

# CS1110

## Lecture 21: **More sequence algorithms**

### Announcements

Two morals from A4:

- Sometimes even seemingly random human behavior can be predicted precisely (e.g., fraction-converted fixed point).
- A good enough idea (small  $t$ ) promoted by even a small but vocal group (large  $d$ ) can *change the whole world*.

Typo in A6 `_drawHBar` spec: see Piazza @309.

No office hours *next* Wed-Fri (can't start grading until Thu Apr 18)

*Next Tue lab = office hours. No next Wed lab at all.*

*Processed regrade requests on the front table by end of class.*

# 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  $h..k-1$  for  $k \geq h$  and  $h$  and  $k$  valid indices for  $b$ , and target value  $v$ , *return* index  $n$  of  $v$ 's first occurrence in  $b[h..k-1]$  (-1 if not found)

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



If  $i < k$  and  $b[i] = v$ , return  $i$  as  $n$ ; if  $i == k$ ,  $v$  isn't in  $b$ .

# Linear Search



```
def linear_search(b,h,k,v):  
    """(see previous)"""  
    i = h  
    # inv says: b[h..i-1] not v; start: b[h..h-1] not v;  
    # end: b[i] is v or i is k.  
    while i < k and b[i] != v:  
        | i = i + 1  
    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 repetend make progress?
4. Does the repetend keep **inv** true?

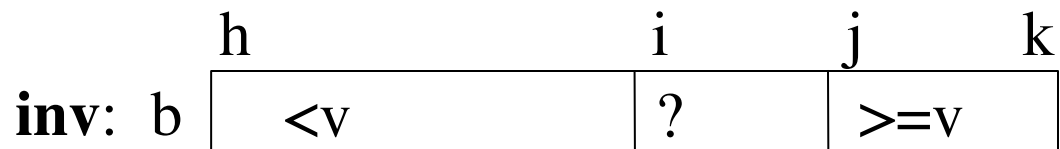
# Binary search in sorted lists

---

**Goal:** Given sorted list  $b$ , search range  $h..k$  for  $k \geq h$  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  $b[h..n-1]$ .

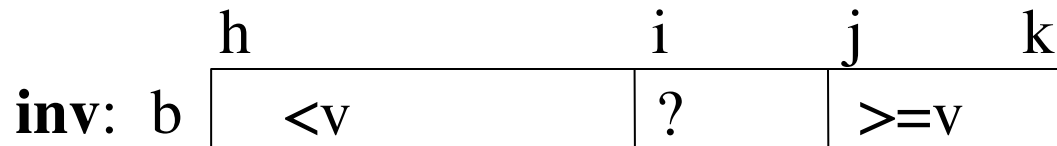
**Idea:** keep indices  $i$  and  $j$ , marking position of next thing not known to be  $< v$ , and the first thing known to be  $\geq v$ . Check halfway btwn 'em.



If  $i \leq k$  and  $b[i] = v$ , return  $i$  as  $n$ ; if  $i > k$  or  $i == j$  and  $b[i] \neq v$  or

# (most of) Binary search implementation

---



`def bin_search(b,h,k,v): # omitting the last return for space`

`"""(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:  $b[h..i-1] < v, b[j..k] \geq v, i \leq j$ ; start:  $b[h..h-1] < v, b[k+1..k] \geq v$

`while` Q2: (A)  $i == j$  (B)  $i < j$  (C)  $i \leq j$

`if`  $b[i] == v$ :

`return`  $i$

$mid = (i+j)/2$

`if`  $b[mid] < v$ :

Q3: (A)  $i = mid$  (B)  $i = mid + 1$

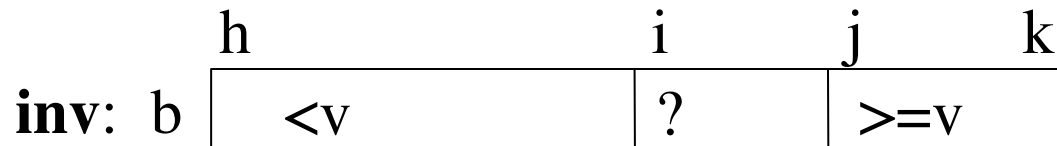
`else:`

4/9/13

`j = mid` # may skip vast section of  $b$

# (most of) Binary search implementation

---



```
def bin_search(b,h,k,v): # omitting the last return for space
    """(see previous)"""
    i=h; j=k+1
    # inv: b[h..i-1] < v, b[j..k] >=v, i <= j; start: b[h..h-1] < v, b[k+1..k] >= v
    while i < j:
        if b[i] == v:
            return i
        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
```

# Sorting: Selection Sort

pre:  $b$ 

0		n
?		

post:  $b$ 

0		n
sorted		

## Selection Sort:

inv:  $b$ 

0		i		n
sorted, $\leq b[i..]$		$\geq b[0..i-1]$ or ? if $i = 0$		

### INITIALIZE AND COMPLETE

while ...:

#  $j$  is min item in  $b[i..n-1]$

$j = i + b[i:n].\text{index}(\min(b[i:n]))$

$i$   $n$

2	4	4	6	6	8	9	9	7	8	9
---	---	---	---	---	---	---	---	---	---	---

Note the swap of the reds

$i$   $n$

2	4	4	6	6	7	9	9	8	8	9
---	---	---	---	---	---	---	---	---	---	---

$i$   $n$

2	4	4	6	6	7	9	9	8	8	9
---	---	---	---	---	---	---	---	---	---	---



# Sorting: Selection Sort

---

## Selection Sort:

inv:  $b$ 

0		i		n
	sorted, $\leq b[i..]$		$\geq b[0..i-1]$ or ? if $i = 0$	

$i=0$

**while**  $i < n$ :

$j = i + b[i:n].\text{index}(\text{min}(b[i:n]))$

$b[i], b[j] = b[j], b[i]$

$i = i + 1$

# 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"

pre:  $b$ 

$h$	$k$
?	

 (values in  $h..k$  are unknown)

post:  $b$ 

$h$	$k$	
$<0$	$=0$	$>0$

inv:  $b$ 

$h$	$t$	$i$	$j$	$k$
$<0$	?	$=0$		$>0$

$b[h..t-1] <0, b[t..i-1]$  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 points 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[i-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)
```

< 0		?			= 0		> 0	
h		t			i	j		k
-1	-2	3	-1	0	0	0	6	3

h		t		i		j		k
-1	-2	3	-1	0	0	0	6	3

←

h			t	i		j		k
-1	-2	-1	3	0	0	0	6	3



h				t		j		k
-1	-2	-1	0	0	0	3	6	3

