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) ≡ !p || !q

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

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

↳ repeat Action 2 as long as p || q evaluates true

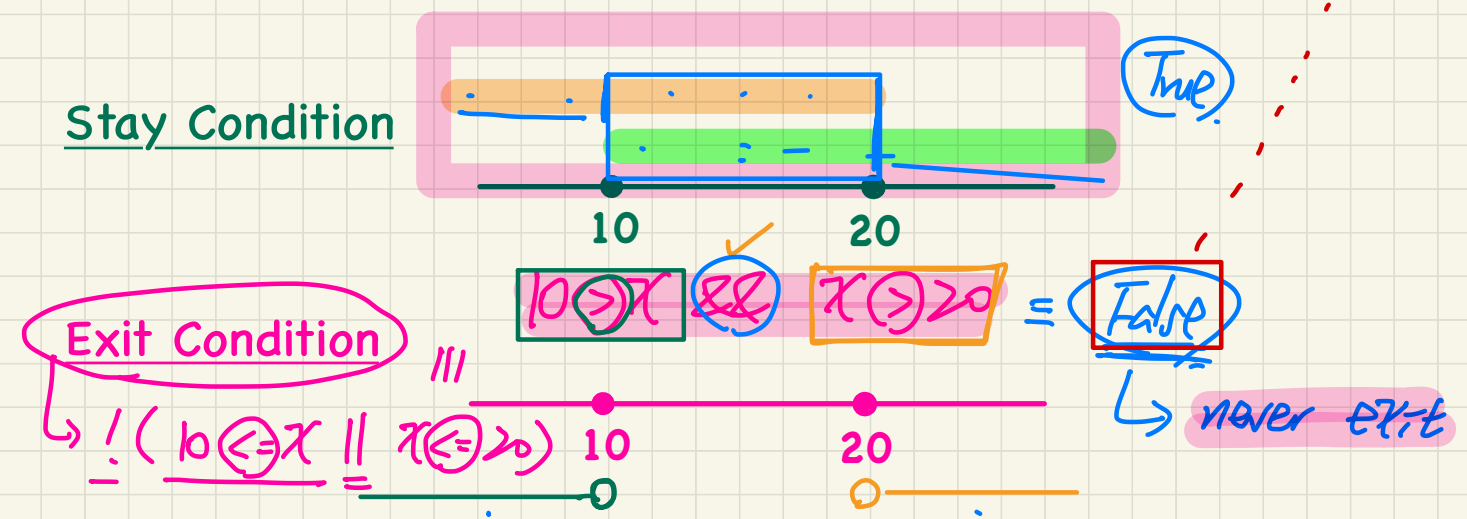
↳ exit from loop: !(p || q) ≡ !p && !q

Stay Condition vs. Exit Condition: Exercise

Consider the following loop:

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

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

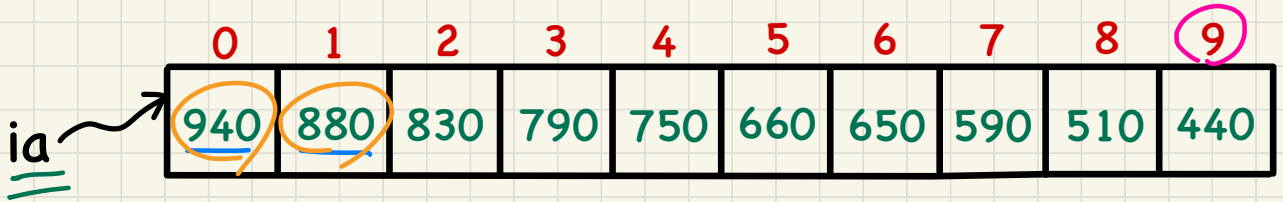


Lecture 3

Part D

Loops -
Arrays: Declaration and Initialization

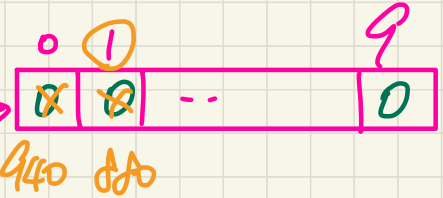
Initializing an Array of Integers (1)



