## CS1110 2 November 2010

## Developing array algorithms. Reading: 8.3..8.5

Important point: how we create the invariant, as a picture

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.

---

Prelim next Tuesday, 7:30PM. Olin 155 and 255
Review session, Sunday 1–3. Phillips 101
Handout describes what will be covered.

Quiz in class, Thursday, 4 October
Memorize the 4 loopy questions and be able to tell whether a given loop satisfies them or not.

Reason for quiz:
1. You need to understand the 4 loopy questions in order to understand the array algorithms we will be developing.
2. You need to know about the 4 loopy questions for the prelim

---

### Developing algorithms on arrays

We develop several important algorithms on arrays.

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

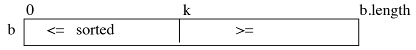
Then, draw the invariant by drawing another picture that “generalizes” the precondition and postcondition, since the invariant is true at the beginning and at the end.

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?

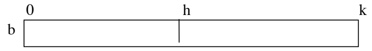
---

### Horizontal notation for arrays, strings, Vectors



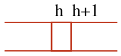
Example of an assertion about an array b. It asserts that:

1. b[0..k–1] is sorted (i.e. its values are in ascending order)
2. Everything in b[0..k–1] is ≤ everything in b[k..b.length–1]



Given the index h of the First element of a segment and the index k of the element that Follows the segment, the number of values in the segment is k – h.

b[h .. k – 1] has k – h elements in it.



(h+1) – h = 1

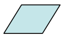
---

Generalize: To derive or induce (a general conception or principle) from particulars.
To make general: render applicable to a wider class

Generalization: All dogs hate cats

**square**
sides: equal
angles: equal

**rhombus**
sides: equal



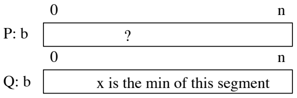
rhombus is a generalization of square
square is a particular kind of rhombus

problem: Tile an 8 x 8 kitchen
generalization: Tile a $2^n$ x $2^n$ kitchen (all using L-shaped tiles)
generalization: Tile an n x n kitchen

---

Invariant as picture: Generalizing pre- and post-condition

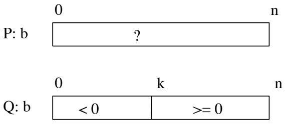
Finding the minimum of an array. Given array b satisfying precondition P, store a value in x to truthify postcondition Q:



and n >= 0 (values in 0..n are unknown)

**The invariant as picture: Generalizing pre- and post-condition**

Put negative values before nonnegative ones. Given is precondition P:



(values in 0..n-1 are unknown)

(values in 0..k-1 are < 0,
values in k..n-1 are > 0)

## The invariant as picture: Generalizing pre- and post-condition

**Dutch national flag**. Swap values of 0..n-1 to put the reds first, then the whites, then the blues. That is, given precondition P, swap value of b [0.n] to truthify postcondition Q:

0 n
P: b | ? | (values in 0..n-1 are unknown)

0 n
Q: b | reds | whites | blues |

---

## How to make invariant look like initial condition

0 n
pre b | ? |

0 j k l n
inv b | reds | whites | ? | blues |

1. Make red, white, blue section empty: use formulas for no. of values in these sections, set j, k , l so that they have 0 elements.

2. Compare precondition with invariant. E.g. in precondition, 0 marks first unknown. In invariant, k marks first unknown. Therefore, k and 0 must be the same.

---

**Partition algorithm:** Given an array b[h..k] with some value x in b[h]:

h k
P: b | x | ? |

Swap elements of b[h..k] and store in j to truthify P:

h j k
Q: b | <= x | x | >= x |

change: b h k | 3 5 4 1 6 2 3 8 1 |

into b h j k | 1 2 1 3 5 4 6 3 8 |

or b h j k | 1 2 3 1 3 4 5 6 8 |

x is called the pivot value.

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

---

## Linear search

Vague spec.: Find first occurrence of v in b[h..k-1].
Better spec.: Store an integer in i to truthify postcondition Q:

Q: 1. v is not in b[h..i-1]
2. i = k OR v = b[k]

h k
P: b | v is in here |

h i k
Q: b | x not here | x | ? |

OR b h i k
| x not here |

---

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

Below, the array is in non-descending order:

h k
P: b | ? |

h i k
Q: b | <= v | > v |

Called binary search because each iteration of the loop cuts the array segment still to be processed in half

---

**Reversal:** Reverse the elements of array segment b[h..k].

h k
precondition P: | not reversed |

h k
postcondition Q: | reversed |

Change: b h k | 1 2 3 4 5 6 7 8 9 9 9 9 |

into b h k | 9 9 9 9 8 7 6 5 4 3 2 1 |