Lecture 27

Sorting
Announcements for This Lecture

Finishing Up

• Submit a course evaluation
  ▪ Will get an e-mail for this
  ▪ Part of “participation grade”

• Final: Dec 17th 9-11:30am
  ▪ Study guide is posted
  ▪ Announce reviews on Tues.

• Conflict with Final time?
  ▪ Submit to conflict to CMS by next Tuesday!

Assignment 7

• Should be on bolt collisions
• Use weekend for final touches
  ▪ Multiple lives
  ▪ Winning or losing the game
• Also work on the extension
  ▪ Add anything you want
  ▪ ONLY NEED ONE
• Ask on Piazza if unsure
• All else is extra credit
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:
  
  **post:**
  1. \( v \) is not in \( b[h..i-1] \)
  2. \( i = k \) OR \( v = b[i] \)

<table>
<thead>
<tr>
<th>h</th>
<th>i</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre: ( 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>post: ( b )</td>
<td>v not here</td>
<td>v</td>
</tr>
</tbody>
</table>
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. $i = k \ OR \ v = b[i]$

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

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

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

pre: b

h

? 

i

k

post: b

v not here

v

? 

OR

i

h

k

b

v not here

inv: b

v not here

? 


Linear Search

```python
def linear_search(b, v, h, k):
    
    """Returns: first occurrence of v in b[h..k-1]""

    # Store in i index of the first v in b[h..k-1]
    i = h

    # invariant: v is not in b[h..i-1]
    while i < k and b[i] != v:
        i = i + 1

    # post: v is not in b[h..i-1]
    # i >= k or b[i] == v
    return i if i < k else -1
```

Analyzing the Loop

1. Does the initialization make \texttt{inv} true?
2. Is \texttt{post} true when \texttt{inv} is true and \texttt{condition} is false?
3. Does the repetend make progress?
4. Does the repetend keep the invariant \texttt{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:** $b[h..i-1] < v$ and $v \leq b[i..k]$

- Below, the array is in non-descending order:

<table>
<thead>
<tr>
<th></th>
<th>h</th>
<th>i</th>
<th>k</th>
</tr>
</thead>
</table>
  **pre:** $b$ | $<$ | v | $\geq$ | v |

12/5/19

Sorting
Binary Search

- Look for value \( v \) in sorted segment \( b[h..k] \)

<table>
<thead>
<tr>
<th></th>
<th>( h )</th>
<th>( k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre:</td>
<td>( b )</td>
<td>?</td>
</tr>
<tr>
<td>post:</td>
<td>( b )</td>
<td>(&lt; v)</td>
</tr>
<tr>
<td>inv:</td>
<td>( b )</td>
<td>(&lt; v)</td>
</tr>
</tbody>
</table>

New statement of the invariant guarantees that we get leftmost position of \( v \) if found

- if \( v \) is 3, set \( i \) to 0
- if \( v \) is 4, set \( i \) to 5
- if \( v \) is 5, set \( i \) to 7
- if \( v \) is 8, set \( i \) to 10

Example: \( b \)
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:** \( b[h..i-1] < v \) and \( v \leq b[i..k] \)
- Below, the array is in non-descending order:

  ![Diagram]

  Called **binary search** because each iteration of the loop cuts the array segment still to be processed in half.
Binary Search

pre:  b

post: b

inv:  b

i = h; j = k+1;

while i != j:

Looking at b[i] gives linear search from left.
Looking at b[j-1] gives linear search from right.
Looking at middle: b[(i+j)/2] gives binary search.