Approach 1: Initializer

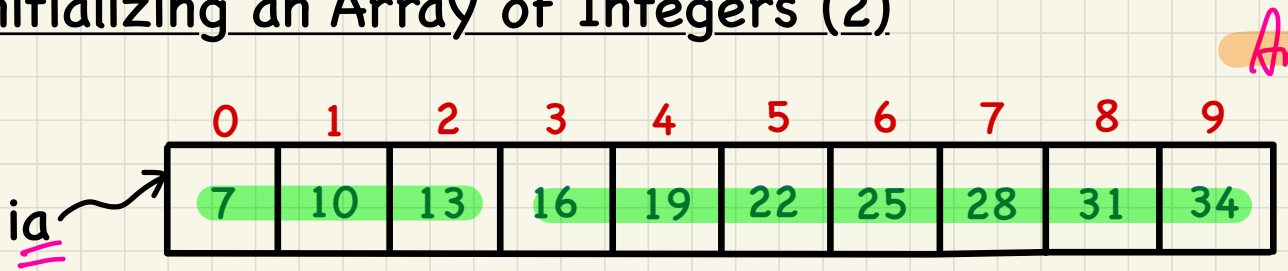
```
int[] ia = {940, 880, 830, ..., 440};
```

Approach 2: Discrete Assignments




```
int[] ia = new int[10];
ia[0] = 940;  ia[1] = 880;  ...
```

Initializing an Array of Integers (2)

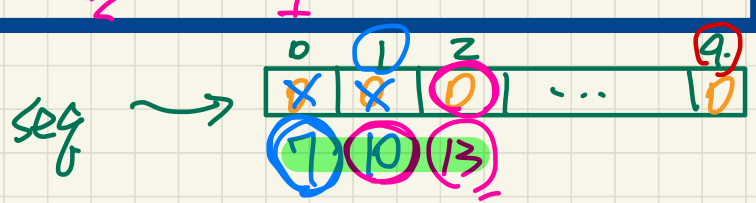


ArrayIndexOutOfBoundsException

Invalid index

Approach 3: Patterning Stored Values

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



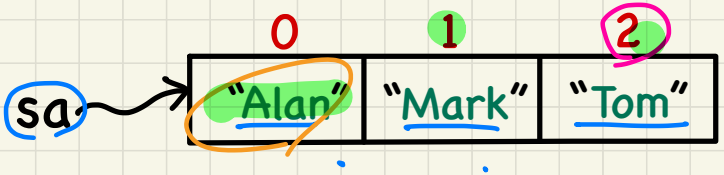
i	i < seq.length	i - 1	seq[i - 1]
0		-1	seq[-1]
1		0	seq[0] 7
2		1	seq[1] 10
3			
4			
5			
6			
7			
8			
9			
10	F		

True.

1st it.

4th.

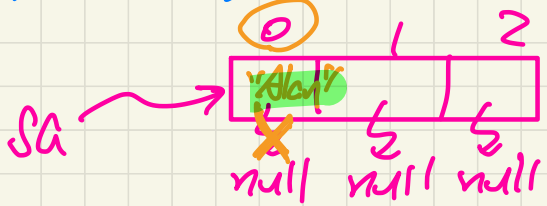
Initializing an Array of Strings



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)

i >= a.length

first invalid index

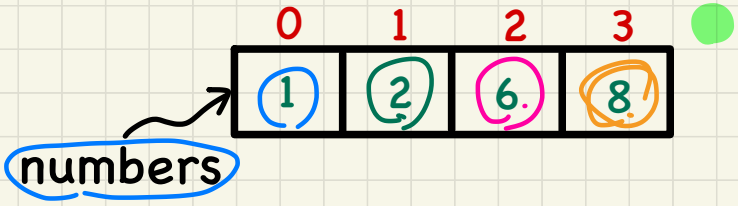
```
int[] a = new int[100];
int i = 0;
while(i < a.length) {
    /* Actions to repeat. */
    i ++;
}
```

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

0 < 0 (F) 4

0.0/0 → division by zero exception.

