CS1110

Lecture 22: More sequence algorithms

Announcements

Coming up

Friday: A4 due
Monday: Lab 11 due
Tuesday: lecture -> review,
lab -> office hours,
Prelim 2

Is this gonna be on the exam?

Labs up through 11: yes
Assignments up through A4: yes
Lecutres up through Tuesday: yes
Today's lecture: no

Slides by D. Gries, L. Lee, S. Marschner, W. White

Invariants: Keep in mind

- At heart, an invariant is just a way to *document what you want your variables to mean*.

  This is why you want your code to keep the invariant true; you want to keep things consistent in your program, and in your head.

- In our notation, both

  `b[i+1..i]` and

  `b[i..i-1]`

denote an empty sequence.

Linear search in unsorted lists

Goal: *Given* (unsorted) list b, search range h..k with k ≥ h (and h and k valid indices for b), and target value v, *return* index n of v's first occurrence in b[h..k] (-1 if not found)

Restated as postcondition: if n = -1, then v is not in b[h..k]. Otherwise, v = b[n] and v is not in b[h..n-1].

Idea: keep an index i, marking position of next thing unchecked; everything to its left has been verified to not be v.

Linear Search


```
def linear_search(b, h, k, v):
    """(see previous)"""
    i = h
    # inv: v not in b[h..i-1]
    while i <= k and b[i] != v:
        i = i + 1
    # post: b[i] = v and v not in b[h..i-1], or,
    #       i = k+1 and v not in b[h..k]
    n = i if i <= k else -1
    return n
```

Analyzing the Loop

1. Does the initialization make **inv** true?

2. Is **post** true when **inv** is true and **condition** is false?

3. Does the loop body make progress?

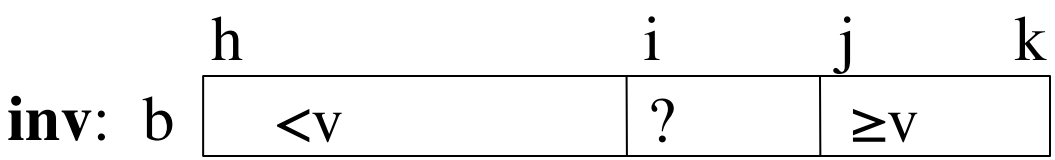
4. Does the loop body keep **inv** true?

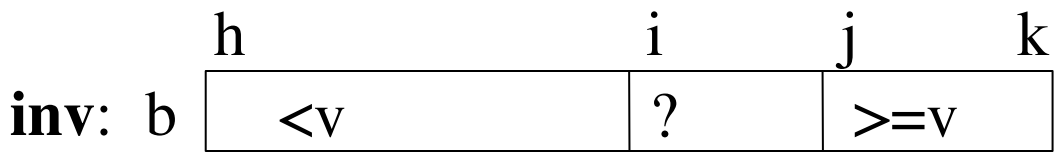
Binary search in *sorted* lists

Goal: *Given* **sorted** list b, search range h..k with k ≥ h (and h and k valid indices for b), and target value v, *return* index n of v's first occurrence in b[h..k] (-1 if not found)

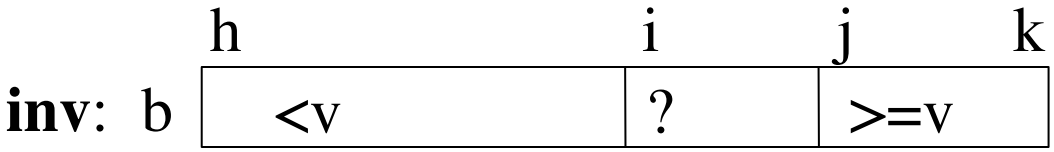
Restated as postcondition: if n = -1, then v is not in b[h..k]. Otherwise, v = b[n] and v is not in b[h..n-1].

