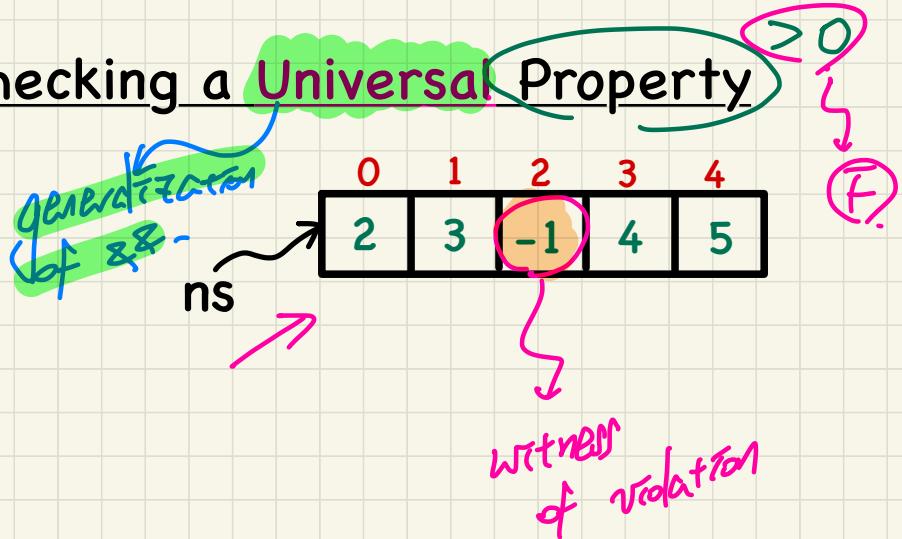


i	<code>i &lt; a.length</code>	<code>a[i]</code>	<code>a[i] &gt; max</code>
0	True	23	<u><code>a[0] &gt; a[0]</code></u> ↙ <b>F</b>
1	True	-2	<i>Ist iteration</i> ↙
2	True	46	<i>Always compare <code>a[0]</code> with result</i> $\Rightarrow$ <b>F</b>
3	False		

Console



# Computational Problem: Checking a Universal Property



boolean allPositive

$ns[0] > 0$
$ns[1] > 0$
$ns[2] > 0$
$ns[3] > 0$
$ns[4] > 0$

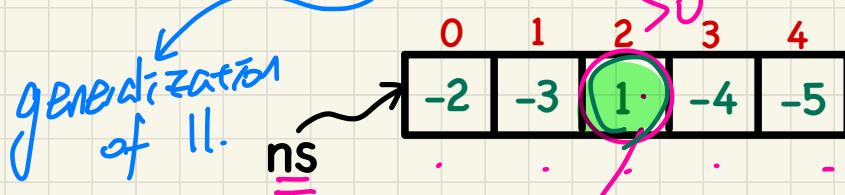
Zero of &&

False && b = False

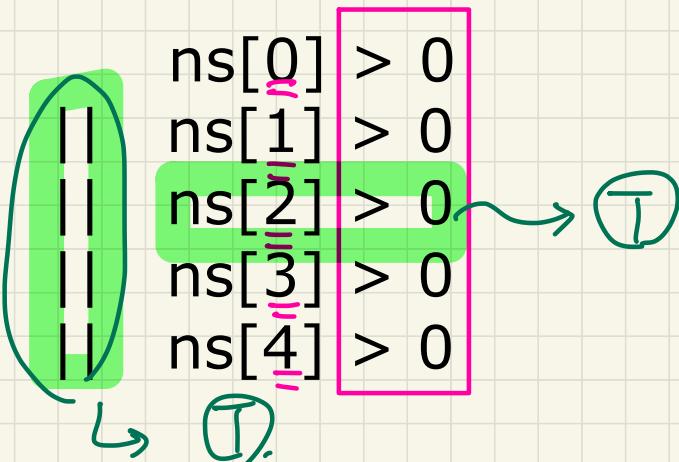
Identity of &&

True && b = b.

## Computational Problem: Checking an Existential Property



**boolean** atLeastOnePositive



True  $\| b \equiv \text{True}$

False  $\| b \equiv b$

# Computational Problem: Are All Numbers Positive?

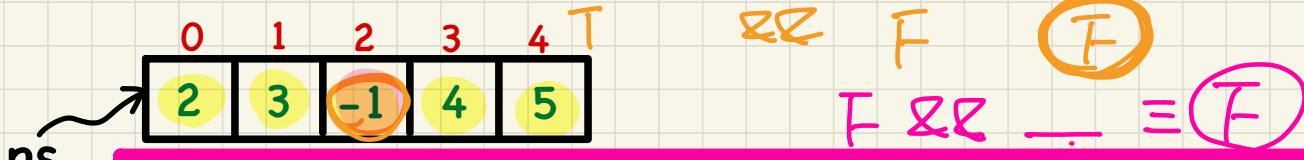
what if  $\text{E. } X$   
↳

```
1 int [] ns = {2, 3, -1, 4, 5};  
2 boolean soFarOnlyPosNums = true;  
3 int i = 0;  
4 while (i < ns.length) {  
5     soFarOnlyPosNums = soFarOnlyPosNums && (ns[i] > 0);  
6     i = i + 1;  
7 }
```

