

Lecture 26

Sequence Algorithms (Continued)

Announcements for This Lecture

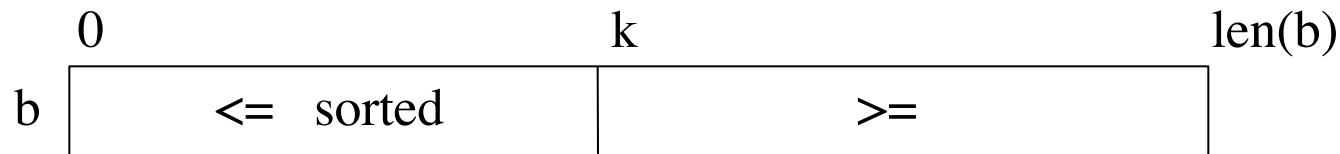
Lab/Finals

- **Lab 12** is the final lab
 - Can use Consulting hours
 - Due **next Wednesday 9:30**
- **Final: Dec 10th 2:00-4:30pm**
 - Study guide is posted
 - Announce reviews next week.
- **Conflict with Final time?**
 - Submit to conflict to CMS
by next TUESDAY!

Assignments

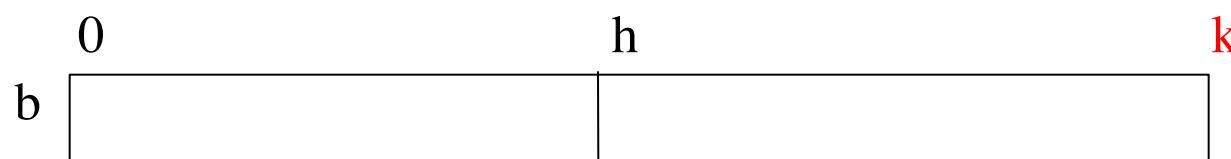
- **A6** is now graded
 - **Mean:** 91.5 **Median:** 95
 - **Std Dev:** 12.9
 - **Mean:** 11 hr **Median:** 10 hr
 - **Std Dev:** 5.7 hr
 - **SEVERAL** AI hearings
- **A7** is due **Tuesday Dec. 4**
 - Due at midnight
 - Some extensions possible

Recall: Horizontal Notation



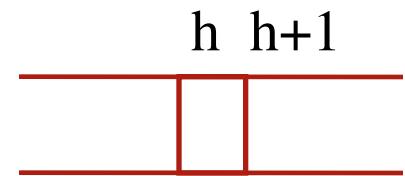
Example of an assertion about an sequence b . It asserts that:

1. $b[0..k-1]$ is sorted (i.e. its values are in ascending order)
2. Everything in $b[0..k-1]$ is \leq everything in $b[k..\text{len}(b)-1]$



Given index h of the **first element** of a segment and index k of the **element that follows** that segment, the number of values in the segment is $k - h$.

$b[h .. k - 1]$ has $k - h$ elements in it.



$$(h+1) - h = 1$$

Partition Algorithm

- Given a sequence $b[h..k]$ with some value x in $b[h]$:

pre:	b	x	?	k
------	---	---	---	---

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

h		i	i+1		k
post: b	$\leq \textcolor{violet}{x}$	$\textcolor{violet}{x}$		$\geq \textcolor{violet}{x}$	

h	i	j	k	
inv: b	$\leq \textcolor{purple}{x}$	$\textcolor{purple}{x}$?	$\geq \textcolor{purple}{x}$

- Agrees with precondition when $i = h, j = k+1$
 - Agrees with postcondition when $j = i+1$

Partition Algorithm Implementation

```
def partition(b, h, k):
    """Partition list b[h..k] around a pivot x = b[h]"""
    i = h; j = k+1; x = b[h]
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x
    while i < j-1:
        if b[i+1] >= x:
            # Move to end of block.
            swap(b,i+1,j-1)
            j = j - 1
        else: # b[i+1] < x
            swap(b,i,i+1)
            i = i + 1
    # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x
    return i
```

partition(b,h,k), not partition(b[h:k+1])

Remember, slicing always copies the list!

