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 $b[h]$:

  ![Diagram]

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

    ![Diagram]

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

Remember, slicing always copies the list!

We want to partition the original list
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

1 2 3 1 5 0 6 3 8

11/27/18
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
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
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
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 | h | t | i | j | k
| < 0 | = 0 | > 0
```

```
inv: b | h | t | i | j | k
| < 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

\[
\begin{array}{c|c|c|c}
& h & k \\
\hline
\text{pre: } b & ? \\
\hline
\text{post: } b & <0 & =0 & >0 \\
\hline
\text{inv: } b & <0 & ? & =0 & >0 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c}
& h & t & i & j & k \\
\hline
\text{pre: } t = h, & & & & & \\
& i = k+1, & & & & \\
& j = k & & & & \\
\hline
\text{post: } t = i
\end{array}
\]
Dutch National Flag Algorithm

```python
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)
```

<table>
<thead>
<tr>
<th></th>
<th>&lt; 0</th>
<th>?</th>
<th>= 0</th>
<th>&gt; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>-1</td>
<td>3</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>t</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

11/27/18
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)
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)
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)
Changing the Invariant

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

<table>
<thead>
<tr>
<th>pre: b</th>
<th>h</th>
<th>?</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>post: b</td>
<td>h</td>
<td>&lt;0</td>
<td>=0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>inv: b</th>
<th>h</th>
<th>t</th>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;0</td>
<td>=0</td>
<td>?</td>
<td>&gt;0</td>
<td></td>
</tr>
</tbody>
</table>
Changing the Invariant

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

<table>
<thead>
<tr>
<th>pre</th>
<th>b</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>post</td>
<td>b</td>
<td>&lt;0</td>
</tr>
<tr>
<td>inv</td>
<td>b</td>
<td>&lt;0</td>
</tr>
<tr>
<td>pre</td>
<td>t = h, i = h, j = k</td>
<td></td>
</tr>
<tr>
<td>post</td>
<td>t = j+1</td>
<td></td>
</tr>
</tbody>
</table>
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)
Changing the Invariant

```python
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)
```

---

<table>
<thead>
<tr>
<th>&lt; 0</th>
<th>= 0</th>
<th>?</th>
<th>&gt; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>i</td>
<td>t</td>
<td>j</td>
</tr>
<tr>
<td>-1</td>
<td>-2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

11/27/18  Sequences (Continued) 19
Changing the Invariant

```python
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)
```

<table>
<thead>
<tr>
<th></th>
<th>&lt; 0</th>
<th>= 0</th>
<th>?</th>
<th>&gt; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>1</td>
<td>-2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>j</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>k</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>

11/27/18
Sequences (Continued)
Changing the Invariant

```python
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)
```

### Table

<table>
<thead>
<tr>
<th></th>
<th>&lt; 0</th>
<th>= 0</th>
<th>?</th>
<th>&gt; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>-2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>j</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>k</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Diagram

```
-1 0 3 6 3
  
-1 -2 0 0 -1 3 6 3
  
-1 -2 0 0 -1 3 6 3
```
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)
def dnf(b, h, k):
    """Returns: partition points"""
    t = h; i = h, j = k;
    # b[h..t-1] <, b[t..i-1] =, b[i..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)
            i = i-1; 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
Flag of Mauritius

<table>
<thead>
<tr>
<th>&lt;0, o</th>
<th>&lt;0, e</th>
<th>≥ 0, o</th>
<th>?</th>
<th>≥ 0, e</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>r</td>
<td>s</td>
<td>i</td>
<td>t</td>
</tr>
<tr>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>-4</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td>-5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

One swap is not good enough
Flag of Mauritius

<table>
<thead>
<tr>
<th>h</th>
<th>r</th>
<th>s</th>
<th>i</th>
<th>t</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>-4</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>-5</td>
<td>-6</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Need two swaps for two spaces
Flag of Mauritius

<table>
<thead>
<tr>
<th>&lt; 0, o</th>
<th>&lt; 0, e</th>
<th>≥ 0, o</th>
<th>?</th>
<th>≥ 0, e</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>r</td>
<td>s</td>
<td>i</td>
<td>t</td>
</tr>
<tr>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>-4</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

And adjust the loop variables
Flag of Mauritius

<table>
<thead>
<tr>
<th>&lt;0, o</th>
<th>&lt;0, e</th>
<th>?</th>
<th>≥0, e</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>r=s</td>
<td>i</td>
<td>t</td>
</tr>
<tr>
<td>-1</td>
<td>-4</td>
<td>-5</td>
<td>1</td>
</tr>
<tr>
<td>-3</td>
<td>-2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>-7</td>
<td>-6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BUT NOT ALWAYS!
Flag of Mauritius

<table>
<thead>
<tr>
<th>&lt;0, o</th>
<th>&lt;0, e</th>
<th>?</th>
<th>≥0, e</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>r=s</td>
<td>i</td>
<td>t</td>
</tr>
<tr>
<td>-1</td>
<td>-3</td>
<td>-7</td>
<td>-5</td>
</tr>
<tr>
<td>-4</td>
<td>-2</td>
<td>-6</td>
<td>1</td>
</tr>
<tr>
<td>-5</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Have to check if second swap is okay

BUT NOT ALWAYS!
Flag of Mauritius

<table>
<thead>
<tr>
<th>h</th>
<th>r</th>
<th>s</th>
<th>i</th>
<th>t</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>-4</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>-5</td>
<td>-6</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

See algorithms.py for Python code
### Flag of Mauritius

<table>
<thead>
<tr>
<th>&lt;0, o</th>
<th>&lt;0, e</th>
<th>≥0, o</th>
<th>?</th>
<th>≥0, e</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>r</td>
<td>s</td>
<td>i</td>
<td>t</td>
</tr>
<tr>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>-4</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

See `algorithms.py` for Python code.
Flag of Mauritius

<table>
<thead>
<tr>
<th>&lt;0, o</th>
<th>&lt;0, e</th>
<th>≥0, o</th>
<th>?</th>
<th>≥0, e</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>r</td>
<td>s</td>
<td>i</td>
<td>t</td>
</tr>
<tr>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>-4</td>
<td>7</td>
</tr>
<tr>
<td>-5</td>
<td>-6</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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


<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
</tr>
</tbody>
</table>

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


<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>0.4</td>
<td>0.7</td>
<td>1.0</td>
</tr>
</tbody>
</table>

weighted d6, favoring 5, 6
**Loaded Dice**

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

  $\begin{array}{ccc}
  0 & \cdots & n \\
  \text{pre: b} & \text{?} \\
  0 & i & n \\
  \text{post: b} & \text{r > sum} & \text{r <= sum}
  \end{array}$

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

<table>
<thead>
<tr>
<th>pre:</th>
<th>b</th>
<th>not reversed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>post:</td>
<td>b</td>
<td>reversed</td>
</tr>
<tr>
<td></td>
<td>h</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>change:</th>
<th>1 2 3 4 5 6 7 8 9 9 9 9</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>h</td>
<td>k</td>
</tr>
<tr>
<td>into</td>
<td>9 9 9 9 8 7 6 5 4 3 2 1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>inv:</th>
<th>b</th>
<th>swapped</th>
<th>not reversed</th>
<th>swapped</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>i</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>j</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>k</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>