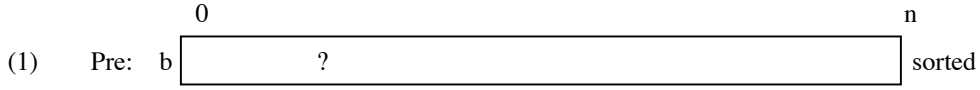
Mistakes in developing binary search

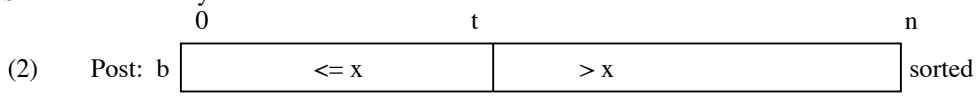
The quiz on Thursday asked students to write binary search. Some of the answers were pretty bad. Some people did not learn our binary search but wrote something they had seen before —and badly. That won't do. In this document, we discuss various problems with the binary searches that were "developed" on the quiz.

1. Write the pre- and post-conditions first, and label them!
Reason: If you don't have the spec of the algorithm, how can you write the algorithm? How can the reader know what you are doing? If you don't label them, how can the reader know what you are doing?

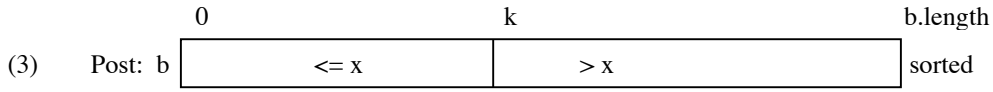
You should write this something like this:

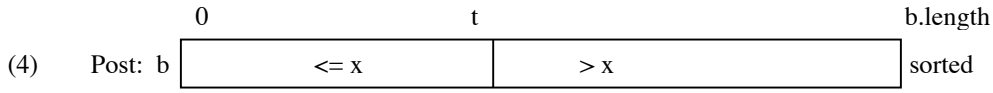


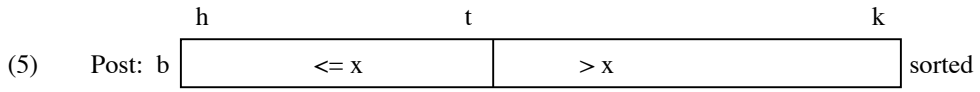
Store in t to truthify:



You could instead write any of the following three as the postcondition, and there are other variations:



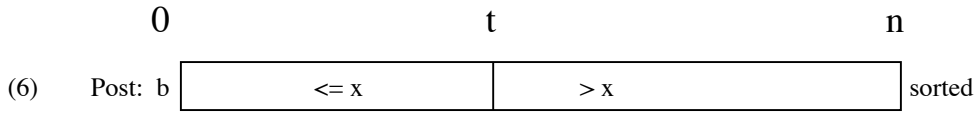




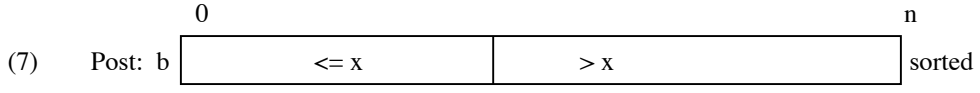
So, you can label the segment to be searched any way you want. You can define the boundary of the two sections in different ways. But if you don't have a segment of values that are ≤ x and a segment of values that are > x, **then you are not writing the binary search as requested**.

2. Common mistakes in writing the postcondition.
2.1 **Not writing the postcondition**. This is a real problem. How can we know what the algorithm is supposed to do?

2.2 **Putting the indices that mark boundaries in the wrong place**. In the following, how can you tell whether t labels the last element of the first section or the first element of the last section? What is the 0 doing to the left of the array?



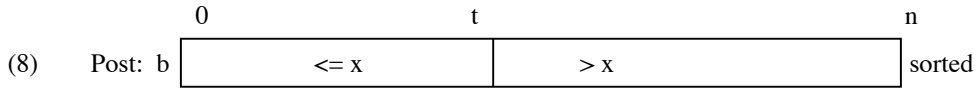
2.3 **Not marking a boundary.** In the following, there is no t, so what is supposed to be stored in t?



