Lecture 26

Sequence Algorithms (Continued)
Announcements for This Lecture

Assignment & Lab

- A6 is not graded yet
  - Done early next week
- A7 due **Fri, Dec. 11**
  - Friday after classes
  - Milestone not adjusted
  - Is your paddle moving?
- Lab Today: Office Hours
  - Get help on A7 paddle
  - Anyone can go to any lab

Next Week

- Last Week of Class!
  - Finish sorting algorithms
  - Special final lecture
- Lab held, but is optional
  - More invariant practice
  - Also use lab time on A7
- Details about the exam
  - Multiple review sessions
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.
Partition Algorithm

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

pre:  
\[ h \] 
\[ x \] 
\[ \_ \] 
\[ ? \] 

post:  
\[ h \] 
\[ i \] 
\[ i+1 \] 
\[ k \] 
\[ <= x \] 
\[ x \] 
\[ >= x \]

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

inv:  
\[ h \] 
\[ i \] 
\[ j \] 
\[ k \] 
\[ <= x \] 
\[ x \] 
\[ ? \] 
\[ >= x \]

• 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
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>6</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
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

11/24/15

1 2 3 1 5 0 6 3 8
1 2 3
1 2 1 3 5 0 6 3 8
**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
```

---

11/24/15  Sequences (Continued) 9
**Dutch National Flag Variant**

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

\[
\begin{array}{c|c|c|c|c}
& b & \text{?} & & \\
\hline
\text{pre: } & h & k & & \\
\hline
\text{post: } & h & \text{< 0} & \text{= 0} & \text{> 0} & \\
\hline
\text{inv: } & h & t & i & j & k \\
\end{array}
\]
Dutch National Flag Variant

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

```
<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>t</th>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td>?</td>
<td>=0</td>
<td>&gt;0</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>h</th>
<th>t</th>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td>?</td>
<td>=0</td>
<td>&gt;0</td>
<td></td>
</tr>
</tbody>
</table>
```

```
| pre: t = h, |
| i = k+1, |
| j = k |

| post: t = i |
```
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] > 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
    # 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] > 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
    # 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)
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>7</td>
<td>5</td>
</tr>
<tr>
<td>-5</td>
<td>-6</td>
<td>1</td>
<td>0</td>
<td>4</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</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>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Need two swaps for two spaces

11/24/15

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</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>2 4</td>
<td></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>≥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>

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   -3</td>
<td>-2   -4</td>
<td>7   5</td>
<td>-5  -6 1 0</td>
<td>2  4</td>
</tr>
</tbody>
</table>

See algorithms.py for Python code
Linear Search

• **Vague**: Find first occurrence of $v$ in $b[h..k-1]$. 
Linear Search

- **Vague:** Find first occurrence of v in b[h..k-1].
- **Better:** Store an integer in i to truthify result condition post:

  post:  
  1. v is not in b[h..i-1]
  2. i = k  OR  v = b[i]
Linear Search

- **Vague**: Find first occurrence of \( v \) in \( b[h..k-1] \).
- **Better**: Store an integer in \( i \) to truthify result condition post:
  
  \[
  \text{post:} \quad 1. \ v \text{ is not in } b[h..i-1] \\
  2. \ i = k \ OR \ v = b[i]
  \]

\[
\begin{array}{c}
\text{pre: } b \\
h & ? & k \\
\end{array}
\]

\[
\begin{array}{c}
\text{post: } b \\
h & i & k \\
v \text{ not here} & v & ?
\end{array}
\]
Linear Search

- **Vague**: Find first occurrence of v in b[h..k-1].
- **Better**: Store an integer in i to truthify result condition post:
  
  post: 1. v is not in b[h..i-1]
  2. \( i = k \) OR \( v = b[i] \)

\[
\begin{array}{c|c|c}
\text{pre: b} & \_ & ? \\
\hline
\text{post: b} & \text{v not here} & v & ? \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
\text{h} & i & k \\
\hline
\text{b} & \_ & \text{v not here} \\
\end{array}
\]
## Linear Search

<table>
<thead>
<tr>
<th></th>
<th>h</th>
<th>i</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre: b</td>
<td>?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>post: b</td>
<td>v not here</td>
<td>v</td>
<td>?</td>
</tr>
<tr>
<td>OR</td>
<td>i</td>
<td>k</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>v not here</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>h</th>
<th>i</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>inv: b</td>
<td>v not here</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>
Linear Search

```python
def linear_search(b, c, 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 += 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

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 the invariant inv true?
Binary Search

• **Vague:** Look for \(v\) in sorted sequence segment \(b[h..k]\).
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:

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td></td>
<td>i</td>
<td>k</td>
</tr>
</tbody>
</table>
```

**pre:** $b$ is unordered

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>i</td>
</tr>
</tbody>
</table>
```

**post:** $b$ is ordered

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= v</td>
<td>&gt; v</td>
</tr>
</tbody>
</table>
```
# 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:**

<table>
<thead>
<tr>
<th>h</th>
<th>?</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pre:</strong> ( b )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>h</th>
<th>i</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>post:</strong> ( b )</td>
<td>( \leq v )</td>
<td>( &gt; v )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>h</th>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inv:</strong> ( b )</td>
<td>( &lt; v )</td>
<td>?</td>
<td>( &gt; v )</td>
</tr>
</tbody>
</table>

Called *binary search* because each iteration of the loop cuts the array segment still to be processed in half.
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></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 ≤ 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>p</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>

11/24/15  Sequences (Continued)
Loaded Dice

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

  
  \[
  \begin{array}{c|c|c|c}
  0 & \cdots & n \\
  \hline
  \text{pre: b} & ? & ? & ? \\
  \hline
  0 & \cdots & i & \cdots & n \\
  \hline
  \text{post: b} & r > \text{sum} & r \leq \text{sum} \\
  \end{array}
  \]

- **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: b</th>
<th>not reversed</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td></td>
</tr>
<tr>
<td>post: b</td>
<td>reversed</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>h</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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