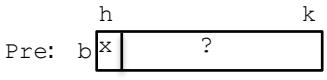
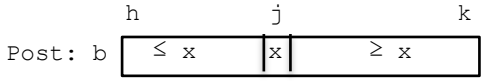
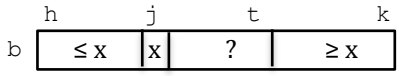
The partition algorithm of quicksort

Suppose `b[h..k]` looks like the precondition given below. Here, `x` is not a program variable but just a name for the contents of `b[h]`. We assume that `h < k` —that is, `b[h..k]` has at least two elements. The partition algorithm swaps the values of `b[h..k]` and stores a value in `j` to truthify postcondition `Post`, shown to the right. If `b[h..k]` contains several elements equal to `x`, it doesn't matter whether they are placed in the left or the right segment.

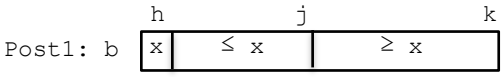




You may have seen the development of a partition algorithm using the invariant shown to the right. Instead, let's try something different. We use a different invariant, and you will see that the algorithm makes at most `(k+1-h)/2` swaps.

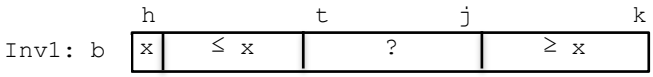


Let's leave the `x` in `b[h]` and swap values in `b[h+1..k]` to truthify postcondition `Post1`:



Then, to truthify `Post` above, all we have to do is swap `b[h]` and `b[j]`.

Given precondition `Pre`, we want an algorithm that swaps elements of `b[h+1..k]` to truthify `Post1`. We develop the loop invariant in the standard way:



The initialization is: `t= h+1; j= k;` .

The loop must continue as long as the `?` section contains a value, i.e. as long as `t ≤ j`. We see that if `j = t-1`, postcondition `Post1` holds.

The repetend will make progress by reducing the size of section `b[t..j]`. If `b[t] ≤ x`, increase `t`. If `b[j] ≥ x`, decrease `j`; if `b[j] < x < b[t]`, swap `b[t]` and `b[j]`, increase `t`, and decrease `j`. We end up with this code, which we call `Partition1`:

```
Partition1:  int t= h+1;
             int j= k;
             // invariant: Inv1
             while (t <= j) {
                if (b[t] <= x) t= t+1;
                else if (b[j] >= x) j= j-1;
                else { Swap b[t] and b[j]; t= t+1; j= j-1; }
             }
```

Putting it all together, we write this partition algorithm as a method:

```
/** Give precondition Pre (above) swap elements of b[h..k] to truthify Post and return j. */
public static int partition(int b, int h, int k) {
    // Pre
    Partition1
    // Post1
    Swap b[h] and b[j];
    // Post
    return j;
}
```

It is readily seen that this method takes time proportional to the size of `b[h..k]` It makes at most `(k+1-h)/2` swaps. It uses `O(1)` space.