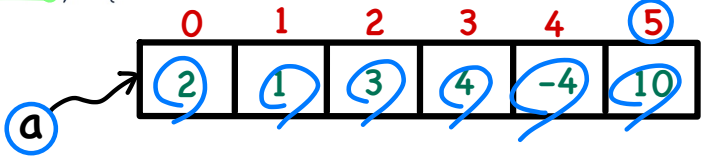
4

i	sum
0	1
1	3
2	9
3	17
4	

exit.

Computational Problem: Conditional Printing

```
for(int i = 0; i < a.length; i ++) {
    if (a[i] > 0) {
        System.out.println(a[i]);
    }
}
```



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
6	F.		

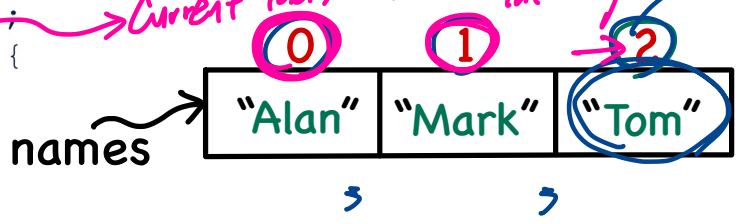
at end the 6th it.

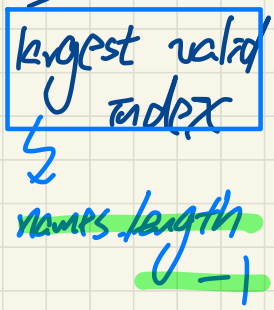
Console

```
2
1
3
4
10
```

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(".");
```





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?

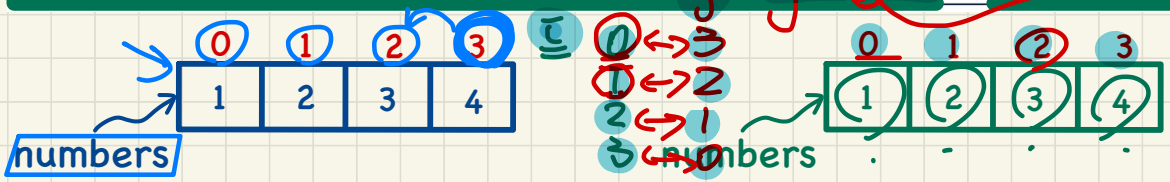
e.g., Given array $\{1, 2, 3, 4\}$, print `4 3 2 1`.

Solution 1: Change bounds and updates of loop counter.

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

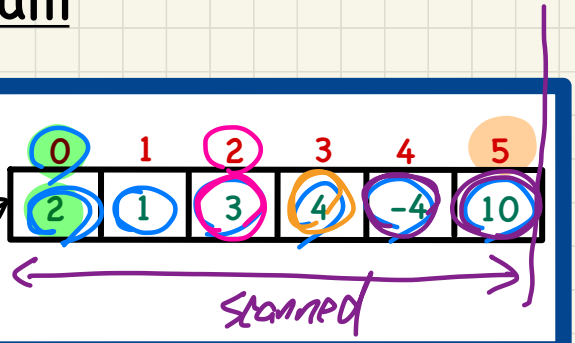
Solution 2: Change indexing.

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



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

6

Console

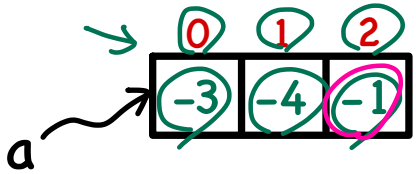
Max is 10

Computational Problem: Finding Maximum

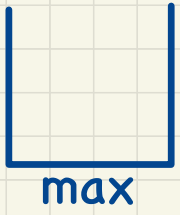
Q: What if we change the initialization in **L1** to `int max = 0`?

Exercise 1

