Review 7

Sequence Algorithms
Three Types of Questions

• Write body of a loop to satisfy a given invariant.
  - Problem 6, Fall 2013 (Final)
  - Problem 6, Spring 2014 (Final)

• Given an invariant with code, identify all errors.
  - Problem 6, Spring 2014 (Prelim 2)
  - Problem 6, Spring 2013 (Final)

• Given an example, rewrite it with new invariant.
  - Problem 8, Fall 2014 (Final)
Horizontal Notation for Sequences

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.
• **DON’T** put variables directly above vertical line.

```
<table>
<thead>
<tr>
<th>h</th>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>&lt;= x</td>
<td>x</td>
<td>?</td>
</tr>
</tbody>
</table>
```

- Where is j?
- Is it unknown or >= x?
Algorithm Inputs

• We may specify that the list in the algorithm is
  ▪ $b[0..\text{len}(b)-1]$ or
  ▪ a segment $b[h..k]$ or
  ▪ a segment $b[m..n-1]$

• Work with whatever is given!

![Array segment diagram]

• Remember formula for # of values in an array segment
  ▪ Following – First
  ▪ e.g. the number of values in $b[h..k]$ is $k+1-h$. 
Three Types of Questions

• Write body of a loop to satisfy a given invariant.
  ▪ Problem 6, Fall 2013 (Final)
  ▪ Problem 6, Spring 2014 (Final)

• Given an invariant with code, identify all errors.
  ▪ Problem 6, Spring 2014 (Prelim 2)
  ▪ Problem 6, Spring 2013 (Final)

• Given an example, rewrite it with new invariant.
  ▪ Problem 8, Fall 2014 (Final)
Exercise 6, Fall 2013 Final

• Example:
  - Input [1, 2, 2, 2, 4, 4, 4]
  - Output [1, 2, 2, 2, 1, 2, 4]
# Solution to Fall 2013 Final

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>p</th>
<th>h</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inv:</code></td>
<td>b unchanged</td>
<td>Unchanged, values all in (b[h+1..k])</td>
<td>(b[p+1..k]) w/o duplicates</td>
<td></td>
</tr>
</tbody>
</table>

# Assume \(0 \leq k\), so the list segment has at least one element

\(p = \) \\
\(h = \)

# `inv:` \(b[h+1..k]\) is original \(b[p+1..k]\) with no duplicates

# \(b[p+1..h]\) is unchanged from original list w/ values in \(b[h+1..k]\)

# \(b[0..p]\) is unchanged from original list

```python
while :
```

```
# Assume 0 <= k, so the list segment has at least one element

\[ p = k - 1 \]

\[ h = k - 1 \]

# inv: \( b[h+1..k] \) is original \( b[p+1..k] \) with no duplicates

# \( b[p+1..h] \) is unchanged from original list w/ values in \( b[h+1..k] \)

# \( b[0..p] \) is unchanged from original list

while :
# Assume 0 <= k, so the list segment has at least one element

\[ p = k - 1 \]

\[ h = k - 1 \]

# inv: \( b[h+1..k] \) is original \( b[p+1..k] \) with no duplicates

# \( b[p+1..h] \) is unchanged from original list w/ values in \( b[h+1..k] \)

# \( b[0..p] \) is unchanged from original list

**while** 0 <= \( p \):

```
<table>
<thead>
<tr>
<th>inv: b unchanged</th>
<th>Unchanged, values all in b[h+1..k]</th>
<th>b[p+1..k] w/o duplicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>p</td>
<td>h</td>
</tr>
<tr>
<td></td>
<td></td>
<td>k</td>
</tr>
</tbody>
</table>
```

**while 0 <= p:**
# Assume 0 <= k, so the list segment has at least one element

\[
p = k - 1
\]

\[
h = k - 1
\]

# inv: b[h+1..k] is original b[p+1..k] with no duplicates

# b[p+1..h] is unchanged from original list w/ values in b[h+1..k]

# b[0..p] is unchanged from original list

while 0 <= p:
  if b[p] != b[p+1]:
    b[h] = b[p]
    h = h - 1
    p = p - 1
• **Example:**

  - Input \( s_1 = 'abracadabra' \), \( s_2 = 'abc' \)
  - Output \( 'abacaabardr' \) (or \( 'aaaabbcdrdr' \))
# convert to a list b

