Lecture 25

Searching & Sorting
Announcements for This Lecture

Prelim 2

- Prelim, Tonight at 7:30
  - A–L in Bailey 101
  - M–Z in Uris G01
- Material up to Nov. 8
  - Recursion + Loops + Classes
- Graded this Weekend
  - Grades posted on Monday
  - Need time for make-ups

Assignments

- A6 still not graded
  - Will be done by after break
  - Staff needs to take their time
- A7 is due Monday Dec. 5
  - Extensions are possible
  - Contact your lab instructor
def linear_search(v, b):
    """Returns: first occurrence of v in b (-1 if not found)"
    Precond: b a list of number, v a number
    """
    # Loop variable
    i = 0
    while i < len(b) and b[i] != v:
        i = i + 1
    if i == len(b):  # not found
        return -1
    return i
def linear_search(v, b):
    """Returns: first occurrence of v in b (-1 if not found)
    Precond: b a list of number, v a number
    """
    # Loop variable
    i = 0
    while i < len(b) and b[i] != v:
        i = i + 1
    if i == len(b):  # not found
        return -1
    return i

How many entries do we have to look at?
All of them!
def linear_search(v, b):
    """Returns: last occurrence of v in b (-1 if not found)
    Precond: b a list of number, v a number
    """
    # Loop variable
    i = len(b) - 1
    while i >= 0 and b[i] != v:
        i = i - 1
    # Equals -1 if not found
    return i

How many entries do we have to look at?

All of them!
Is There a Better Way?

- Thinking of number 0..100
  - You get to guess number
  - I tell you higher or lower
  - Continue until get it right

- **Goal:** Keep # guesses low
  - Use my answers to help

- **Strategy?**
  - Start guess in the middle
  - Answer eliminates half
  - Go to middle of remaining

0 50 100
Is There a Better Way?

Thinking of number 0..100
- You get to guess number
- I tell you higher or lower
- Continue until get it right

Goal: Keep # guesses low
- Use my answers to help

Strategy?
- Start guess in the middle
- Answer eliminates half
- Go to middle of remaining
Is There a Better Way?

- Thinking of number 0..100
  - You get to guess number
  - I tell you higher or lower
  - Continue until get it right

- **Goal:** Keep # guesses low
  - Use my answers to help

- **Strategy?**
  - Start guess in the middle
  - Answer eliminates half
  - Go to middle of remaining
Is There a Better Way?

- Thinking of number 0..100
  - You get to guess number
  - I tell you higher or lower
  - Continue until get it right

- **Goal:** Keep # guesses low
  - Use my answers to help

- **Strategy?**
  - Start guess in the middle
  - Answer eliminates half
  - Go to middle of remaining
Is There a Better Way?

• Thinking of number 0..100
  ▪ You get to guess number
  ▪ I tell you higher or lower
  ▪ Continue until get it right

• **Goal:** Keep # guesses low
  ▪ Use my answers to help

• **Strategy?**
  ▪ Start guess in the middle
  ▪ Answer eliminates half
  ▪ Go to middle of remaining

Correct!
def binary_search(v,b):
    # Loop variable(s)
    i = 0, j = len(b)
    while i < j and b[i] != v:
        mid = (i+j)//2
        if b[mid] < v:
            i = mid+1
        elif b[mid] > v:
            j = mid
        else:
            return mid
    return -1 # not found

Requires that the data is sorted!

But few checks!
Observation About Sorting

- Sorting data can speed up searching
  - Sorting takes time, but do it once
  - Afterwards, can search many times
- Not just searching. Also speeds up
  - Duplicate elimination in data sets
  - Data compression
  - Physics computations in computer games
- Why it is a major area of computer science
The Sorting Challenge

• **Given:** A list of numbers
• **Goal:** Sort those numbers using only
  - Iteration (while-loops or for-loops)
  - Comparisons (< or >)
  - Assignment statements

• **Why?** For proper **analysis**.
  - Methods/functions come with hidden costs
  - Everything above has no hidden costs
  - Each comparison or assignment is “1 step”
This Requires Some Notation