```
1  int max = a[0];
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);
```



i	i < a.length	a[i]	a[i] > max
0	T	-3	-3 > 0 F
1	T	-4	-4 > 0 F
2	T	-1	-1 > 0 F



Console

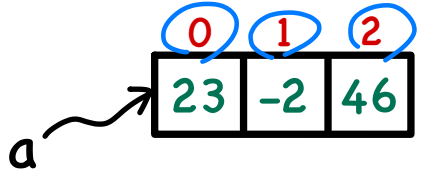
Max is 0

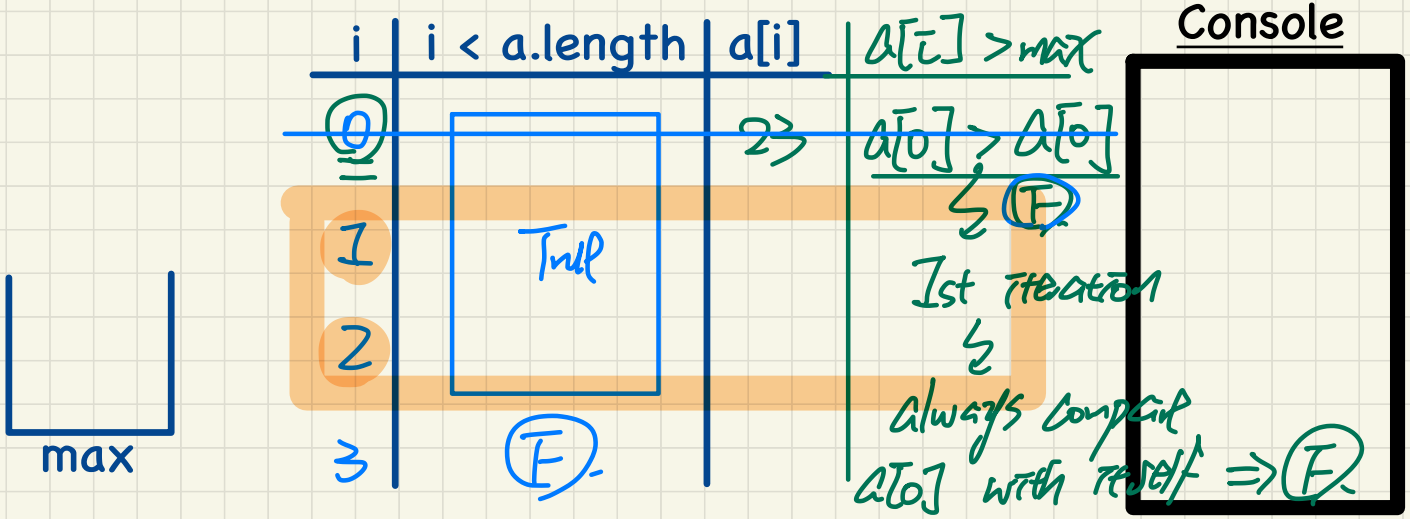
Computational Problem: Finding Maximum

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

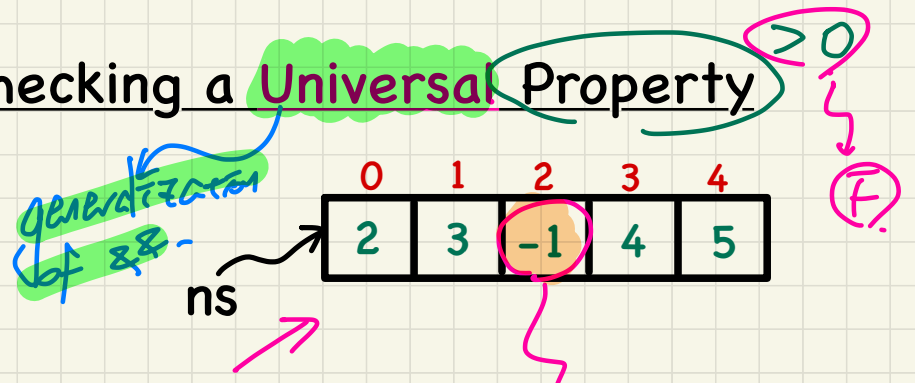
Exercise 2