```python
b = list(s1)
```

# initialize counters

```python
while
```

# post: b[0..j] in s2; b[i+1..n-1] not in s2

# convert b back to a string
# convert to a list b
b = list(s1)

# initialize counters
i = 0
j = len(b) - 1

# inv: b[0..i-1] in s2; b[j+1..n-1] not in s2

while True:
    Inv:
    | 0 | i | j | len(b) |
    |---|---|---|--------|
    | Elts in s2 | ??? | Elts not in s2 |

# post: b[0..j] in s2; b[i+1..n-1] not in s2

# convert b back to a string
# convert to a list b
b = list(s1)

# initialize counters
i = 0
j = len(b) - 1

# inv: b[0..i-1] in s2; b[j+1..n-1] not in s2

while j != i - 1:

    # post: b[0..j] in s2; b[i+1..n-1] not in s2

# convert b back to a string

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>i</th>
<th>j</th>
<th>len(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inv:</strong></td>
<td>Elts in s2</td>
<td>???</td>
<td>Elts not in s2</td>
<td></td>
</tr>
</tbody>
</table>
Solution to Spring 2014 Final

```python
# convert to a list b
b = list(s1)

# initialize counters
i = 0
j = len(b) - 1

# inv: b[0..i-1] in s2; b[j+1..n-1] not in s2

while j != i - 1:
    if b[i] in s2:
        i = i + 1
    else:
        b[i], b[j] = b[j], b[i]  # Fancy swap syntax in python
        j = j - 1

# post: b[0..j] in s2; b[i+1..n-1] not in s2

# convert b back to a string
```
# convert to a list b
b = list(s1)

# initialize counters
i = 0
j = len(b) - 1

# inv: b[0..i-1] in s2; b[j+1..n-1] not in s2

while j != i - 1:
    if b[i] in s2:
        i = i + 1
    else:
        b[i], b[j] = b[j], b[i]  # Fancy swap syntax in python
        j = j - 1

# post: b[0..j] in s2; b[i+1..n-1] not in s2

# convert b back to a string
result = ''.join(b)
Three Types of Questions

• Write body of a loop to satisfy a given invariant.
  ▪ Problem 6, Fall 2013 (Final)
  ▪ Problem 6, Spring 2014 (Final)

• Given an invariant with code, identify all errors.
  ▪ Problem 6, Spring 2014 (Prelim 2)
  ▪ Problem 6, Spring 2013 (Final)

• Given an example, rewrite it with new invariant.
  ▪ Problem 8, Fall 2014 (Final)
def partition(b, z):
    i = 1
    k = len(b)

    # inv: b[0..i-1] <= z and b[k..] > z
    while i != k:
        if b[i] <= z:
            i = i + 1
        else:
            k = k - 1
            b[i], b[k] = b[k], b[i]  # python swap

    # post: b[0..k-1] <= z and b[k..] > z
    return k
def partition(b, z):
    i = 0
    k = len(b)

    # inv: b[0..i-1] <= z and b[k..] > z
    while i != k:
        if b[i] <= z:
            i = i + 1
        else:
            k = k - 1
            b[i], b[k] = b[k], b[i]  # python swap

    # post: b[0..k-1] <= z and b[k..] > z
    return k
def partition(b, z):
    i = -1
    k = len(b)
    # inv: b[0..i] <= z and b[k..] > z
    while i != k:
        if b[i+1] <= z:
            i = i + 1
        else:
            b[i+1], b[k-1] = b[k-1], b[i+1]  # python swap
            k = k - 1
    # post: b[0..k-1] <= z and b[k..] > z
    return k
```python
def partition(b, z):

    i = -1
    inv: b[0..i] <= z and b[k..] > z

    k = len(b)
    # inv: b[0..i] <= z and b[k..] > z

    while i != k:
        if b[i+1] <= z:
            i = i + 1
        else:
            b[i+1], b[k-1] = b[k-1], b[i+1]  # python swap
            k = k-1

