

Exercise 1. Suppose we have a non-empty array X of n elements known to be in sorted order, i.e.,

$$X_0 \leq X_1 \leq X_2 \leq \dots \leq X_i \leq X_{i+1} \dots \leq X_{n-2} \leq X_{n-1}$$

Given a **key** k , we want an efficient algorithm which finds, if possible, the index i of some element of X with the property that $X_i = k$. If no such index exists, we want to know the index i where k belongs if it were to be inserted into the array. More precisely, this means that $X_{i-1} < k < X_i$. For simplicity, assume the existence of sentinel values at each end of the array: $X_{-1} = -\infty$ and $X_n = +\infty$. (The idea here is that $-\infty$ is smaller than any other key and that $+\infty$ is larger than any other key.) Show how to use **binary search** to solve this problem.

Solution. Don't let anyone tell you otherwise: binary search is more subtle than you might first think. Let's think carefully and get the details right! For the solution here, we will use the strategy of "wishful thinking," coupled with careful use of loop invariants.

Wishful Thinking

Imagine that the "hard work" has been done. What would we like to know about the array? Oftentimes, a diagram can be a useful way to convey the answer to this kind of question. For example, here is a diagram of what we would like to know about X :

0	i	n
$< k$	$\geq k$	

This diagram is a compact way of stating that X_0, X_1, \dots, X_{i-1} are all less than k , whereas $X_i, X_{i+1}, \dots, X_{n-1}$ are all at least as large as k . Consider two cases:

- If k is present in the array, its index is i . There may be duplicate values of k , in which case i is the leftmost index of the k values. The range of possible values for i is $0 \leq i \leq n - 1$.
- If k is absent, then i is the index of where k belongs if we wanted to insert it. In this case, the possible range of values is $0 \leq i \leq n$. Note: when k is larger than every value in X , we have $i = n$.

If we can *somehow* determine where to place the "fence" i , we essentially will have solved the original problem, as follows:

```

if  $i = n$ 
    return  $\langle \text{FALSE}, n \rangle$  //  $k$  is absent and belongs at the end
elseif  $X_i = k$ 
    return  $\langle \text{TRUE}, i \rangle$  //  $k$  is present
else return  $\langle \text{FALSE}, i \rangle$  //  $k$  is present and belongs in the interior
    
```

Loop Invariant

Since we've all seen binary search before, we have a rough idea of how it is going to work. Essentially, we keep track of a portion of the array and continue "squeezing in", throwing out one half at a time. Let i and j be two index values that keep track of the portion of the array we need to explore further. Using our wishful thinking as a guide, we can devise the following invariant:

0	i j	n
$< k$	uninspected	$\geq k$

More traditionally, i and j would have names like **low** and **high**, but the diagram is clearer if single letters are used.

Step 1: Establish the Invariant

Before we begin the loop, we need to initialize the variables to make our invariant hold. Here is where the invariant has a payoff: **no guesswork is needed** to establish the initial values. Since all elements of the array are initially uninspected, we need $i = 0$ and $j = n - 1$. (Compare this with the invariant diagram to see that these are the correct initializations.)

Step 2: Maintain the Invariant

We now ask one of the more important questions involved with loop design:

What loop body will reduce the size of the uninspected region and, at the same time, preserve the invariant?

Of importance here is that we only need to worry about what happens as a result of *one* iteration.

Due the familiarity of binary search, this will be relatively simple especially since we know exactly what we're trying to do. Find the midpoint of the uninspected region and make a probe at that spot. Let's say the midpoint lands at m . If $X_m \geq k$, then all of the elements X_m through X_j must also be greater than or equal to k . This means we can set $j = m - 1$. Conversely, if $X_m < k$, then all the elements X_i through X_m must also be less than k , so we can set $i = m + 1$. Either way, we get to reduce the uninspected region and preserve the invariant. **No guesswork is needed.**

Step 3: Determine loop termination

For a **while** loop, we need a Boolean expression which describes the condition of *not* being done. For us, we are not done when we have uninspected elements. (Look back at the “wishful thinking” diagram — every element in the array was accounted for.)

The loop invariant once again has a payoff: we see that there are uninspected elements whenever $i \leq j$. **No guesswork is needed.**

Step 4: Assemble the pieces

BINARYSEARCH(X, n, k)

```
// X is a sorted array of n elements and k is the search key
// BINARYSEARCH returns the ordered pair ⟨outcome, i⟩:
//   outcome = TRUE implies the key was found;  $X[i] = k$ 
//   outcome = FALSE implies the key is absent;  $X_{i-1} < k < X_i$ 
```

```
1   $i = 0$            // Establish the invariant
2   $j = n - 1$ 
3  while  $i \leq j$ 
4      // The loop invariant diagram from page 1 holds at this point
5       $m = \lfloor (i + j) / 2 \rfloor$ 
6      if  $X_m \geq k$ 
7           $j = m - 1$ 
8      else  $i = m + 1$ 
9  // At this point we have established the “wishful thinking” diagram
10 if  $i = n$ 
11     return ⟨FALSE,  $n$ ⟩ //  $k$  is absent and belongs at the end
12 elseif  $X_i = k$ 
13     return ⟨TRUE,  $i$ ⟩ //  $k$  is present
14 else return ⟨FALSE,  $i$ ⟩ //  $k$  is present and belongs in the interior
```