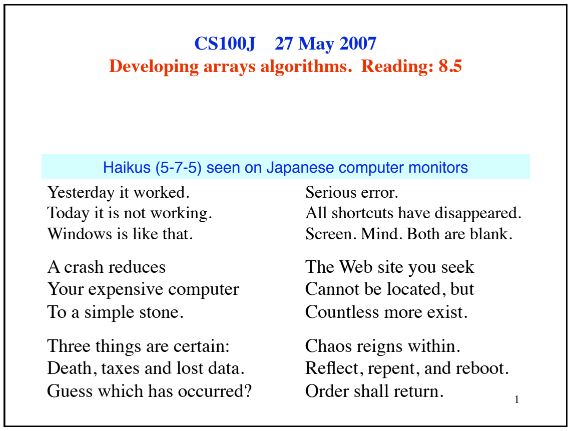


CS100J 27 May 2007
Developing arrays algorithms. Reading: 8.5

Haikus (5-7-5) seen on Japanese computer monitors

Yesterday it worked.
Today it is not working.
Windows is like that.

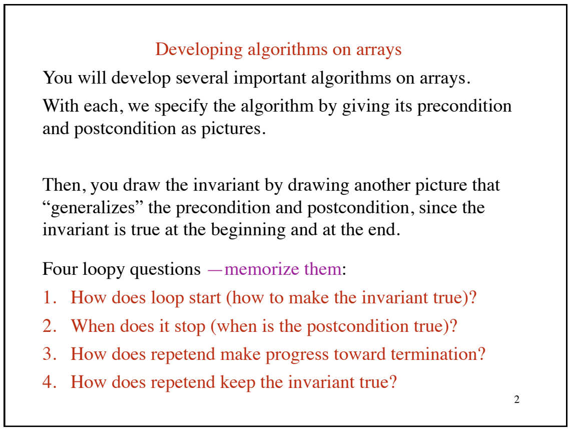
A crash reduces
Your expensive computer
To a simple stone.

Three things are certain:
Death, taxes and lost data.
Guess which has occurred?

Serious error.
All shortcuts have disappeared.
Screen. Mind. Both are blank.

The Web site you seek
Cannot be located, but
Countless more exist.

Chaos reigns within.
Reflect, repent, and reboot.
Order shall return.



Developing algorithms on arrays

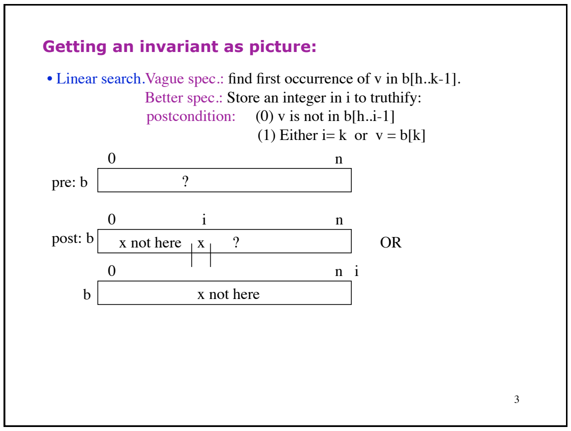
You will develop several important algorithms on arrays.

With each, we specify the algorithm by giving its precondition and postcondition as pictures.

Then, you draw the invariant by drawing another picture that "generalizes" the precondition and postcondition, since the invariant is true at the beginning and at the end.

Four loopy questions —memorize them:

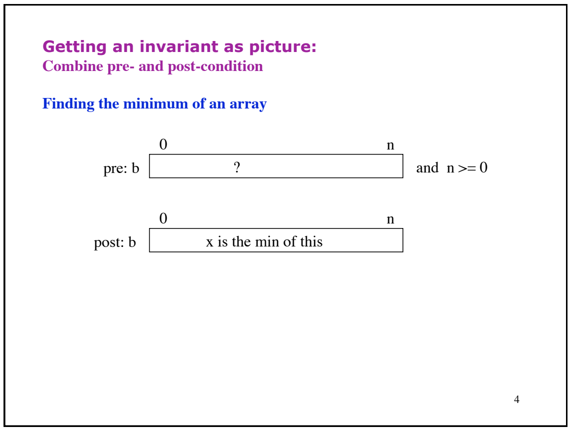
1. How does loop start (how to make the invariant true)?
2. When does it stop (when is the postcondition true)?
3. How does repetend make progress toward termination?
4. How does repetend keep the invariant true?



