CS1110

Lecture 20: Sequence algorithms

Upcoming schedule
Today (April 4) A6 out: A4 due tomorrow. Fix to memotable printing posted; see Piazza @303.
Tu Apr 9: lecture on searching & sorting – last material on the exam
  Probably a new lab exercise, for prelim exercise
Th Apr 11: lecture = review session
Sat Apr 13: A6 due (yes, we cancelled A5!).
Tu Apr 16: lecture = office hours, in Thurston 102
  Exam that evening, same location as before
  Probably no new lab exercise that week
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??
    # Note: we have h & k explicit to simplify recursive structure.

¹Also, computing poker-hand scores.
Motivation: A Famous Sorting Function

```python
def qsort(b, h, k):
    """Make b[h..k] sorted.
    Pre: b: list of ints; k>=h-1""
    i = partition(b, h, k)
    qsort(b, 0, i-1)
    qsort(b, i+1, k)

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""
    # Can you do this without creating extra lists?
    Clicker Q2: base case
    Clicker Q1: recursive case
```

# Can you do this **without** creating extra lists?
**Pictorial Notation for Sequence Assertions**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>h</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>some property p</td>
<td>some property q</td>
<td></td>
</tr>
</tbody>
</table>

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.

$$((h +1) – h = 1; \text{ it's all consistent, hurrah.})$$
Partition Algorithm

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

  
  \[
  \begin{array}{c|c|c|c|c|c|c|c|c|c|}
  \hline
  \text{pre: } b & x & \ldots & \text{?} \\
  \hline
  \end{array}
  \]

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

  
  \[
  \begin{array}{c|c|c|c|c|c|c|c|c|c|}
  \hline
  \text{post: } b & \text{h} & i & i+1 & k & \text{pre: } b & \text{?} \\
  \hline
  \end{array}
  \]

  \[
  \begin{array}{c|c|c|c|c|c|c|c|c|c|}
  \hline
  \text{change: } b & 3 & 5 & 4 & 1 & 6 & 2 & 3 & 8 & 1 \\
  \hline
  \end{array}
  \]

  \[
  \begin{array}{c|c|c|c|c|c|c|c|c|c|}
  \hline
  \text{into } b & 1 & 2 & 1 & 3 & 5 & 4 & 6 & 3 & 8 \\
  \hline
  \end{array}
  \]

  \[
  \begin{array}{c|c|c|c|c|c|c|c|c|c|}
  \hline
  \text{or } b & 1 & 2 & 3 & 1 & 3 & 4 & 5 & 6 & 8 \\
  \hline
  \end{array}
  \]

  or...

- $x$ is called the pivot value

- $x$ is not a program variable, but a standin for a number: value initially in $b[h]$
Motivation: A Famous Sorting Function

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

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""
    # Can you do this in place,
    # i.e., w/out
    # creating extra lists?
```

Clicker Q2: base case
if k < h:  # empty is sorted
    return

Clicker Q1: recursive case
An Invariant to Guide Our Thinking

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

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

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

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

- 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. Assume a swap function swap(b,ind1, ind2).
    Pre: k>=h""
    CLICKER Q5
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x, b[i+1..j-1] unknown
    while CLICKER Q4
        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
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
    # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x
    return i
Developing Algorithms on Sequences

- Specify the algorithm by giving its **precondition** and **postcondition** as pictures.
- Draw the **invariant** by drawing another picture that “generalizes” the **precondition** and **postcondition**
  - The invariant is true at the beginning and at the end
- The four loop design questions (**memorize them**)
  1. How does loop start (how to make the invariant true)?
  2. How does it stop (is the postcondition true)?
  3. How does repetend make progress toward termination?
  4. How does repetend keep the invariant true?
**Famous "Sort-Like" Example**

- Dutch national flag: tri-color
  - Sequence of $h..k$ of red ($<0$), white ($=0$), blue ($>0$) "pixels"
  - Arrange to put $<0$ first, then $=0$, then $>0$, return "split pts"

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

**pre:**

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

**post:**

<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>

**inv:**

<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>

$b[h..t-1] <0$, $b[t..i-1]$ unknown, $b[i..j] =0$, $b[j+1..k] >0$
def dnf(b, h, k):
    """(DNF explanation omitted for space.)
    Returns: split-points as a tuple (i,j)"

    # init?
    # 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:
            # what?

        elif b[i-1] == 0:
            # what?

        else:
            # what?

    # 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:
            b[i-1], b[t] = b[t], b[i-1]
            t = t+1
        elif b[i-1] == 0:
            i = i-1
        else:
            b[-1], b[j] = b[j], b[i-1]
            i = i-1; j = j-1
    # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    return (i, j)