We want to partition the **original** list

Partition Algorithm Implementation

```
def partition(b, h, k):
    """Partition list b[h..k] around a pivot x = b[h]"""
    i = h; j = k+1; x = b[h]
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x
    while i < j-1:
        if b[i+1] >= x:
            # Move to end of block.
            swap(b,i+1,j-1)
            j = j - 1
        else: # b[i+1] < x
            swap(b,i,i+1)
            i = i + 1
    # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x
    return i
```

<= x		x	?	>= x	
h	i	i+1	j	k	
1	2	3	1 5 0	6	3 8

Partition Algorithm Implementation

```
def partition(b, h, k):
    """Partition list b[h..k] around a pivot x = b[h]"""
    i = h; j = k+1; x = b[h]
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x
    while i < j-1:
        if b[i+1] >= x:
            # Move to end of block.
            swap(b,i+1,j-1)
            j = j - 1
        else: # b[i+1] < x
            swap(b,i,i+1)
            i = i + 1
    # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x
    return i
```

<= x		x	?	>= x	
h	i	i+1	j	k	
1	2	3	1	5	0

h → i			i+1	j	k
1	2	1	3	5	0

Partition Algorithm Implementation

```
def partition(b, h, k):
    """Partition list b[h..k] around a pivot x = b[h]"""
    i = h; j = k+1; x = b[h]
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x
    while i < j-1:
        if b[i+1] >= x:
            # Move to end of block.
            swap(b,i+1,j-1)
            j = j - 1
        else: # b[i+1] < x
            swap(b,i,i+1)
            i = i + 1
    # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x
    return i
```

<= x		x	?	>= x		
h	i	i+1	j	k		
1	2	3	1	5	0	6 3 8

h → i			i+1	j	k	
1	2	1	3	5	0	6 3 8

h			i	j ←	k
1	2	1	3	0	5 6 3 8

Partition Algorithm Implementation

```

def partition(b, h, k):
    """Partition list b[h..k] around a pivot x = b[h]"""
    i = h; j = k+1; x = b[h]
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x
    while i < j-1:
        if b[i+1] >= x:
            # Move to end of block.
            swap(b,i+1,j-1)
            j = j - 1
        else: # b[i+1] < x
            swap(b,i,i+1)
            i = i + 1
    # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x
    return i

```

$\leq x$	x	?	$\geq x$	
h	i	i+1	j	k
1	2	3	1	5 0 6 3 8
h	\rightarrow i	i+1	j	k
1	2	1	3	5 0 6 3 8
h		i	j \leftarrow	k
1	2	1	3	0 5 6 3 8
h		\rightarrow i	j	k
1	2	1	0	3 5 6 3 8

The diagram illustrates the partitioning process for the array [1, 2, 1, 3, 5, 0, 6, 3, 8] around a pivot $x = 3$. The array is shown in a grid with columns labeled by indices $h, i, i+1, j, k$. Red arrows indicate the movement of elements during the partitioning process:

- In the first step, the element at index $i+1$ (which is 1) is moved to the end of the block (index $j-1$), and the index j is decremented.
- In the second step, the element at index i (which is 1) is moved to the end of the block (index $j-1$), and the index j is decremented.
- In the third step, the element at index i (which is 0) is moved to the end of the block (index $j-1$), and the index j is decremented.

Dutch National Flag Variant

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

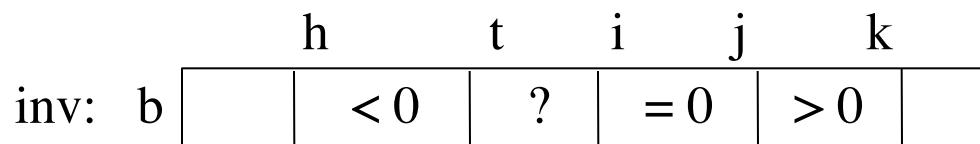
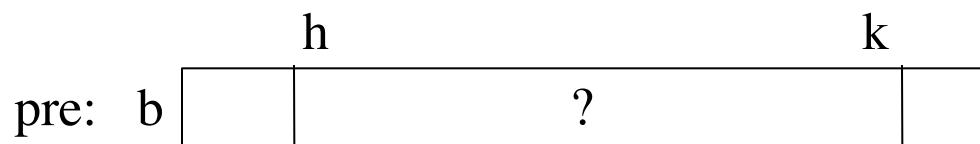
	h		k
pre:	b	?	

	h		k
post: b			
	< 0	$= 0$	> 0

	h	t	i	j	k	
inv:	b	< 0	?	= 0	> 0	

Dutch National Flag Variant

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



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)
```

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

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)
```

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

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

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)

```

< 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

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

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)