    # post: b[0..k-1] <= z and b[k..] > z
    return k
```

def num_space_runs(s):
    """The number of runs of spaces in the string s.
    Examples: ' a f g ' is 4 'a f g' is 2 ' a bc d' is 3.
    Precondition: len(s) >= 1""
    i = 1
    n = 1 if s[0] == ' ' else 0
    # inv: s[0..i] contains n runs of spaces
    while i != len(s):
        if s[i] == ' ' and s[i-1] != ' ':
            n = n+1
        i = i+1
    # post: s[0..len(s)-1] contains n runs of spaces return n
    return n
def num_space_runs(s):
    """The number of runs of spaces in the string s.
    Examples: 'a f g' is 4 'a f g' is 2 'a bc d' is 3.
    Precondition: len(s) >= 1""
    i = 0
    n = 1 if s[0] == '' else 0
    # inv: s[0..i] contains n runs of spaces
    while i != len(s):
        if s[i] == '' and s[i-1] != ' ':
            n = n + 1
            i = i + 1
    # post: s[0..len(s)-1] contains n runs of spaces return n
    return n
def num_space_runs(s):
    """The number of runs of spaces in the string s.
    Examples: ' a f g ' is 4 'a f g' is 2 ' a bc d' is 3.
    Precondition: len(s) >= 1""

    i = 1     i = 0
    n = 1 if s[0] == ' ' else 0
    # inv: s[0..i] contains n runs of spaces

    while i != len(s):
        i != len(s)-1
        if s[i] == ' ' and s[i-1] != ' ':
            n = n+1
            i = i+1

    # post: s[0..len(s)-1] contains n runs of spaces
    return n
def num_space_runs(s):
    """The number of runs of spaces in the string s.
    Examples: ' a f g ' is 4 'a f g' is 2 ' a bc d' is 3.
    Precondition: len(s) >= 1""
    i = 1  # Inv: s[0..i] contains n runs of spaces
    n = 1 if s[0] == ' ' else 0
    while i != len(s):
        if s[i] == ' ' and s[i-1] != ' ':
            n = n+1
            i = i+1
    # post: s[0..len(s)-1] contains n runs of spaces return n
    return n
Three Types of Questions

- Write body of a loop to satisfy a given invariant.
  - Problem 6, Fall 2013 (Final)
  - Problem 6, Spring 2014 (Final)
- Given an invariant with code, identify all errors.
  - Problem 6, Spring 2014 (Prelim 2)
  - Problem 6, Spring 2013 (Final)
- Given an example, rewrite it with new invariant.
  - Problem 8, Fall 2014 (Final)
Partition Example

# Make invariant true at start
j = h
t = k + 1

# inv: b[h..j-1] <= x = b[j] <= b[t..k]
while j < t - 1:
    if b[j+1] <= b[j]:  
        swap b[j] and b[j+1]
        j = j + 1
    else:
        swap b[j+1] and b[t-1]
        t = t - 1

# post: b[h..j-1] <= x = b[j] <= b[j+1..k]

# Make invariant true at start
j = q

# inv: b[h..j-1] <= x = b[j] <= b[q+1..k]
while:
    # post: b[h..j-1] <= x = b[j] <= b[j+1..k]
Partition Example

# Make invariant true at start

```plaintext
j = h
t = k+1
# inv: b[h..j-1] <= x = b[j] <= b[t..k]
while j < t-1:
    if b[j+1] <= b[j]:
        swap b[j] and b[j+1]
        j = j+1
    else:
        swap b[j+1] and b[t-1]
        t = t-1
# post: b[h..j-1] <= x = b[j] <= b[j+1..k]
```

# Make invariant true at start