Idea: keep indices i and j, marking position of next thing not known to be < v, and the first thing known to be ≥ v. Check in the middle.






```
def bin_search(b, h, k, v):
    """(see previous)"""
    Q1: (A) i = h; j = k    (B) i = h-1; j = k+1    (C) i = h-1; j = k    (D) i = h; j = k+1
    # inv: b[h..i-1] < v, b[j..k] >=v, i <= j
    while Q2:    (A) i == j        (B) i < j     (C) i <= j
        mid = (i + j)/2
        if b[mid] < v:
            Q3:     (A) i = mid        (B) i = mid+1
        else:
            j = mid    # may skip vast section of b
    return i if i <= k and b[i] == v else -1
    # post: b[n] = v and v not in b[h..n-1], or, n = -1 and v not in b[h..k]
```




```
def bin_search(b, h, k, v):
    """(see previous)"""
    i = h; j = k+1
    # inv: b[h..i-1] < v, b[j..k] >=v, i <= j
    while i < j:
        mid = (i + j)/2
        if b[mid] < v:
            i = mid + 1  # may skip vast section of b
        else:
            j = mid      # may skip vast section of b
    return i if i <= k and b[i] == v else -1
    # post: b[n] = v and v not in b[h..n-1], or, n = -1 and v not in b[h..k]
```

Analyzing the Loop

1. Does the initialization make **inv** true?

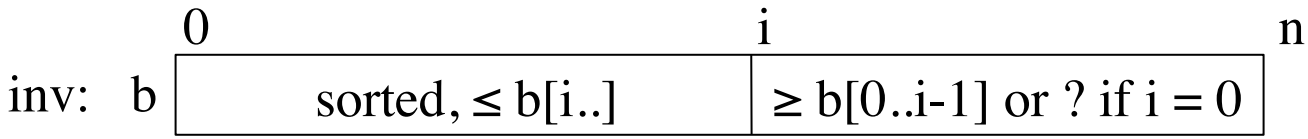
2. Is **post** true when **inv** is true and **condition** is false?

3. Does the loop body make progress?

4. Does the loop body keep **inv** true?

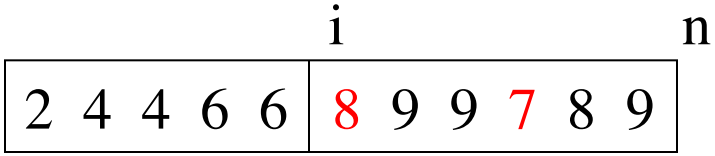
Sorting: Selection Sort

Selection Sort:

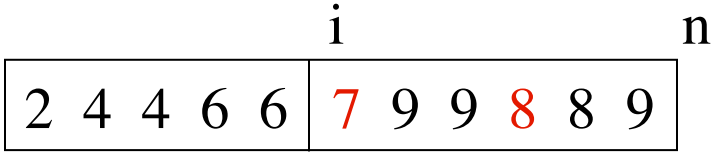


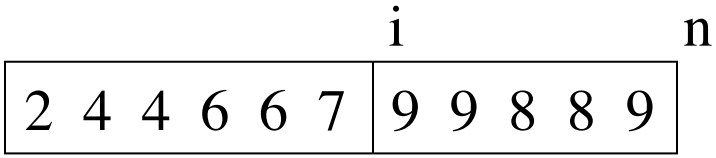
INITIALIZE AND COMPLETE

```
def sort(b):
    """Sort list b in place."""
    ...
    while ...:
        j = min_index(b, i, n)
        # b[j] is min item in b[i..n-1]
        ...
```