```

< 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

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

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

Changing the Invariant

- Different invariants = different code
 - Need to change how we initialize, stop
 - Also need to change the body of the loop

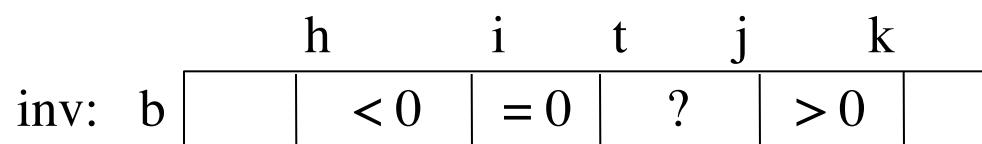
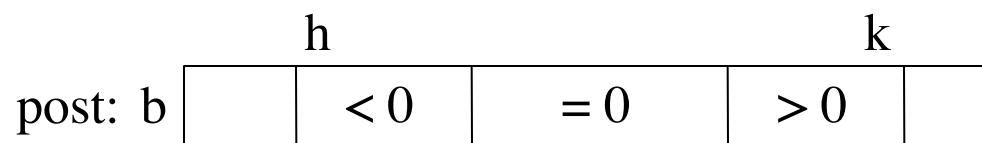
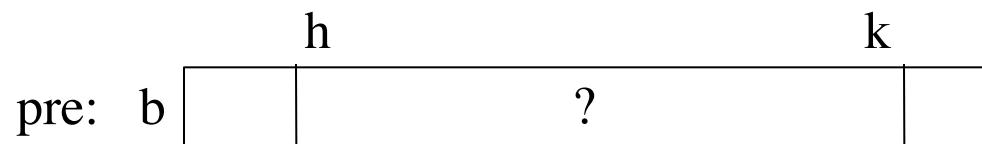
pre: b 

	h		k		
post: b	<input type="checkbox"/>	<input type="checkbox"/> < 0	<input type="checkbox"/> = 0	<input type="checkbox"/> > 0	<input type="checkbox"/>

	h	t	i	j	k	
inv:	b	< 0	= 0	?	> 0	

Changing the Invariant

- Different invariants = different code
 - Need to change how we initialize, stop
 - Also need to change the body of the loop



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

Changing the Invariant

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

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

Changing the Invariant

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

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

Changing the Invariant

```
def dnf(b, h, k):
    """Returns: partition points as a tuple (i,j)"""
    t = h; i = h, j = k;
    # inv: b[h..t-1] < 0, b[i..t-1] = 0, b[t..j] ?, b[j+1..k] > 0
    while t < j+1:
        if b[t] < 0:
            ???
        elif b[t] == 0:
            ???
        else:
            swap(b,t,j)
            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	i	t	j	k
-1 -2	0 0	3 -1 0	6 3	

h	i	t	j	k
-1	-2	0 0	0 -1	3 6 3
←				

Changing the Invariant

```
def dnf(b, h, k):
    """Returns: partition points as a tuple (i,j)"""
    t = h; i = h, j = k;
    # inv: b[h..t-1] < 0, b[i..t-1] = 0, b[t..j] ?, b[j+1..k] > 0
    while t < j+1:
        if b[t] < 0:
            ???
        elif b[t] == 0:
            t = t+1
        else:
            swap(b,t,j)
            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	i	t	j	k
-1 -2	0 0	3 -1 0	6 3	

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

h	i	\rightarrow t/j	k
-1 -2	0 0 0	-1	3 6 3

Changing the Invariant

```

def dnf(b, h, k):
    """Returns: partition points as a tuple (i,j)"""
    t = h; i = h, j = k;
    # inv: b[h..t-1] < 0, b[i..t-1] = 0, b[t..j] ?, b[j+1..k] > 0
    while t < j+1:
        if b[t] < 0:
            swap(b,t,i)
            i = i+1; t = t+1;
        elif b[t] == 0:
            t = t+1
        else:
            swap(b,t,j)
            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	i	t	j	k
-1 -2	0 0	3 -1 0	6 3	

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

h	i	$\rightarrow t/j$	k
-1 -2	0 0 0	-1	3 6 3

h	$\rightarrow i$	$j \rightarrow t$	k
-1 -2 -1	0 0 0	3 6 3	

Changing the Invariant

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