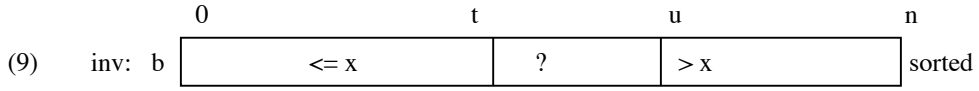
2.4 Writing "b[t] is the rightmost occurrence of x" in the invariant. We have said that if x is in the array section to be searched, b[t] will be the rightmost occurrence of x. This is a help in remembering what the postcondition is: there is a section of values ≤ x and a section ≥ x. This should help in writing any of the postconditions (2)-(5). However, there is nothing that requires x to be in the array, so don't put in the invariant "b[t] is the rightmost occurrence of x".

3. Writing the invariant.
A key here is that you *develop* theinvariant and algorithm from the given pre- and post-condition. Don't memorize the invariant and code! Develop them. If you write postcondition (5) above but then write the invariant and algorithm that goes with precondition (2), that is an indication that you are not developing but trying to write down something that you memorized. And you are doing it badly.

In the rest of this document, we will use postcondition (8):



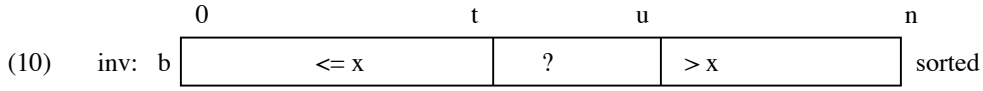
Here is the invariant:



4. Mistakes in writing the invariant.
4.1. Putting t in the wrong place. For example, t should not mark the first element of the unknown section.

4.2. Not writing t anywhere, but using a different variable. How can you know what to store in t initially if t doesn't appear in the invariant?

4.3 Putting u as the index of the last unknown, as shown below. Remember that progress must be made in the repetend by removing half of the unknowns (this is binary search), and with the following invariant, progress may not be made properly.



For example, suppose the unknown segment has 1 value, so that t+1 = u. Then the average e = (t+u)/2 is t (remember, this is **int** arithmetic). So setting t to e does not remove anything from the unknown section.

5. Developing the algorithm.
For later purposes of explanation, we present the algorithm developed with postcondition (8) and invariant (9):

```
t= –1; u= n;
// invariant: picture (9) above
while ( t+1 != u) {
    int e= (t+u) / 2;
    if ( b[e] <= x)  t= e;
    else u= e;
}
// postcondition: picture (8) above
```

This was developed using the four loopy questions. We now discuss mistakes people make in writing this algorithm.

5.1 Writing an algorithm that does not go with the invariant. Suppose you use postcondition (9) above but write something like this:

```
t= 0;  h= b.length;
while ( t != u) {  … }
```

This is a clear indication that you are did not look at the invariant and postcondition when writing the algorithm. You are trying to write what you memorized. Why is h assigned the value b.length, when neither the postcondition nor the invariant mentions b.length?

5.2 Forgetting to initialize t and u before the loop. Remember the four loopy questions —the first is: How does it start? Meaning: what should be done to truthify the invariant? If ask and answer each loopy question in turn (here, questions 3 and 4 have to be combined, because there are two ways to make progress), you lessen the chance of making mistakes.

5.3. Writing the loop condition t != u. If you use the loop condition t != u, you are not calculating properly. Here are two ways to think about it:

1. The unknown section is b[t+1..u-1]. It is empty when t+1 – u = 0, or when t+1 = u.
2. Look at invariant (9). The unknown section is empty when t is next to u. that means t+1 = u.

5.4. Writing the repetend like this:

```
int e= (t+u)/2;
if (b[e] <= x)
    t= e;
u= e
```

The repetend should assign to t or u, but not both. You won't make the this mistake if you think as follows: There are two cases: t has to be increased or u has to be decreased. Let's first write the structure of the if-statement:

```
if ( b[e] <= x) {        }
else {                   }
```

Then figure out what to do in each case. Develop all if statements in this fashion.

5.5 Incrementing t or decrementing u. The occurrence of things like t= t+1 or u= u–1 is a clear sign that you are not developing the repetend using the invariant.

5.6. Writing a return statement. Don't write a return statement! This is not a method, it is simply a sequence of statements to store something in t.

5.7 Stopping when x is found. Don't test for b[e] = x! There is nothing in the invariant or postcondition that would suggest the need for it. Making this test is a clear sign that you are not developing the required algorithm.