```
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);
```

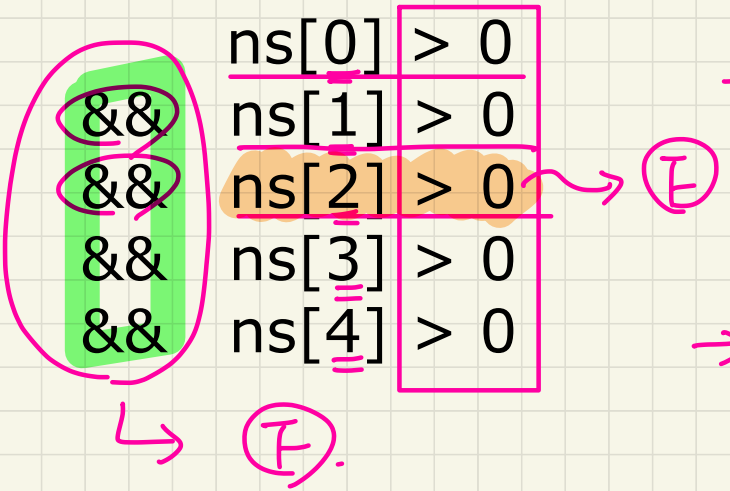




Computational Problem: Checking a Universal Property



boolean *allPositive*



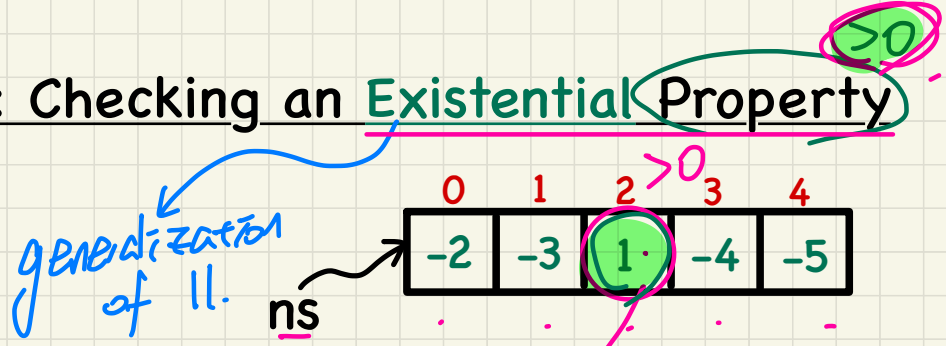
→ Zero of &&

False && b ≡ False

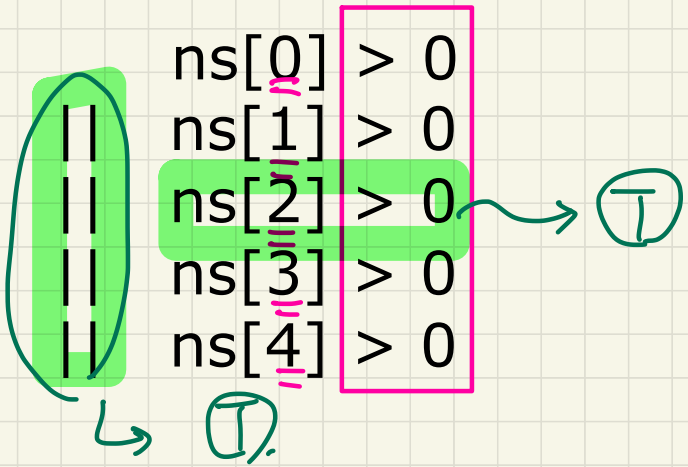
→ Identity of &&

True && b ≡ b.

Computational Problem: Checking an Existential Property



boolean *atLeastOnePositive*



Zero of ||

True || b ≡ True

Identity of ||

False || b ≡ b

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 = soFarOnlyPosNums && (ns[i] > 0);
6     i = i + 1;
7  }
```

Version 1

what if F? ✗

identity of &&

ns[i] > 0 check 可 meaningless (zero of &&).