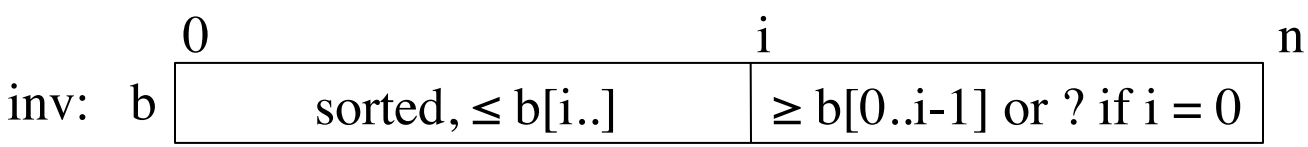


Note the swap of the reds



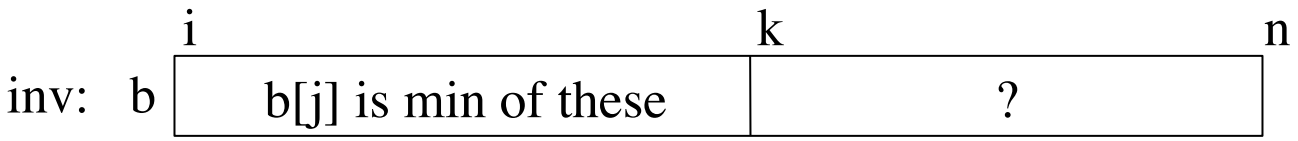


```
def selection_sort(b):
    """Sort list b in place."""
    n = len(b)
    i = ?
    # inv: b[0..i-1] sorted;
    #        b[i..n-1] >= b[0..i-1]
    while ?
        j = min_index(b, i, n)
        # b[j] is min item in b[i..n-1]
        ?, ? = ?, ?
        i += 1
    # post: b[0..n-1] sorted
```



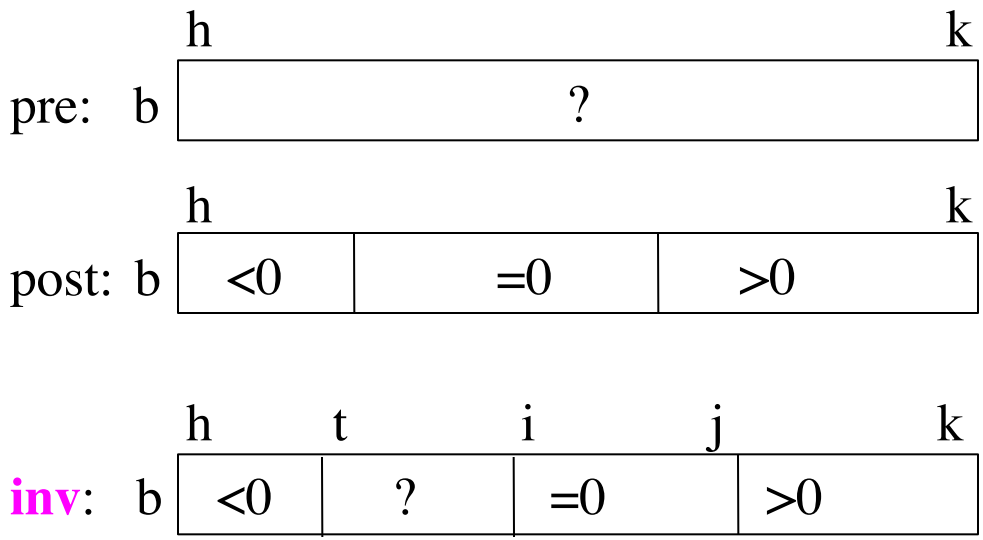
Helper for Selection Sort

```
def min_index(b, i, n):
    """return index of min item in b[i...n-1]. Pre: i <= n."""
    j = ?
    k = ?
    # inv: ?
    while ?
        j = ?
        k += 1
    # post: ?
```



Famous "Sort-Like" Example

- Dutch national flag: tri-color
  - Sequence of h..k of red (<0), white (=0), blue (>0) "pixels"
  - Arrange to put <0 first, then =0 , then >0, return "split pts"



(values in h..k are unknown)

b[h..t-1] <0, b[t..i-1] unknown, b[i..j] =0, b[j+1..k] >0

Dutch National Flag Algorithm

```
def dnf(b, h, k):
    """(DNF explanation omitted for space.)
    Returns: split-points as a tuple (i,j)"""
   # init?
    # inv: b[h..t-1] < 0, b[t..i-1] ?, b[i..j] = 0, b[j+1..k] > 0
    while t < i:
        if b[i-1] < 0:
            #  what?

        elif b[i-1] == 0:
            # what?
        else:
            # what?

    # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    return (i, j)
```

Dutch National Flag Algorithm

```
def dnf(b, h, k):
    """Returns: partition indices as a tuple (i, j)"""
    t = h; i = k+1, j = k;
    # inv: b[h..t-1] < 0, b[t..i-1] ?, b[i..j] = 0, b[j+1..k] > 0
    while t < i:
        if b[i-1] < 0:
            b[i-1], b[t] = b[t], b[i-1]
            t = t+1
        elif b[i-1] == 0:
            i = i-1
        else:
            b[-1], b[j] = b[j], b[i-1
            i = i-1; j = j-1
    # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    return (i, j)
```