Lecture 21: Invariants, Quicksort, Partitioning

Upcoming schedule

Friday Apr 18: A4 due
Monday Apr 21: deadline for lab 11 checkoff
Tuesday Apr 22
• Lecture = (optional) review session
• Lab sessions = (optional) drop-in office hours. No new lab.
• Prelim: 7:30-9pm, 200 Baker Lab (= the auditorium)
  (No S/U students, so it should be less crowded)
Wednesday Apr 23: No labs or office hours

Prelim grades announced by email from CMS, hopefully by Thursday morning.
**Prelim 2 coverage:** Lectures 12-22 (this Thursday), associated labs and assignments: recursion, using classes effectively, loops and loop invariants, sequence algorithms.

**Preparing for the exam**
Past exams are posted on the Exams section of the website. Profs Lee and Marschner wrote the Spring 2013 one, and the topic coverage is equivalent.

Review all lectures up to and including April 17. Be able to do A3, A4, labs 8-11 from scratch, cold. Be able to do the worked invariants exercises yourself.

As always, come to our many office hours/consulting hours for in-person help; see the Staff section of the webpage.

As always, watch Piazza for announcements, for helpful answers to other people's questions, etc.
def ap1(s):
    """Returns: num of adjacent equal pairs in string s"""
    x = 0;  k = 0
    # k: next possible ending place of adjacent pair, i.e.
    # inv: x = # adjacent equal pairs in s[0..k-1]
    while k < len(s):
        x += (s[k-1] == s[k])
        k += 1
    return x

(A) Error in line 3  (B) line 6  (C) line 7  (D) line 8  (E) no error
def ap1(s):
    """Returns: num of adjacent equal pairs in string s"""
    x = 0;  k = 0
    # k: next possible ending place of adjacent pair, i.e.
    # inv: x = # adjacent equal pairs in s[0..k-1]
    while k < len(s):
        x += (s[k-1] == s[k])
        k += 1
    return x

(A) Error in line 3: the one-line fix is to change "k=0" to "k=1"
At most one line needs to be fixed

1. `def ap2(s):`
2. """Returns: num of adjacent equal pairs in string s""
3. `x = 0; j = 0`
4. `# j: next possible start place of adjacent pair, i.e.`
5. `#inv: x = # adjacent equal pairs in s[0..j]`
6. `while j < len(s)-1:`
   j += 1
7. `x += (s[j] == s[j+1])`
8. `return x`

(A) Error in line 3  (B) line 6  (C) line 7  (D) line 8  (E) no error
At most one line needs to be fixed

1. def ap2(s):
2. 
   """Returns: num of adjacent equal pairs in string s""
3. x = 0; j = 0
4. # j: next possible start place of adjacent pair, i.e.
5. # inv: x = # adjacent equal pairs in s[0..j]
6. while j < len(s)-1:
   j += 1
7. x += (s[j] == s[j+1])
8. return x

(C) line 7: one-line fix is "s[j] == s[j+1]" to "s[j-1]==s[j]"
Warning about Python indexing errors

Since negative indices are not a syntactic error in Python, it can be especially hard to debug indexing errors.

Example: conceptually, starting with k=0 and checking s[k] vs s[k-1] is an error because you're referring to a non-existent position "before the start of the string".

However, Python interprets s[-1] as meaning the item at the end of the string. Those aren't what the previous slides meant to compare!
Q: Given a list of items, how can we arrange for them to be sorted in increasing order, in a time- and space-efficient manner?
  Applications: making items easier to find.¹

```
def sort(b, h, k):
    """Sort  b[h..k] in place.  Pre: b: list of ints; k>=h-1""
    # Start with b[h], and organize the rest according to it?? No...
    # Note: we have h & k explicit to simplify recursive
    # structure.

    ¹Also, computing poker-hand probabilities.
Motivation: A Famous Sorting Algorithm

```python
def qsort(b, h, k):
    # Make b[h..k] sorted.
    # Pre: b: list of ints; k>=h-1"
    i = partition(b, h, k)