T T && T && T && F F && — ≡ 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  }
```

Version 1

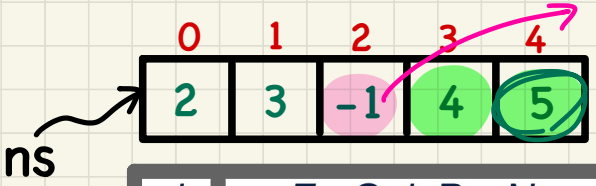
i	seenSomePosNum	i < ns.length	stay?	ns[i]	ns[i] > 0
0	false	true	YES	-2	false
1	false	true	YES	-3	false
2	false	true	YES	1	true
3	true	true	YES	-4	false
4	true	true	YES	-5	false
5	true	false	NO	–	–

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	–	–

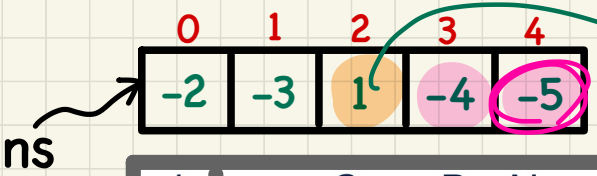
X

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.



witness ⇒ expected exist. check: ☹

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	–	–

X

Computational Problem: Are All Numbers Positive?

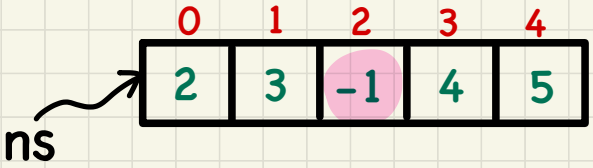
Version 3

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

F

F - exit

unnecessary



T && F = F

exit: !(stopn && i<ns.len.)

have just seen a number ≤ 0

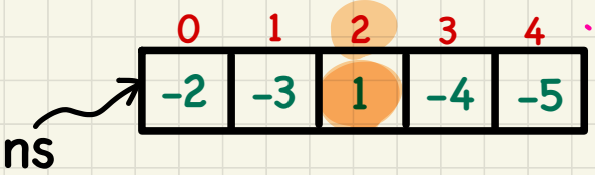
!stopn || i >= ns.len.

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  }
```

Version 3



unnecessary — loop already exit.

exit: !(!sspn && i<ns.len)

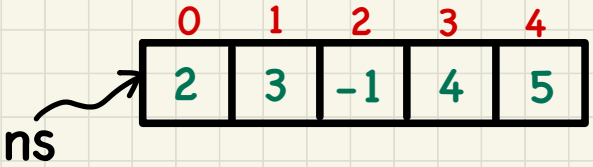
sspn || i>=ns.len

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	NO	–	–

Computational Problem: Are All Numbers Positive?

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

Version 4

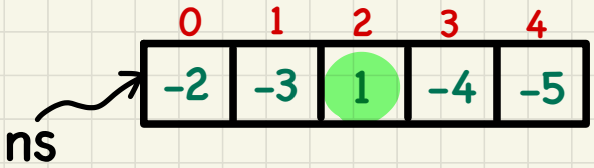


i	*soFarOnlyPosNums*	*i < ns.length*	stay?	*ns[i]*	*ns[i] > 0*
0	*true*	*true*	YES	2	*true*
1	*true*	*true*	YES	3	*true*
2	*true*	*true*	YES	-1	*false*
3	*false*	*true*	NO	–	–

Computational Problem: At Least One Number Positive?

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

Version 4



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*	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; }
```