```
def dnf(b, h, k):  
    """Returns: partition points"""  
    t = h; i = k+1, j = k;  
    # b[h..t-1] <, b[t..i-1] ?, b[i..j] =, b[j+1..k] >  
    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)  
            j = j-1  
    # b[h..i-1] <, b[i..j] =, b[j+1..k] >  
    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	
--	---------	----------	---------	----------	--

 h k

inv: b

	< 0, o	< 0, e	≥ 0, o	?	≥ 0, e	
--	--------	--------	--------	---	--------	--

 h r s i t k



Flag of Mauritius

$< 0, o$	$< 0, e$	$\geq 0, o$	$?$	$\geq 0, e$
h	r	s	i	t k
-1 -3	-2 -4	7 5	-5 -6 1 0	2 4

h	r	s	i	t	k
-1	-3	-5 -4	7 5	-2 -6 1 0	2 4
					

One swap is not good enough

Flag of Mauritius

$< 0, o$	$< 0, e$	$\geq 0, o$	$?$	$\geq 0, e$
h	r	s	i	t k
-1 -3	-2 -4	7 5	-5 -6 1 0	2 4

h	r	s	i	t	k
-1	-3	-5 -4	-2 5	7 -6 1 0	2 4
					

Need two swaps
for two spaces

Flag of Mauritius

$< 0, o$	$< 0, e$	$\geq 0, o$	i	$?$	$\geq 0, e$
h	r	s			t k
-1 -3	-2 -4	7 5	-5 -6 1 0	2 4	

h	\rightarrow	r	\rightarrow	s	\rightarrow	i		t	k
-1 -3	$\textcolor{red}{-5}$	-4	$\textcolor{blue}{-2}$	5	$\textcolor{blue}{7}$	-6	1 0	2 4	



And adjust the
loop variables

Flag of Mauritius

$< 0, o$	$< 0, e$	$?$	$\geq 0, e$
h	r=s	i	t k
-1 -3 -7	-4 -2 -6	-5 1 0	2 4

h	r=s	i	t	k
-1 -3 -7	-5 -2 -6	-4 1 0	2 4	

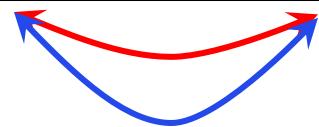


BUT NOT
ALWAYS!

Flag of Mauritius

$< 0, o$	$< 0, e$	$?$	$\geq 0, e$
h	r=s	i	t k
-1 -3 -7	-4 -2 -6	-5 1 0	2 4

h	r=s	i	t	k
-1 -3 -7	-4 -2 -6	-5 1 0	2 4	



BUT NOT
ALWAYS!

Have to check if second swap is okay

Flag of Mauritius

$< 0, o$	$< 0, e$	$\geq 0, o$	$?$	$\geq 0, e$
h	r	s	i	t k
-1 -3	-2 -4	7 5	-5 -6 1 0	2 4

h	\rightarrow	r	\rightarrow	s	\rightarrow	i		t	k
-1 -3	$\textcolor{red}{-5}$	-4	$\textcolor{blue}{-2}$	5	$\textcolor{blue}{7}$	-6	1 0	2	4

See algorithms.py
for Python code

Flag of Mauritius

$< 0, o$	$< 0, e$	$\geq 0, o$	$?$	$\geq 0, e$
h	r	s	i	t k
-1 -3	-2 -4	7 5	-5 -6 1 0	2 4

h $\xrightarrow{}$ r $\xrightarrow{}$ s $\xrightarrow{}$ i $\quad t \quad k$

-1	-3	-5	-4	-2	5	7	-6	1	0	2	4
----	----	-----------	----	-----------	---	----------	----	---	---	---	---

h $\quad r \quad \xrightarrow{}$ s $\xrightarrow{}$ i $\quad t \quad k$

-1	-3	-5	-4	-2	-6	7	5	1	0	2	4
----	----	----	----	----	-----------	---	----------	---	---	---	---

See algorithms.py
for Python code

Flag of Mauritius

$< 0, o$	$< 0, e$	$\geq 0, o$	$?$	$\geq 0, e$
h	r	s	i	t k
-1 -3	-2 -4	7 5	-5 -6 1 0	2 4

h \xrightarrow{r} \xrightarrow{s} \xrightarrow{i} t k

-1	-3	-5	-4	-2	5	7	-6	1	0	2	4
----	----	-----------	----	-----------	---	----------	----	---	---	---	---

h r \xrightarrow{s} \xrightarrow{i} t k

-1	-3	-5	-4	-2	-6	7	5	1	0	2	4
----	----	----	----	----	-----------	---	----------	---	---	---	---

h r s \xrightarrow{i} t k

-1	-3	-5	-4	-2	-6	7	5	1	0	2	4
----	----	----	----	----	-----------	---	----------	---	---	---	---