Identity of  $\&\&$

Version 1

$ns[i] > 0$   
check if  
meaning less  
(Zero  $\cup$  f  $\&\&$ )



i	soFarOnlyPosNums	$i < ns.length$	stay?	ns[i]	$ns[i] > 0$
0	true (F)	true	YES	2	true
1	true	true	YES	3	true
2	true	true	YES	-1	false
3	false	true	YES	4	true
4	false	true	YES	5	true
5	false	false	No	-	-

# Computational Problem: At Least One Number Positive?

```
1 int [] ns = {-2, -3, 1, -4, -5};  
2 boolean seenSomePosNum = false;  
3 int i = 0;  
4 while (i < ns.length) {  
5     seenSomePosNum = seenSomePosNum || (ns[i] > 0);  
6     i = i + 1;  
7 }
```

identity of  
||

Version 1

T || E  
" "  
= T

E || F ≡ F  
F || T = T

True || — ≡ True

ns



F || F ≡ F  
F || T = T

i	seenSomePosNum	i < ns.length	stay?	ns[i]	ns[i] > 0
0	false	true	YES	-2	false
1	false	true	YES	-3	false
2	false	true	YES	1	true
3	true	true	YES	-4	false
4	true	true	YES	-5	false
5	true	false	NO	-	-

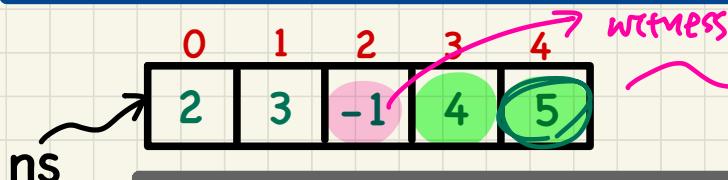
T

# Computational Problem: Are All Numbers Positive?

```
1 int [] ns = {2, 3, -1, 4, 5};  
2 boolean soFarOnlyPosNums = true;  
3 int i = 0;  
4 while (i < ns.length) {  
5     soFarOnlyPosNums = ns[i] > 0; /* wrong */  
6     i = i + 1;  
7 }
```

Version 2

the final value of corresponds to  
the last check



expected: univ. property : F

i	soFarOnlyPosNums	$i < ns.length$	stay?	$ns[i]$	$ns[i] > 0$
0	true	true	YES	2	true
1	true	true	YES	3	true
2	true	true	YES	-1	false
3	false	true	YES	4	true
4	true	true	YES	5	true
5	true	false	No	-	-

# Computational Problem: At Least One Number Positive?

```
1 int [] ns = {-2, -3, 1, -4, -5};  
2 boolean seenSomePosNum = false;  
3 int i = 0;  
4 while (i < ns.length) {  
5     seenSomePosNum = ns[i] > 0; /* wrong */  
6     i = i + 1;  
7 }
```

## Version 2

final result corresponds to  $ns[4] > 0$ .



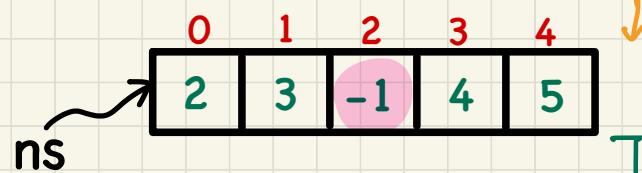
i	seenSomePosNum	$i < ns.length$	stay?	$ns[i]$	$ns[i] > 0$
0	false	true	YES	-2	false
1	false	true	YES	-3	false
2	false	true	YES	1	true
3	true	true	YES	-4	false
4	false	true	YES	-5	false
5	false	false	NO	-	-

# Computational Problem: Are All Numbers Positive?

```
1 int [] ns = {2, 3, -1, 4, 5};  
2 boolean soFarOnlyPosNums = true;  
3 int i = 0; ✓ F - exit  
4 while (soFarOnlyPosNums && i < ns.length) {  
5     soFarOnlyPosNums = soFarOnlyPosNums && ns[i] > 0;  
6     i = i + 1;  
7 }
```

Version 3

unnecessary



exit: ! ( soFarN && i < ns.length )  
T && F = F  
have just seen a number  $\leq 0$

i	soFarOnlyPosNums	$i < ns.length$	stay?	$ns[i]$	$ns[i] > 0$
0	true	true	YES	2	true
1	true	true	YES	3	true
2	true	true	YES	-1	false
3	false	true	No	-	-

# Computational Problem: At Least One Number Positive?

```
1 int [] ns = {-2, -3, 1, -4, -5};  
2 boolean seenSomePosNum = false;  
3 int i = 0;  
4 while (!seenSomePosNum && i < ns.length) {  
5     seenSomePosNum = seenSomePosNum || ns[i] > 0;  
6     i = i + 1;  
7 }
```

