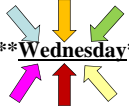


### Announcements

- Final Exam conflicts due tonight at 11:59pm
- Final Exam review sessions on the 14<sup>th</sup>
- Labs on 5/9 and 5/10 will be office hours
- Assignment 5
  - Due 11:59pm on \*\*\***Wednesday**\*\*\* May 10<sup>th</sup>
- Lab 13 is out



### Dutch National Flag Variant

- Sequence of integer values
  - 'red' = negatives, 'white' = 0, 'blues' = positive
  - Only rearrange part of the list, not all

pre: b [ h | | | ? | | | k ]

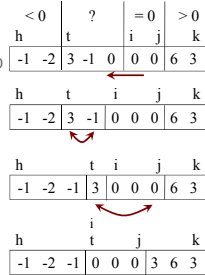
post: b [ | <0 | =0 | >0 | | | ]

inv: b [ | <0 | ? | =0 | >0 | | ]

pre: t = h,  
i = k+1,  
j = k  
post: t = i

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



### Flag of Mauritius

- Now we have four colors!
  - Negatives: 'red' = odd, 'purple' = even
  - Positives: 'yellow' = odd, 'green' = even

pre: b [ h | | | ? | | | k ]

post: b [ | <0 odd | <0 even | ≥0 odd | ≥0 even | ]

inv: b [ | <0, o | <0, e | ≥0, o | ? | ≥0, e | ]



### Linear Search

pre: b [ h | | | ? | | | k ]

post: b [ | v not here | v | | ? | | | k ]

OR

b [ h | | | v not here | | | k ]

inv: b [ | v not here | | | ? | | | k ]

### Linear Search

```
def linear_search(b,c,h):
    """Returns: first occurrence of c in b[h..]"""
    # Store in i the index of the first c in b[h..]
    i = h
    # invariant: c is not in b[0..i-1]
    while i < len(b) and b[i] != c:
        i = i + 1
    # post: c is not in b[h..i-1]
    # i >= len(b) or b[i] == c
    return i if i < len(b) else -1
```

#### Analyzing the Loop

- Does the initialization make **inv** true?
- Is **post** true when **inv** is true and **condition** is false?
- Does the repetend make progress?
- Does the repetend keep the invariant **inv** true?

### Binary Search

- **Vague:** Look for  $v$  in **sorted** sequence segment  $b[h..k]$ .
- **Better:**
  - **Precondition:**  $b[h..k-1]$  is sorted (in ascending order).
  - **Postcondition:**  $b[h..i] \leq v$  and  $v < b[i+1..k-1]$
- Below, the array is in non-descending order:

pre:  $b$   $?$

post:  $b$   $< v$   $> v$

inv:  $b$   $< v$   $?$   $> v$

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

### Binary Search

- Look for value  $v$  in **sorted** segment  $b[h..k]$

pre:  $b$   $?$

post:  $b$   $< v$   $>= v$

inv:  $b$   $< v$   $?$   $>= v$

New statement of the invariant guarantees that we get **leftmost** position of  $v$  if found

Example  $b$  3 3 3 3 3 4 4 6 7 7

- if  $v$  is 3, set  $i$  to 0
- if  $v$  is 4, set  $i$  to 5
- if  $v$  is 5, set  $i$  to 7
- if  $v$  is 8, set  $i$  to 10

### Binary Search

pre:  $b$   $?$

post:  $b$   $< v$   $>= v$

inv:  $b$   $< v$   $?$   $>= v$

New statement of the invariant guarantees that we get **leftmost** position of  $v$  if found

$i = h; j = k+1;$   
**while**  $i \neq j:$

Looking at  $b[i]$  gives **linear search from left.**  
 Looking at  $b[j-1]$  gives **linear search from right.**  
 Looking at middle:  $b[(i+j)/2]$  gives **binary search.**

### Sorting: Arranging in Ascending Order

pre:  $b$   $?$   $n$     post:  $b$  sorted  $n$

**Insertion Sort:**

inv:  $b$  sorted  $?$   $n$

$i = 0$   
**while**  $i < n:$

# Push  $b[i]$  down into its  
 # sorted position in  $b[0..i]$

$0$  2 4 4 6 6 7 5

$0$  2 4 4 5 6 6 7

$i = i+1$

### Insertion Sort: Moving into Position

$i = 0$   
**while**  $i < n:$

push\_down( $b, i$ )  
 $i = i+1$

**def** push\_down( $b, i$ ):

$j = i$   
**while**  $j > 0:$

**if**  $b[j-1] > b[j]:$  swap shown in the lecture about lists

swap( $b, j-1, j$ )

$j = j-1$

$0$  2 4 4 6 6 7 5

$0$  2 4 4 6 6 5 7

$0$  2 4 4 6 5 6 7

$0$  2 4 4 5 6 6 7

### QuickSort

**def** quick\_sort( $b, h, k$ ):

"""Sort the array fragment  $b[h..k]$ """

**if**  $b[h..k]$  has fewer than 2 elements:  
 return

$j = \text{partition}(b, h, k)$

#  $b[h..j-1] \leq b[j] \leq b[j+1..k]$   
 # Sort  $b[h..j-1]$  and  $b[j+1..k]$

quick\_sort( $b, h, j-1$ )  
 quick\_sort( $b, j+1, k$ )

pre:  $b$   $x$   $?$

post:  $b$   $<= x$   $x$   $>= x$