See algorithms.py
for Python code

Extras Not Covered in Class

Loaded Dice

- Sequence p of length n represents n -sided die
 - Contents of p sum to 1
 - $p[k]$ is probability die rolls the number k

1	2	3	4	5	6
0.1	0.1	0.1	0.1	0.3	0.3

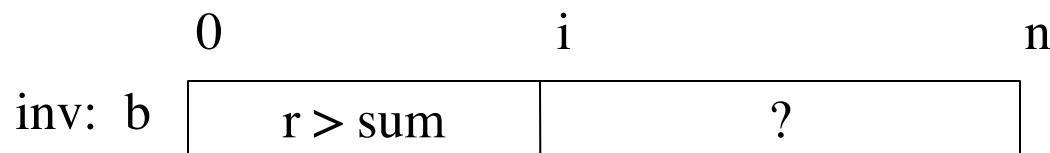
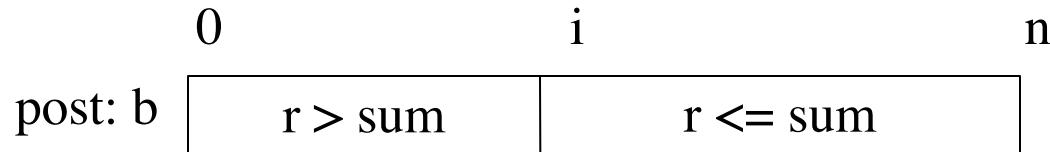
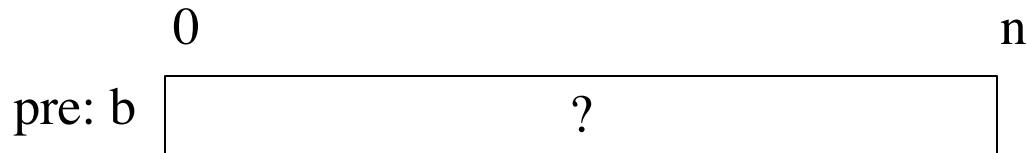
weighted d6, favoring 5, 6

- Goal: Want to “roll the die”
 - Generate random number r between 0 and 1
 - Pick $p[i]$ such that $p[i-1] < r \leq p[i]$

0.1	0.1	0.1	0.1	0.3	0.3
0.1	0.2	0.3	0.4	0.7	1.0

Loaded Dice

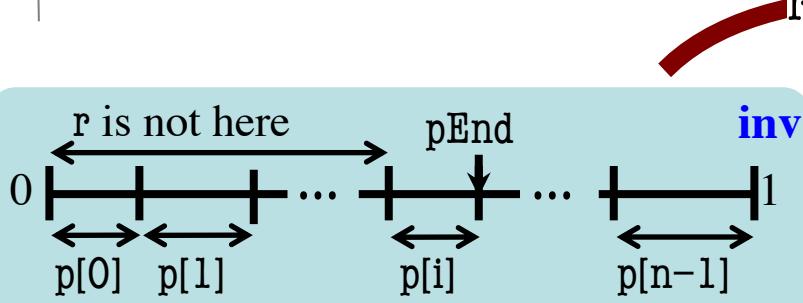
- **Want:** Value i such that $p[i-1] < r \leq p[i]$



- Same as precondition if $i = 0$
- Postcondition is invariant + false loop condition

Loaded Dice

```
def roll(p):
    """Returns: randint in 0..len(p)-1; i returned with prob. p[i]
    Precondition: p list of positive floats that sum to 1."""
    r = random.random() # r in [0,1)
    # Think of interval [0,1] divided into segments of size p[i]
    # Store into i the segment number in which r falls.
    i = 0; sum_of = p[0]
    # inv: r >= sum of p[0] .. p[i-1]; pEnd = sum of p[0] .. p[i]
    while r >= sum_of:
        sum_of = sum_of + p[i+1]
        i = i + 1
    # post: sum of p[0] .. p[i-1] <= r < sum of p[0] .. p[i]
    return i
```



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?

Reversing a Sequence

pre: b not reversed

post: b reversed h k

change: b h k

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

	h	i	j	k
inv:	b	swapped	not reversed	swapped