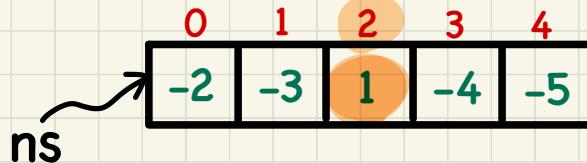
$\neg \text{seenSomePosNum} \wedge i < \text{ns.length} \equiv \text{F} \wedge \text{T} \equiv \text{F}$

$\neg \text{seenSomePosNum} \equiv \text{F}$

$\neg \text{seenSomePosNum} \wedge \text{ns}[i] > 0 \equiv \text{F} \wedge \text{T} \equiv \text{F}$

$\text{seenSomePosNum} \equiv \text{T}$

Version 3



unnecessary exit:  $\neg (\neg \text{seenSomePosNum} \wedge i < \text{ns.length})$

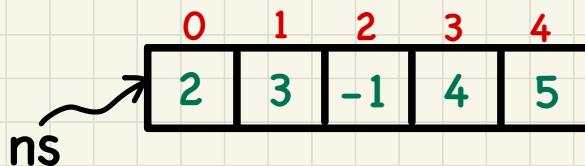
$\downarrow$  loop already exit.  $\equiv \text{seenSomePosNum} \equiv i \geq \text{ns.length}$

i	seenSomePosNum	$i < \text{ns.length}$	stay?	$\text{ns}[i]$	$\text{ns}[i] > 0$
0	false	true	YES	-2	false
1	false	true	YES	-3	false
2	false	true	YES	1	true
3	true	true	No	-	-

# Computational Problem: Are All Numbers Positive?

```
1 int [] ns = {2, 3, -1, 4, 5};  
2 boolean soFarOnlyPosNums = true;  
3 int i = 0; F  
4 while (soFarOnlyPosNums && i < ns.length) {  
5     soFarOnlyPosNums = ns[i] > 0;  
6     i = i + 1;  
7 }
```

Version 4

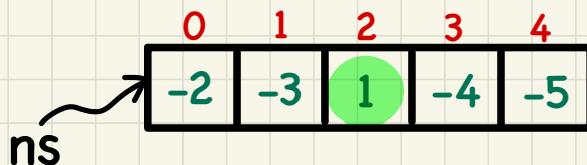


$i$	$soFarOnlyPosNums$	$i < ns.length$	stay?	$ns[i]$	$ns[i] > 0$
0	true	true	YES	2	true
1	true	true	YES	3	true
2	true	true	YES	-1	false
3	false	true	No	-	-

# Computational Problem: At Least One Number Positive?

```
1 int [] ns = {-2, -3, 1, -4, -5};  
2 boolean seenSomePosNum = false;  
3 int i = 0; !I = E  
4 while (!seenSomePosNum && i < ns.length) {  
5     seenSomePosNum = ns[i] > 0;  
6     i = i + 1;  
7 }
```

Version 4



<i>i</i>	<i>seenSomePosNum</i>	<i>i &lt; ns.length</i>	stay?	<i>ns[i]</i>	<i>ns[i] &gt; 0</i>
0	<i>false</i>	<i>true</i>	YES	-2	<i>false</i>
1	<i>false</i>	<i>true</i>	YES	-3	<i>false</i>
2	<i>false</i>	<i>true</i>	YES	1	<i>true</i>
3	<i>true</i>	<i>true</i>	No	-	-

# Computational Problem: Are All Numbers Positive?

Four possible solutions (`soFarOnlyPosNums` initialized `true`): [summary](#)

1. Scan the entire array and accumulate the result.

```
for (int i = 0; i < ns.length; i++) {  
    soFarOnlyPosNums = soFarOnlyPosNums && ns[i] > 0; }
```

2. Scan the entire array but the result is not accumulative.

```
for (int i = 0; i < ns.length; i++) {  
    soFarOnlyPosNums = ns[i] > 0; } /* Not working. Why? */
```

3. The result is accumulative until the early exit point.

```
for (int i = 0; soFarOnlyPosNums && i < ns.length; i++) {  
    soFarOnlyPosNums = soFarOnlyPosNums && ns[i] > 0; }
```

4. The result is not accumulative until the early exit point.

```
for (int i = 0; soFarOnlyPosNums && i < ns.length; i++) {  
    soFarOnlyPosNums = ns[i] > 0; }
```

# Computational Problem: At Least One Number Positive?

Four possible solutions (seenSomePosNum initialized *false*):

summary

1. Scan the entire array and accumulate the result.

```
for (int i = 0; i < ns.length; i++) {  
    seenSomePosNum = seenSomePosNum || ns[i] > 0; }
```

2. Scan the entire array but the result is **not** accumulative.

```
for (int i = 0; i < ns.length; i++) {  
    seenSomePosNum = ns[i] > 0; } /* Not working. Why? */
```

3. The result is accumulative until the early exit point.

```
for (int i = 0; !seenSomePosNum && i < ns.length; i++) {  
    seenSomePosNum = seenSomePosNum || ns[i] > 0; }
```

4. The result is **not** accumulative until the early exit point.

```
for (int i = 0; !seenSomePosNum && i < ns.length; i++) {  
    seenSomePosNum = ns[i] > 0; }
```