Review 7

Sequence Algorithms
Three Types of Questions

• Write body of a loop to satisfy a given invariant.
  ▪ 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)
  ▪ Problem 7, Fall 2018 (Final)
Horizontal Notation for Sequences

Example of an assertion about a 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 ≤ everything in `b[k..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.
DOs and DON’Ts #3

- DON’T put variables directly above vertical line.

<table>
<thead>
<tr>
<th></th>
<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>
<td>&gt;= x</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!**

- 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, 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)
  - Problem 7, Fall 2018 (Final)
Exercise 6, Spring 2014 Final

- **Example:**

- Input $s_1 = 'abracadabra$', $s_2 = 'abc'$

- Output 'abacaabardr' (or 'aaaabbcrrdr')
# convert to a list b
b = list(s1)

# initialize counters

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

    # post: b[0..j] in s2; b[i+1..n-1] not in s2
    # convert b back to a string

    # invariants:
    Inv: 0 Elts in s2 ?? Elts not in s2
# 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

    Inv:  
    | 0 | i | j | len(b) |
    |---|---|---|--------|
    | Elts in s2 | ??? | Elts not in s2 |
# 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, 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)
  - Problem 7, Fall 2018 (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):

\[i = -1\]

\[k = \text{len}(b)\]

# inv: \(b[0..i] \leq z\) and \(b[k..] \geq z\)

while \(i \neq k\):

    if \(b[i+1] \leq z\):
        \(i = i + 1\)
    else:
        \(b[i+1], b[k-1] = b[k-1], b[i+1]\)  # python swap
        \(k = k-1\)

# post: \(b[0..k-1] \leq z\) and \(b[k..] \geq z\)

return \(k\)
def partition(b, z):

table:

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

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 = 1  # i = 0
    n = 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 = 1
    n = 1 if s[0] == ' ' else 0
    # inv: s[0..i] contains n runs of spaces
    while i != len(s):
        i = i + 1
        if s[i] == ' ' and s[i-1] != ' ':
            n = n + 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] != '': s[i+1] == '' and s[i] != '':
            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, 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)
  ▪ Problem 7, Fall 2018 (Final)
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 \(b[j+1] \leq b[j]\):
        \text{swap } b[j] \text{ and } b[j+1]
        j = j+1
    else:
        \text{swap } b[j+1] \text{ 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 = q
\(t = \) 

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

while : 

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

\[ h \quad j \quad t \quad k \]

\text{inv: } b \begin{array}{|c|c|c|}
\hline
\leq x & x & ??? & \geq x \\
\hline
\end{array} \]
# 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 =
\[ q = \]
# inv: \( b[h..j-1] \leq x = b[j] \leq b[q+1..k] \)

while :
  # 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:

# 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 = h
q = k

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

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

\[
\begin{align*}
j &= h \\
t &= k + 1
\end{align*}
\]

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

while \( j < t - 1 \):

\[
\begin{align*}
\text{if } b[j+1] &\leq b[j]: \\
&\quad \text{swap } b[j] \text{ and } b[j+1] \\
&\quad j = j + 1
\end{align*}
\]

\[
\begin{align*}
\text{else:} \\
&\quad \text{swap } b[j+1] \text{ and } b[t-1] \\
&\quad t = t - 1
\end{align*}
\]

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

---

# Make invariant true at start

\[
\begin{align*}
j &= m \\
m &= \text{m}
\end{align*}
\]

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

while \( j < t \):

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

---

\[
\begin{align*}
h &\quad j &\quad t &\quad k \\
\text{inv: } b &\quad \leq x &\quad x &\quad ??? &\quad \geq x
\end{align*}
\]
# 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: \( b[h..j-1] \leq x = b[j] \leq b[j+1..k] \)
# Make invariant true at start
\( j = h \)
\( t = k+1 \)

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

\textbf{while} \( j < t-1 \):
\hspace{1em}
\textbf{if} \( b[j+1] \leq b[j] \):
\hspace{2em} swap \( b[j] \) and \( b[j+1] \)
\hspace{2em} \( j = j+1 \)
\textbf{else}:
\hspace{2em} swap \( b[j+1] \) and \( b[t-1] \)
\hspace{2em} \( t = t-1 \)

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

\textbf{# Make invariant true at start}
\( j = h \)
\( m = h \)

\textbf{while} \( m < k \):
\hspace{1em}
\textbf{# inv:} \( b[h..j-1] \leq x = b[j] \leq b[j+1..m] \)

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

\begin{tabular}{|c|c|c|c|}
\hline
\textbf{inv:} & \( b \) & \( \leq x \) & \( x \) & \( ?? \) & \( \geq x \) \\
\hline
\end{tabular}

\begin{tabular}{|c|c|c|c|c|c|}
\hline
\textbf{inv:} & \( b \) & \( \leq x \) & \( x \) & \( \geq x \) & \( ?? \) \\
\hline
\end{tabular}
# Partition Example

## Implementation

```
# Make invariant true at start
j = h
m = 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
# inv: b[h..j-1] <= x = b[j] <= b[j+1..k]
while m < k:
    if b[m+1] <= 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] <= x = b[j] <= b[j+1..k]
```

### Invariants

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

<table>
<thead>
<tr>
<th>h</th>
<th>j</th>
<th>m</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= x</td>
<td>x</td>
<td>&gt;= x</td>
<td>??</td>
</tr>
</tbody>
</table>
What is Fair Game for this Question?

- Segregation (see Fall 2014 Final)
- Partition from **Lab 13**
- Dutch-National-Flag from **Lab 13**
- The non-recursive sorting algorithms
  - Insertion Sort (**Lecture 27**)
  - Selection Sort (**Lecture 27**)
  - But changing invariants changes helpers too
- Binary Search (**Lectures 26 & 27**)
Questions?