New statement of the invariant guarantees that we get leftmost position of v if found
Sorting: Arranging in Ascending Order

Insertion Sort:

\[ i = 0 \]

while \( i < n \):

# Push \( b[i] \) down into its
# sorted position in \( b[0..i] \)

\[ i = i + 1 \]
Insertion Sort: Moving into Position

i = 0

while i < n:
    push_down(b, i)
    i = i + 1

def push_down(b, i):
    j = i
    while j > 0:
        if b[j-1] > b[j]:
            swap(b, j-1, j)
            j = j - 1

0
2  4  4  6  6  7

5

swap shown in the lecture about lists
Insertion Sort: Moving into Position

\[
i = 0
\]
\[
\text{while } i < n:
\]
\[
\qquad \text{push\_down}(b, i)
\]
\[
\qquad i = i + 1
\]
\[
\text{def push\_down}(b, i):
\]
\[
\qquad j = i
\]
\[
\qquad \text{while } j > 0:
\]
\[
\qquad \qquad \text{if } b[j-1] > b[j]:
\]
\[
\qquad \qquad \quad \text{swap}(b, j-1, j)
\]
\[
\qquad \quad j = j - 1
\]

12/5/19  Sorting  16
Insertion Sort: Moving into Position

\[ i = 0 \]

\[ \text{while } i < n: \]

\[ \quad \text{push\_down}(b, i) \]

\[ \quad i = i + 1 \]

\def push\_down(b, i):

\[ j = i \]

\[ \quad \text{while } j > 0: \]

\[ \quad \text{if } b[j-1] > b[j]: \]

\[ \quad \quad \text{swap}(b, j-1, j) \]

\[ \quad j = j - 1 \]
Insertion Sort: Moving into Position

i = 0

while i < n:
    push_down(b, i)
    i = i + 1

def push_down(b, i):
    j = i
    while j > 0:
        if b[j-1] > b[j]:
            swap(b, j-1, j)
        j = j - 1

0  i
2 4 4 6 6 7 5

12/5/19
The Importance of Helper Functions

```
i = 0
while i < n:
    push_down(b,i)
    i = i+1

def push_down(b, i):
    j = i
    while j > 0:
        if b[j-1] > b[j]:
            swap(b,j-1,j)
        j = j-1
        i = i + 1
```

```
i = 0
while i < n:
    j = i
    while j > 0:
        if b[j-1] > b[j]:
            temp = b[j]
            b[j] = b[j-1]
            b[j-1] = temp
            j = j - 1
        i = i + 1
```
def push_down(b, i):
    """Push value at position i into sorted position in b[0..i-1]"""
    j = i
    while j > 0:
        if b[j-1] > b[j]:
            swap(b, j-1, j)
        j = j-1

• b[0..i-1]: i elements

• Worst case:
  - i = 0: 0 swaps
  - i = 1: 1 swap
  - i = 2: 2 swaps

• Pushdown is in a loop
  - Called for i in 0..n
  - i swaps each time

Total Swaps: 0 + 1 + 2 + 3 + … (n-1) = (n-1)*n/2 = (n^2-n)/2
def push_down(b, i):
    """Push value at position i into sorted position in b[0..i-1]""
    j = i
    while j > 0:
        if b[j-1] > b[j]:
            swap(b, j-1, j)
        j = j - 1

• b[0..i-1]: i elements

• Worst case:
  ▪ i = 0: 0 swaps
  ▪ i = 1: 1 swap
  ▪ i = 2: 2 swaps

• Pushdown is in a loop
  ▪ Called for i in 0..n
  ▪ i swaps each time

Insertion sort is an n^2 algorithm

Total Swaps: 0 + 1 + 2 + 3 + … (n-1) = (n-1)*n/2 = (n^2-n)/2
Algorithm “Complexity”

• **Given**: a list of length $n$ and a problem to solve
• **Complexity**: *rough* number of steps to solve worst case
• Suppose we can compute 1000 operations a second:

