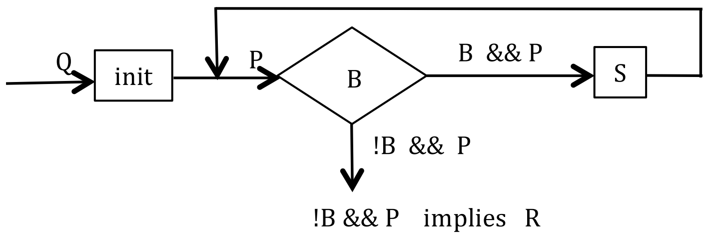
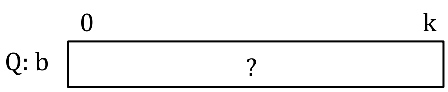
Answering the four loopy questions. Example 2

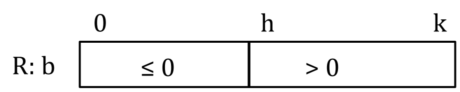
We give a second example of answering the four loopy questions, this time where the precondition, postcondition, and loop invariant are given as pictures. But we do it differently. We *develop* the initialization and loop using the four loopy questions. To help you remember the four loopy questions, we give the general flowchart for a loop with initialization: init; **while** (B) S .



In precondition Q below, the query "?" means that we know little about the values in array segment b[0..k]. The purpose of the algorithm we are going to write is to rearrange the values in b[0..k] so that postcondition R is true. Evidently, we are supposed to put all the non-positive values on the left and the positive ones the right.

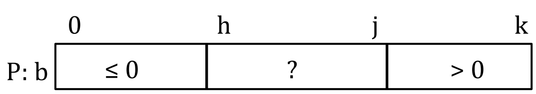
Postcondition R can be written in mathematics like this: $b[0..h-1] \leq 0$ && $b[h..k] > 0$.





Note that the algorithm may *not* change k. The algorithm deals not with the whole array but only with its first k+1 elements.

We will use invariant P shown below. It is a generalization —we'll explain that word later— of the pre- and post-conditions.



1. First loopy question: Does the algorithm start right: is {Q} initialization {P} true?

We have to find the initialization. Initially, for invariant P to be true, it must look like precondition Q. This means that segments b[0..h-1] and b[j+1..k] of P must be empty. By our formula *Follower minus the First*, the number of elements in b[0..h-1] is h–0. So initially, h must be 0. In the same way, the number of values in b[j+1..k] is k–j, so j must equal k. Therefore, the initialization is:

h= 0; j= k;

Wasn't that easy?

2. Second loopy question: Does it stop right: Does P && !B imply R?

The loop must end with R true. To make invariant P look like R, query segment b[h..j] must be empty. Looking at the invariant, you can see that b[h..j] is *not* empty when $h \leq j$ —if h = j, b[h..j] has 1 element. So the loop condition B is $h \leq j$ and !B is h = j+1.

3. Third and fourth loopy questions: Does the repetend make progress toward termination and keep the invariant true?

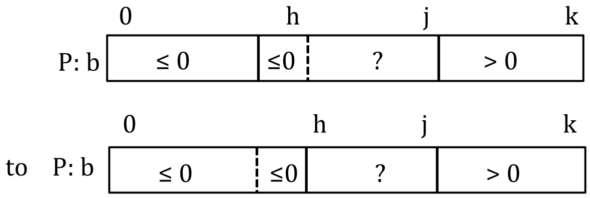
The repetend will get closer to termination by reducing the size of query segment b[h..j]. That means that at least one element in the query segment must be moved to either b[0..h] or b[j+1..k]. Let's see how to do this.

Look at element b[h]. Either $b[h] \leq 0$ or $b[h] > 0$, and we should do something different in each case. So we will use an if-statement as shown to the right. If $b[h] \leq 0$, then b[h] can be placed in the leftmost segment simply by increasing h by 1. To see this in pictures,

```
if (b[h] <= 0) { ... }
else { ... }
```

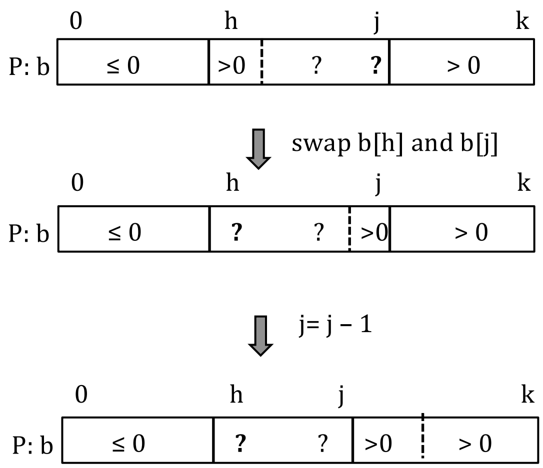
Answering the four loopy questions. Example 2

we change



That's done using h= h+1.

If b[h] > 0, then that value belongs in the last segment. So, as shown below, we swap b[h] with b[j] and then decrease j. Variable h should not be increased, since we do not know what is in it.



This yields the algorithm

```
// Rearrange values of b[0..k] to put the positive values on the right
int h= 0;
int j= k;
// invariant P: show above in a picture
while (h <= j) {
      if (b[h] <= 0) { h= h + 1; }
      else {
         Swap b[h] and b[j];
         j= j – 1;
      }
}
// postcondition R:  shown above in a picture
```