    # base case

    # Can you do this without creating extra lists?

    def partition(b, h, k):
        # Let x = b[h] be the pivot value. Rearrange b[h..k] so that there is an i where b[h..i-1] <= x, b[i]=x; b[i+1..k] >=x. Return i.
        Pre: k>=h"
```

Clicker Q1: recursive case
Pictorial Notation for Sequence Assertions

\[
\begin{array}{c|c|c}
\hline
0 & h & k \\
\hline
b & \text{some property } p & \text{some property } q \\
\hline
\end{array}
\]

Equivalent to:

*Property p holds on all items in \( b[0..h-1] \), and property q holds on all items in \( b[h..k] \).*

(The precise location of the "vertical bars" matters.)

Can also indicate single items.

\[((h +1) – h = 1; it's all consistent, hurrah.)\]
Partition Algorithm

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

  ![Precondition Table]

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

  ![Postcondition Table]

- $x$ is called the **pivot value**

  - $x$ is not a program variable, but a standin for a number: the value initially in $b[h]$
def qsort(b, h, k):
    """Make b[h..k] sorted.
    Pre: b: list of ints; k>=h-1""
    # base case
    if k < h:  # empty is sorted
        return
    i = partition(b, h, k)

Clicker Q1: recursive case

(A) qsort(b,h,i-1)
    qsort (b, i+1, k)
(B) qsort(b,h,i-1)
    qsort(b,h,i+1,k)
(C) qsort (b,h,k)
(D) qsort(b,h,i+1)
    qsort(b,i+2,k)
(E) more than one works
def qsort(b, h, k):
    """Make b[h..k] sorted.
    Pre: b: list of ints; k>=h-1""

    # base case
    if k < h:  # empty is sorted
        return

    i = partition(b, h, k)

    # recursive case
    qsort(b,h,i-1)
    qsort(b,i+1,k)
An Invariant to Guide Our Thinking

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

\[
\begin{array}{|c|c|}
\hline
\text{h} & \text{k} \\
\hline
\text{pre: } b & \text{x} \\
\hline
\text{post: } b & \leq x \quad x \quad \geq x \\
\hline
\end{array}
\]

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

\[
\begin{array}{|c|c|c|}
\hline
\text{h} & \text{i} & \text{i+1} \\
\hline
\text{post: } b & \leq x & x & \geq x \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{inv: } b & \leq x & x & ? & \geq x \\
\hline
\end{array}
\]

- 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]; return index of pivot point
    Pre: k>=h""

    what goes here?
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x, b[i+1..j-1] unknown
    while what goes here?
        if b[i+1] >= x:
            # Move to end of block.
            b[i+1], b[j-1] = b[j-1], b[i+1]
            j = j - 1
        else:  # b[i+1] < x
            CLICKER Q3
            # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x
    return i
(A) \( b[i-1], b[j-1] = b[j-1], b[i-1] \)
\[ j = j - 1 \]

(B) \( b[i+1], b[j-1] = b[j-1], b[i-1] \)

(C) \( b[i+1], b[i] = b[i], b[i+1] \)

(D) \( b[i+1], b[i] = b[i], b[i+1] \)
\[ i += 1 \]
```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.
            b[i+1], b[j-1] = b[j-1], b[i+1]
            j = j - 1
        else:
            # b[i+1] < x
            b[i], b[i+1] = b[i+1], b[i]
            i = i + 1
    return i
```

```
# Example usage
b = [1, 2, 3, 5, 0, 6, 3, 8]
partition(b, 0, 7)
```

```
<table>
<thead>
<tr>
<th>h</th>
<th>i</th>
<th>i+1</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
```

```
<= x | x | ? | >= x
```

```
<table>
<thead>
<tr>
<th>h</th>
<th>i</th>
<th>i+1</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
```

```
<= x | x | ? | >= x
```

```
<table>
<thead>
<tr>
<th>h</th>
<th>i</th>
<th>i+1</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
```