<table>
<thead>
<tr>
<th>Complexity</th>
<th>$n=10$</th>
<th>$n=100$</th>
<th>$n=1000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>0.01 s</td>
<td>0.1 s</td>
<td>1 s</td>
</tr>
<tr>
<td>$n \log n$</td>
<td>0.016 s</td>
<td>0.32 s</td>
<td>4.79 s</td>
</tr>
<tr>
<td>$n^2$</td>
<td>0.1 s</td>
<td>10 s</td>
<td>16.7 m</td>
</tr>
<tr>
<td>$n^3$</td>
<td>1 s</td>
<td>16.7 m</td>
<td>11.6 d</td>
</tr>
<tr>
<td>$2^n$</td>
<td>1 s</td>
<td>4x10$^{19}$ y</td>
<td>3x10$^{290}$ y</td>
</tr>
</tbody>
</table>

**Major Topic in 2110**: Beyond scope of this course
Sorting: Changing the Invariant

pre: \( b \) sorted
post: \( b \) sorted

Selection Sort:

pre: \( b \) ?
post: \( b \) sorted

inv: \( b \) sorted, \( \leq b[i..] \) \( \geq b[0..i-1] \)

\[ i = 0 \]

while \( i < n \):

\# Find minimum in \( b[i..] \)

\# Move it to position \( i \)

\[ i = i + 1 \]

First segment always contains smaller values
Sorting: Changing the Invariant

pre: \( b \) sorted

post: \( b \) sorted

Selection Sort:

\[
\begin{align*}
\text{inv: } & b \text{ sorted, } \leq b[i..] \geq b[0..i-1] \\
\text{i = 0} \\
\text{while } i < n: \\
& \text{# Find minimum in } b[i..] \\
& \text{# Move it to position } i \\
& i = i + 1
\end{align*}
\]

First segment always contains smaller values

Compared to insertion sort, selection sort is

A: Slower
B: About the same
C: Faster
D: I don’t know
**Sorting: Changing the Invariant**

**Selection Sort:**

- **Pre:** $b \text{?}$
- **Post:** $b \text{sorted}$

**Inv:** $b \leq b[i..] \geq b[0..i-1]$  

- $i = 0$
- **while** $i < n$:  
  - $j = \text{index of min of } b[i..n-1]$  
  - $\text{swap}(b,i,j)$  
  - $i = i + 1$

First segment always contains smaller values

Selection sort also is an $n^2$ algorithm

This is $n$ steps

12/5/19

Sorting 25
Partition Algorithm

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

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

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

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

  \[
  \begin{array}{|c|c|c|}
  \hline
  h & i & k \\
  \hline
  \end{array}
  \]

change: \quad \begin{array}{|c|c|c|c|c|c|c|c|c|}
  \hline
  3 & 5 & 4 & 1 & 6 & 2 & 3 & 8 & 1 \\
  \hline
  \end{array}

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

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

• x is called the pivot value

  - x is not a program variable
  - denotes value initially in b[h]
Sorting with Partitions

• Given a list segment \(b[h..k]\) with some value \(x\) in \(b[h]\):

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

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

\[
\begin{array}{c}
\text{post: } b & \leq y & y & \geq y & x & \geq x \\
\end{array}
\]

Recursive partitions = sorting

- Called \textbf{QuickSort} (why???)
- Popular, fast sorting technique
def quick_sort(b, h, k):
    """Sort the array fragment b[h..k]""
    if b[h..k] has fewer than 2 elements:
        return
    j = partition(b, h, k)
    # b[h..j-1] <= b[j] <= b[j+1..k]
    # Sort b[h..j-1] and b[j+1..k]
    quick_sort(b, h, j-1)
    quick_sort(b, j+1, k)

• **Worst Case:**
  - array already sorted
  - Or almost sorted
  - $n^2$ in that case

• **Average Case:**
  - array is scrambled
  - $n \log n$ in that case
  - Best sorting time!
Final Word About Algorithms

- **Algorithm:**
  - Step-by-step way to do something
  - Not tied to specific language

- **Implementation:**
  - An algorithm in a specific language
  - Many times, not the “hard part”

- **Higher Level Computer Science courses:**
  - We teach advanced algorithms (pictures)
  - Implementation you learn on your own