Chapter 12 : The Binary Chop.

In which we explore a classic algorithm.

Given f[0..N] of int, $1 \le N$ and X : int[1]

We are asked to construct a program to meet the following specification

$$\{f.0 \le X \ \wedge \ X < f.N \}$$

$$S$$

$$\{f.i \le X \wedge \ X < f.(i + 1) \ \wedge \ 0 \le i < N \}$$

What we are being asked to find is a "pivot" point a value of i where

$$f.i \le X \wedge \ X < f.(i+1)$$

Before we start it is natural to ask whether such a point is guaranteed to exist.
In class we argue that indeed such a point will exist (the precondition guarantees it).
The pivot need not be unique.

How shall we proceed? The shape of the postcondition offers no real guidance.
But we do observe that in the expression

$$f.i \le X \wedge \ X < f.(i+1)$$

the 2 terms are very tightly bound together. So we introduce a new (suitably bound variable) and rewrite the postcondition as

$$\{f.i \le X \wedge \ X < f.j \ \wedge \ j = i+1 \ \wedge \ 0 \le i < N \}$$

There still is not much about the shape that suggests what to do next. But that is probably because in every example up to now what suggested using a loop was the presence of a quantified expression. Although this is a good guideline, we can use a loop in other circumstances. So, we will choose invariants and a guard and see if constructing a loop can help us.

[1] Note that the upper bound, N, is included here.

Choose invariants.

For invariants we choose

$$P0 : f.i \le X \wedge X < f.j$$
$$P1 : 0 \le i < N$$

Note that one of the major reasons why we choose something as an invariant is because it is easy to establish. Looking at the precondition we can clearly see that these will be easy to establish.

Guard.

$$j \ne i+1$$

Variant.

As our variant we propose $j - i$.

Instantiate the loop template.

Here we decide to put in the parts we already know into our loop template.

```
{f.0 ≤ X ∧ X < f.N }

"establish the loop invariants"

{f.i ≤ X ∧ X < f.j }
Do j ≠ i + 1 →       {f.i ≤ X ∧ X < f.j ∧ j ≠ i + 1 }

                     "decrease variant and maintain invariant"

                     {f.i ≤ X ∧ X < f.j }
Od
{f.i ≤ X ∧ X < f.j ∧ j = i + 1 ∧ 0 ≤ i < N }
```

Establishing the invariant.
From the precondition is it easy to see how to do this.

$$i, j := 0,N$$

Decreasing the variant.

We begin with $i = 0$ and $j = N$. Our variant is $j - i$. Our guard guarantees us that within the loop body i and j will always be at least 2 apart. That is to say

$$\langle \exists\ h : i < h < j \rangle$$

If we gave either i or j the value of such a h then we would certainly decrease the variant. But under what circumstances would this maintain the invariant? Lets try it and see.

Case i := h

$\{f.i \le X \wedge X < f.j \wedge j \ne i + 1\}$

$\{\mathbf{f.h \le X} \wedge X < f.j\}$

$i := h$

$\{f.i \le X \wedge X < f.j\}$

Look at the precondition for the assignment. The second term is true because it is part of the invariant. The first term (in bold) describes the circumstances under which we can perform this assignment and maintain the invariant. In other words we have

If $f.h \le X \rightarrow\ \ i := h$

Case j := h

$\{f.i \le X \wedge X < f.j \wedge j \ne i + 1\}$

$\{f.i \le X \wedge \mathbf{X < f.h}\}$

$j := h$

$\{f.i \le X \wedge X < f.j\}$

Look at the precondition for the assignment. The first term is true because it is part of the invariant. The second term (in bold) describes the circumstances under which we can perform this assignment and maintain the invariant. In other words we have

If $X < f.h \rightarrow\ \ j := h$

As $(X < f.h \vee f.h \leq X) \equiv$ true, the precondition of the If..Fi is true.

Putting it together we get

```
{$f.0 \leq X \wedge X < f.N$}
i, j := 0,N
{$f.i \leq X \wedge X < f.j$}
;Do $j \neq i + 1$ →      {$f.i \leq X \wedge X < f.j \wedge j \neq i + 1$}

                          “choose h so that $i < h < j$”

                          If $f.h \leq X$ →   i := h
                          [] $X < f.h$ →   j := h
                          Fi

                          {$f.i \leq X \wedge X < f.j$}
Od
{$f.i \leq X \wedge X < f.j \wedge j = i + 1 \wedge 0 \leq i < N$}
```

All that remains is to choose the value for h. We could choose i+1 or even j-1. But that would destroy the nice symmetry that the program seems to have. So to preserve the symmetry lets choose h as close to the middle between i and j.

The finished algorithm.

```
{$f.0 \leq X \wedge X < f.N$}
i, j := 0,N
{$f.i \leq X \wedge X < f.j$}
;Do $j \neq i + 1$ →      {$f.i \leq X \wedge X < f.j \wedge j \neq i + 1$}

                          h := (i+j) div 2

                          ;If $f.h \leq X$ →   i := h
                          [] $X < f.h$ →   j := h
                          Fi

                          {$f.i \leq X \wedge X < f.j$}
Od
{$f.i \leq X \wedge X < f.j \wedge j = i + 1 \wedge 0 \leq i < N$}
```

And that is the classic algorithm known as the Binary Chop. It has complexity O(log.N) which is rather nice. And the reason it has that complexity is because we decided to keep the symmetry.

The Binary Search.

Suppose we are now told that f is ordered in ascending order (the $\leq$ relation). Would anything change? Would we still have a pivot? Well the presence of a pivot was guaranteed by the precondition

$$\{f.0 \leq X \wedge X < f.N\}$$

and that has not changed.

However, as we will argue in class the pivot is now unique.

So suppose someone asked us to determine whether X actually was present in the array. If it is, the only place it can be is at the left of the pivot. So all we need do is add a single extra line at the end of our program

```
{f.0 ≤ X ∧ X < f.N }
i, j := 0,N
{f.i ≤ X ∧ X < f.j }
;Do j ≠ i + 1 →     {f.i ≤ X ∧ X < f.j ∧ j ≠ i + 1 }
                    h := (i+j) div 2

                    ;If f.h ≤ X →  i := h
                    [] X < f.h →   j := h
                    Fi

                    {f.i ≤ X ∧ X < f.j }
Od
{f.i ≤ X ∧ X < f.j ∧ j = i + 1 ∧ 0 ≤ i < N }

;Write (f.i = X)
```

This gives us the classical algorithm known as the Binary Search.