```plaintext
j =
q =
# inv: b[h..j-1] <= x = b[j] <= b[q+1..k]
while j < t-1:
    if b[j+1] <= b[j]:
        swap b[j] and b[j+1]
        j = j+1
    else:
        swap b[j+1] and b[t-1]
        t = t-1
# post: b[h..j-1] <= x = b[j] <= b[j+1..k]
```
# Make invariant true at start

\[ j = h \]
\[ t = k+1 \]

# inv: \( b[h..j-1] \leq x = b[j] \leq b[t..k] \)

while \( j < t-1 \):
    if \( b[j+1] \leq b[j] \):
        swap \( b[j] \) and \( b[j+1] \)
        \( j = j + 1 \)
    else:
        swap \( b[j+1] \) and \( b[t-1] \)
        \( t = t - 1 \)

# post: \( b[h..j-1] \leq x = b[j] \leq b[j+1..k] \)
Partition Example

# Make invariant true at start
j = h
t = k+1

# inv: b[h..j-1] <= x = b[j] <= b[t..k]
while j < t-1:
    if b[j+1] <= b[j]:
        swap b[j] and b[j+1]
        j = j+1
    else:
        swap b[j+1] and b[t-1]
        t = t-1

# post: b[h..j-1] <= x = b[j] <= b[j+1..k]

# Make invariant true at start
j = h
q = k

# inv: b[h..j-1] <= x = b[j] <= b[q+1..k]
while j < q:
    if b[j+1] <= b[j]:
        swap b[j] and b[j+1]
        j = j+1
    else:
        swap b[j+1] and b[q]
        q = q-1

# post: b[h..j-1] <= x = b[j] <= b[j+1..k]
Partition Example

# Make invariant true at start
j = h
\( t = k + 1 \)

# inv: \( b[h..j-1] \leq x = b[j] \leq b[t..k] \)

while \( j < t - 1 \):
  if \( b[j+1] \leq b[j] \):
    swap \( b[j] \) and \( b[j+1] \)
    \( j = j + 1 \)
  else:
    swap \( b[j+1] \) and \( b[t-1] \)
    \( t = t - 1 \)

# post: \( b[h..j-1] \leq x = b[j] \leq b[j+1..k] \)

---

# Make invariant true at start
j =
m =

# inv: \( b[h..j-1] \leq x = b[j] \leq b[j+1..m] \)

while :

# post: \( b[h..j-1] \leq x = b[j] \leq b[j+1..k] \)

---

inv: \( b \begin{array}{c} \leq \ x \ \\ \ \\ \ ?? \ \\ \ \\ \ \geq \ x \end{array} \)
### Partition Example

**# Make invariant true at start**

\[
j = h \\
t = k + 1
\]

**# inv:** \( b[h..j-1] \leq x = b[j] \leq b[t..k] \)

**while** \( j < t - 1 \):

- **if** \( b[j+1] \leq b[j] \):
  - **swap** \( b[j] \) and \( b[j+1] \)
  - \( j = j + 1 \)
- **else**:
  - **swap** \( b[j+1] \) and \( b[t-1] \)
  - \( t = t - 1 \)

**# post:** \( b[h..j-1] \leq x = b[j] \leq b[j+1..k] \)

---

**# Make invariant true at start**

\[
j = h \\
m = h
\]

**# inv:** \( b[h..j-1] \leq x = b[j] \leq b[j+1..m] \)

**while**

**# post:** \( b[h..j-1] \leq x = b[j] \leq b[j+1..k] \)

---

**inv:** \( b \begin{array}{c} 
\leq x \\
x \\
??? \\
>= x 
\end{array} \)

---

**inv:** \( b \begin{array}{c} 
\leq x \\
x \\
>= x \\
???
\end{array} \)
Partition Example

# Make invariant true at start
j = h
t = k+1

# inv: b[h..j-1] <= x = b[j] <= b[t..k]
while j < t-1:
  if b[j+1] <= b[j]:  
    swap b[j] and b[j+1]
    j = j+1
  else:
    swap b[j+1] and b[t-1]
    t = t-1

# post: b[h..j-1] <= x = b[j] <= b[j+1..k]

---

# Make invariant true at start
j = h
m = h

# inv: b[h..j-1] <= x = b[j] <= b[j+1..m]
while m < k:

# post: b[h..j-1] <= x = b[j] <= b[j+1..k]
# Make invariant true at start
\[ j = h \]
\[ t = k + 1 \]

# inv: \( b[h..j-1] \leq x = b[j] \leq b[t..k] \)

while \( j < t - 1 \):

  if \( b[j+1] \leq b[j] \):
    swap \( b[j] \) and \( b[j+1] \)
    \( j = j + 1 \)
  else:
    swap \( b[j+1] \) and \( b[t-1] \)
    \( t = t - 1 \)

# post: \( b[h..j-1] \leq x = b[j] \leq b[j+1..k] \)

---

# Make invariant true at start
\[ j = h \]
\[ m = h \]

# inv: \( b[h..j-1] \leq x = b[j] \leq b[j+1..m] \)

while \( m < k \):

  if \( b[m+1] \leq b[j] \):
    swap \( b[j] \) and \( b[m+1] \)
    swap \( b[j+1] \) and \( b[m+1] \)
    \( m = m + 1; j = j + 1 \)
  else:
    \( m = m + 1 \)

# post: \( b[h..j-1] \leq x = b[j] \leq b[j+1..k] \)
Questions?