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 17\textsuperscript{th} 9-11:30am**
  - 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: 89.5  Median: 93
  - Std Dev: 12.5
  - Mean: 15 hr  Median: 15 hr
  - Std Dev: 7 hr
  - SEVERAL AI hearings

- **A7** is due **Tuesday Dec. 10**
  - Extensions are possible
  - Contact your lab instructor
Recall: Horizontal Notation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>k</th>
<th>len(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>&lt;= sorted</td>
<td>&gt;=</td>
<td></td>
</tr>
</tbody>
</table>

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 $b[h]$:  

<table>
<thead>
<tr>
<th>h</th>
<th>x</th>
<th>?</th>
<th>k</th>
</tr>
</thead>
</table>

pre: $b$

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

<table>
<thead>
<tr>
<th>h</th>
<th>i</th>
<th>i+1</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= x</td>
<td>x</td>
<td>&gt;= x</td>
<td></td>
</tr>
</tbody>
</table>

post: $b$

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

12/3/19

Sequences (Continued)
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 Algorithm Implementation**

```python
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
```

```
<table>
<thead>
<tr>
<th>&lt;= x</th>
<th>x</th>
<th>?</th>
<th>&gt;= x</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>i</td>
<td>i+1</td>
<td>j</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
```

```
h       i           j   k
1        2           1   3
1        2           1   3
1        2           1   3
```

```
h       i       j       k
1        2       1       3
      i       1       3
      1       2       3
```

```
h       i       j       k
1        2       1       3
      i       j       k
    i
```

12/3/19
Sequences (Continued)
**Partition Algorithm Implementation**

```python
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
```

12/3/19

Sequences (Continued)
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

\[
\begin{align*}
\text{pre: } & b \quad h \quad ? \quad k \\
\text{post: } & b \quad < 0 \quad = 0 \quad > 0 \\
\text{inv: } & b \quad < 0 \quad ? \quad = 0 \quad > 0
\end{align*}
\]

\[
\begin{align*}
\text{pre: } & t = h, \\
& i = k + 1, \\
& j = k \\
\text{post: } & t = i
\end{align*}
\]
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>h</th>
<th>t</th>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-2</td>
<td>3</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>6</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
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)
def dnf(b, h, k):
    """Returns: partition points as a tuple (i,j)"""
    t = h; i = k+1, j = k;
    # \text{inv}: b[h..t-1] < 0, b[t..i-1] \geq 0, 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
    # \text{post}: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    return (i, j)
**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:

<table>
<thead>
<tr>
<th>h</th>
<th>t</th>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-2</td>
<td>3</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

Sequence:

```
-1 -2 3 -1 0 0 0 6 3
```

Sequence (Continued):

```
-1 -2 -1 3 0 0 0 6 3
```

```
-1 -2 -1 0 0 0 3 6 3
```
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>
<tr>
<td>inv: b</td>
<td>h</td>
<td>t</td>
<td>i</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

```
pre:  b  ?  

post: b  < 0  = 0  > 0

inv:  b  < 0  = 0  ?  > 0
```

```
pre:  t = h, 
i = h, 
j = k
post: t = j+1
```
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] ?i, 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)
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></th>
<th>&lt; 0</th>
<th>= 0</th>
<th>?</th>
<th>&gt; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>-2</td>
<td>0</td>
<td>3</td>
<td>6</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>0</td>
<td>0</td>
</tr>
<tr>
<td>j</td>
<td>6</td>
<td>19</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>k</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</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[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)
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)
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)
Flag of Mauritius

Now we have four colors!

- Negatives: ‘red’ = odd, ‘purple’ = even
- Positives: ‘yellow’ = odd, ‘green’ = even

<table>
<thead>
<tr>
<th>h</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>h</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0 odd</td>
<td>&lt; 0 even</td>
</tr>
</tbody>
</table>

<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>&lt; 0, o</td>
<td>&lt; 0, e</td>
<td>≥ 0, o</td>
<td>?</td>
<td>≥ 0, e</td>
<td></td>
</tr>
</tbody>
</table>
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>
</tbody>
</table>

One swap is not good enough
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 5</td>
</tr>
<tr>
<td>-5</td>
<td>-6</td>
<td>1 0</td>
<td></td>
<td>2 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>-5</td>
<td>-6</td>
<td>1</td>
<td>0</td>
<td>2</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></td>
<td></td>
<td></td>
<td>k</td>
</tr>
<tr>
<td>-1 -3 -7</td>
<td>-4 -2 -6</td>
<td>-5 1 0</td>
<td>2 4</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>-4</td>
</tr>
<tr>
<td>-5</td>
<td>1</td>
<td>0</td>
<td>2 4</td>
</tr>
</tbody>
</table>

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

BUT NOT ALWAYS!

Have to check if second swap is okay
### 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>
</tbody>
</table>

See algorithms.py for Python code

12/3/19

Sequences (Continued)
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 k</td>
</tr>
<tr>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>-4</td>
<td>7 5</td>
</tr>
<tr>
<td>-5</td>
<td>-6</td>
<td>1</td>
<td>0</td>
<td>2 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
### 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></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<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>
  
  Weighted d6, favoring 5, 6

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<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>
<tr>
<td></td>
<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>
**Loaded Dice**

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

  
<table>
<thead>
<tr>
<th>pre: $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$?$</td>
</tr>
</tbody>
</table>

  
<table>
<thead>
<tr>
<th>post: $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r &gt; \text{sum}$</td>
</tr>
</tbody>
</table>

- **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 \texttt{inv} true?
2. Is \texttt{post} true when \texttt{inv} is true and \texttt{condition} is false?
3. Does the repetend make progress?
4. Does the repetend keep \texttt{inv} true?

\begin{center}
\begin{tikzpicture}[scale=0.8]
  \node (r) at (0,0) {r is not here};
  \node (p0) at (1,-1) {p[0]};
  \node (p1) at (2,-1) {p[1]};
  \node (pi) at (3,-1) {p[i]};
  \node (pEnd) at (4,-1) {pEnd};
  \node (inv) at (5,-1) {inv};
  \node (l) at (6,-1) {l};
  \node (post) at (7,-1) {post};

  \draw [->] (r) -- (p0);
  \draw [->] (p0) -- (p1);
  \draw [->] (p1) -- (pi);
  \draw [->] (pi) -- (pEnd);
  \draw [->] (pEnd) -- (inv);
  \draw [->] (inv) -- (l);
  \draw [->] (l) -- (post);

  \draw [->] (0,0) -- (1,-1) node [midway,above] {1}
  \draw [->] (1,-1) -- (2,-1) node [midway,above] {2}
  \draw [->] (2,-1) -- (3,-1) node [midway,above] {3}
  \draw [->] (3,-1) -- (4,-1) node [midway,above] {i}
  \draw [->] (4,-1) -- (5,-1) node [midway,above] {i}
  \draw [->] (5,-1) -- (6,-1) node [midway,above] {n-1}
  \draw [->] (6,-1) -- (7,-1) node [midway,above] {n-1}

  \end{tikzpicture}
\end{center}
### Reversing a Sequence

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

<table>
<thead>
<tr>
<th>change:</th>
<th>b</th>
<th>12345678999</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>k</td>
<td></td>
</tr>
<tr>
<td>into</td>
<td>b</td>
<td>999987654321</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>h</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>j</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

12/3/19

Sequences (Continued)