Lecture 25

Sorting
## Announcements for This Lecture

### Prelim/Finals
- Prelims in **handback room**
  - Upson Hall 305
  - Open “business hours”
  - Get them any day this week
- **Final: Dec 16th 9:00-11:30a**
  - Study guide by end of week
- **Conflict with Final time?**
  - Submit to Final Conflict assignment on CMS
  - **Must be in by December 9th**

### Assignments/Lab
- **A5** will be graded by Thurs.
  - Will give grade breakdown
  - Will review survey too
- **A6** is due next **Monday**
  - One week left
  - Keep up with deadlines
- **Lab 12** is the last lab
  - Due before final exam
  - Consultant hours still open

---

11/20/13  
Sequence Algorithms  2
**Binary Search**

- Look for value v in *sorted* segment b[h..k]

<table>
<thead>
<tr>
<th>pre:</th>
<th>b</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>i</td>
<td>k</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>post:</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>i</td>
</tr>
<tr>
<td>j</td>
<td>k</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>inv:</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>&lt; v</td>
</tr>
<tr>
<td>i</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>&gt;= v</td>
</tr>
</tbody>
</table>

| Example b | 3 3 3 3 3 4 4 6 7 7 |

- 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

New statement of the invariant guarantees that we get *leftmost* position of v if found.
**Binary Search**

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

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

\[
\text{post: } \begin{array}{c} b \\ h & < v & >= v & k \end{array}
\]

\[
\text{inv: } \begin{array}{c} b \\ h & < v & ? & >= v \end{array}
\]

\[i = h; \ j = k+1;\]

\[
\text{while } i \neq j:
\]

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.**
**Sorting: Arranging in Ascending Order**

---

<table>
<thead>
<tr>
<th>pre:</th>
<th>post:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b)</td>
<td>(\text{sorted})</td>
</tr>
</tbody>
</table>

---

**Insertion Sort:**

\[
i = 0 \\
\text{while } i < n: \\
\quad \# \text{ Push } b[i] \text{ down into its} \\
\quad \# \text{ sorted position in } b[0..i] \\
\quad i = i + 1
\]
**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

swap shown in the lecture about lists
The Importance of Helper Functions

```python
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
```

Can you understand all this code below?

```python
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
```

11/13/12
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

Insertion sort is an $n^2$ algorithm
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

Selection Sort:

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: \( b \) \( \Rightarrow ? \) \( b \) \( \Rightarrow \) sorted

post: \( b \) \( \Rightarrow \) sorted

**Selection Sort:**

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

\( i = 0 \)

while \( i < n \):

\( j = \) index of min of \( b[i..n-1] \)

swap\((b, i, j)\)

\( i = i + 1 \)

First segment always contains smaller values

Selection sort also is an \( n^2 \) algorithm

\[ 2 \ 4 \ 4 \ 6 \ 6 \ 8 \ 9 \ 9 \ 7 \ 8 \ 9 \]

\[ 2 \ 4 \ 4 \ 6 \ 6 \ 7 \ 9 \ 9 \ 8 \ 8 \ 9 \]

11/13/12

Sorting
Partition Algorithm

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

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

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

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

change:

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

into

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

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

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

$$
\begin{array}{c|c|c|c|c}
  h & i & i+1 & k \\
  \text{pre: } b & x & ? & \\
  \text{post: } b & \leq y & y & \geq y & x & \geq x \\
\end{array}
$$

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

Partition Recursively

Recursive partitions = sorting
- Called **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