Getting an invariant as picture:

- Linear search.Vague spec.: find first occurrence of v in b[h..k-1].
  Better spec.: Store an integer in i to truthify:
  postcondition: (0) v is not in b[h..i-1]
  (1) Either i= k or v = b[k]

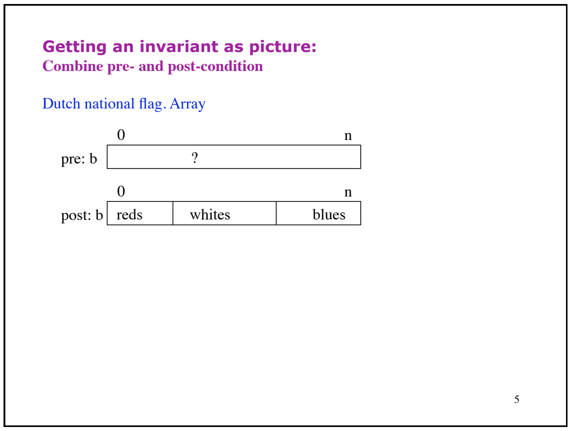
0
n
pre: b
?
0
i
n
post: b
x not here
x
?
OR
0
n
i
b
x not here



Getting an invariant as picture:
Combine pre- and post-condition

Finding the minimum of an array

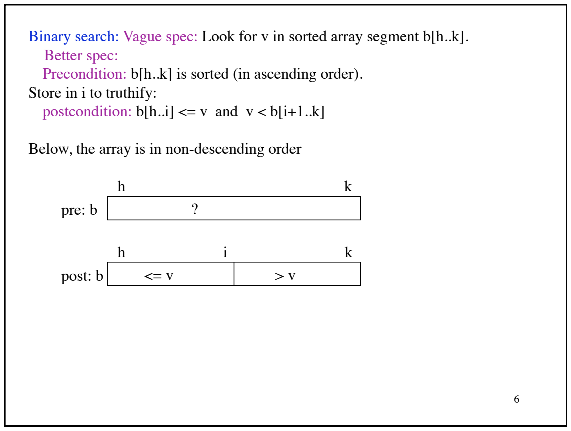
0
n
pre: b
?
and n >= 0
0
n
post: b
x is the min of this



Getting an invariant as picture:
Combine pre- and post-condition

Dutch national flag. Array

0
n
pre: b
?
0
n
post: b
reds
whites
blues

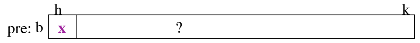


Binary search: Vague spec: Look for v in sorted array segment b[h..k].
Better spec:
Precondition: b[h..k] is sorted (in ascending order).
Store in i to truthify:
postcondition: b[h..i] <= v and v < b[i+1..k]

Below, the array is in non-descending order

h
k
pre: b
?
h
i
k
post: b
<= v
> v

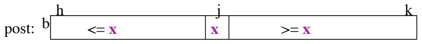
Partition algorithm: x is called the pivot value


pre: b

h k

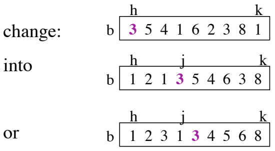
x ?

Swap elements of b[h..k] to produce:


post: b

h j k

<= x x >= x


change:

h k

b 3 5 4 1 6 2 3 8 1

into

h j k

b 1 2 1 3 5 4 6 3 8

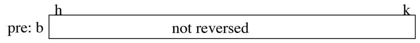
or

h j k

b 1 2 3 1 3 4 5 6 8

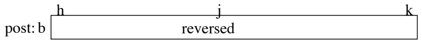
(x is not a program variable; it just denotes the value initially in b[h].)

Reversing array segment b[h..k]


pre: b

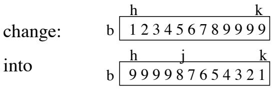
h k

not reversed


post: b

h j k

reversed


change:

h k

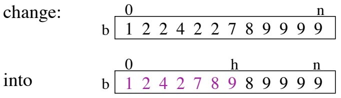
b 1 2 3 4 5 6 7 8 9 9 9 9

into

h j k

b 9 9 9 9 8 7 6 5 4 3 2 1

Remove adjacent duplicates


change:

0 n

b 1 2 2 4 2 2 7 8 9 9 9 9

into

0 h n

b 1 2 4 2 7 8 9 8 9 9 9 9

don’t care what is
in b[k+1..n]

postcondition:

b[0..h] = initial values in b[0..n] but with adj dups removed