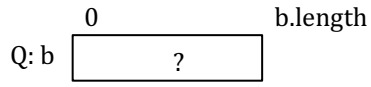
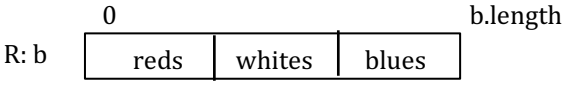
The Dutch National Flag

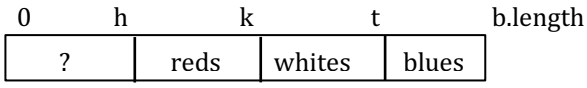
Below is a precondition `Q` —nothing is known about the elements in `b`. Array `b` is known to contain red, white, and blue balls. The purpose of the algorithm is to swap the values of `b` around so that postcondition `R` is true: the red balls are first, then the whites, and then the blues. Edsger Dijkstra developed the problem and its several solutions in the 1970's while doing research on the use of invariants. Red, white, and blue are the colors of the Dutch National Flag.

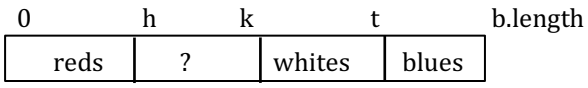


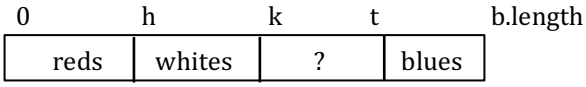


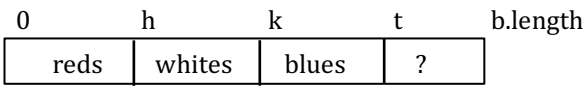
Developing the invariant

A loop or recursion is needed; we decide on a loop. A loop invariant has to be created as a generalization of the precondition and the postcondition. That generalization has to have four segements (at least), labeled ?, reds, whites, and blues. There are actually *four* possibilities for the invariant! They are given below. In each, we have placed three variables `h`, `k`, and `t` to mark the boundaries.









Which invariant should we use? Consider the upper left invariant, and figure out how progress toward termination can be made while maintaining the invariant.

Suppose `b[h]` contains a blue ball. How will it get into the blue section? We have to swap `b[h]` and `b[t]`. But that puts `b[t]`, which was white[1], in `b[h]`. Therefore, another swap is needed: swap `b[h]` and b[`k`]. After that, all three variables `h`, `k`, and `t` have to decemented in order to retruthify the invariant.

Consider the lower left invariant, and let us consider the swaps needed.

(1) If `b[k]` is white, no swap is needed.
(2) If `b[k]` is red, swap `b[k]` and `b[h]`.
(3) If `b[k]` is blue, swap `b[k]` and `b[t]`.

Summary: Use of the upper left invariant requires two swaps for each blue ball. Use of the lower left invariant requires one swap per red and blue and no swap for the white. Thus, the lower left invariant is preferred. A bit of investigation will show that using the upper right invariant will require at most one swap per iteration, while using the lower right invariant suffers from the same problem as the upper left; here, it is 2 swaps for each red ball.

Significance: The significance of what we just did should not be overlooked. We were able to investigate which of two invariants would lead to a better algorithm *without writing a line of code*. The use of invariants makes it easier to think about and analyze algorithms.

Below, we present the algorithm, which was developed using the four loopy questions along with the lower left loop invariant.

```
h= 0; k= 0; t= b.length-1;
// invariant: Lower left invariant, above
while (k <= t) {
   if (b[k] is red) { Swap b[h] and b[k]; h= h+1; k= k+1; }
   else if (b[k] is white) k= k+1;
   else /* b[k] is blue */ { Swap b[k] and b[t]; t= t-1; }
}
```

[1] It is not certain that b[t] was white. There may be no white balls in the array! Our arguments are for the cases that the sections are nonempty, but it can be see that the algorithm works even when some sections are empty.

The Dutch National Flag