- As the list is sorted...
  - Part of the list *will* be sorted
  - Part of the list will *not* be sorted
- Need a way to refer to portions of the list
  - Notation to refer to sorted/unsorted parts
- And have to do it *without* slicing!
  - Slicing makes a *copy*
  - Want to sort original list, not a copy
This Requires Some Notation

• As the list is sorted…
  ▪ Part of the list **will** be sorted
  ▪ Part of the list will **not** be sorted

• Need a way to refer to portions of the list
  ▪ Notation to refer to sorted/unsorted parts

• And have to do it **without** slicing!
  ▪ Slicing makes a **copy**
  ▪ Want to sort original list, not a copy

But we will be less formal than in previous years!
Recall: Range Notation

- \( m..n \) is a range containing \( n+1-m \) values
  - \( 2..5 \) contains 2, 3, 4, 5. Contains 5+1 – 2 = 4 values
  - \( 2..4 \) contains 2, 3, 4. Contains 4+1 – 2 = 3 values
  - \( 2..3 \) contains 2, 3. Contains 3+1 – 2 = 2 values
  - \( 2..2 \) contains 2. Contains 2+1 – 2 = 1 values
  - \( 2..1 \) contains ???

- The notation \( m..n \), always implies that \( m \leq n+1 \)
  - So you can assume that even if we do not say it
  - If \( m = n+1 \), the range has 0 values
Recall: Range Notation

- **m..n** is a range containing **n+1-m** values
  - 2..5 contains 2, 3, 4, 5. Contains 5+1-2-1 = 4 values
  - 2..4 contains 2, 3, 4. Contains 4+1-2-1 = 3 values
  - 2..3 contains 2, 3. Contains 3+1-2-1 = 2 values
  - 2..2 contains 2. Contains 2+1-2-1 = 1 values
  - 2..1 contains ???

- The notation **m..n**, always implies that **m <= n+1**
  - So you can assume that even if we do not say it
  - If **m = n+1**, the range has 0 values

Not the same as range(m,n)
Horizontal Notation

- Want a pictoral way to visualize this sorting
  - Represent the list as long rectangle
  - We saw this idea in divide-and-conquer

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

- Do not show individual boxes
  - Just dividing lines between regions
  - Label dividing lines with indices
  - But index is either left or right of dividing line

\[(h+1) - h = 1\]
Horizontal Notation

- Label regions with properties
  - **Example:** Sorted or ???
    - $b[0..k-1]$ is sorted
    - $b[k..n-1]$ **unknown** (might be sorted)

- Picture allows us to track progress
**Visualizing Sorting**

**Start:**

\[
\begin{array}{c}
0 \text{ \quad } n \\
\text{b} \quad \text{?}
\end{array}
\]

**Goal:**

\[
\begin{array}{c}
0 \quad n \\
\text{b} \quad \text{sorted}
\end{array}
\]

**In-Progress:**

\[
\begin{array}{c}
0 \quad i \quad n \\
\text{b} \quad \text{sorted} \quad ?
\end{array}
\]
Insertion Sort

\[ i = 0 \]

while \( i < n \):

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

\( i = i + 1 \)

Remember the restrictions!
Insertion Sort: Moving into Position

\[ i = 0 \]

\[ \text{while } i < n: \]
\[ \quad \text{push\_down}(b,i) \]
\[ \quad i = i + 1 \]

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

\[ \begin{array}{|c|c|}
0 & i \\
\hline
2 & 4 & 4 & 6 & 6 & 7 & 5
\end{array} \]

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

\[ i = 0 \]

\[
\text{while } i < n:\n\quad \text{push\_down}(b,i)\\
\quad i = i + 1
\]

\[
\text{def push\_down}(b, i):\\
\quad j = i\\
\quad \text{while } j > 0:\n\quad \quad \text{if } b[j-1] > b[j]:\\
\quad \quad \quad \text{swap}(b, j-1, j)\\
\quad \quad j = j-1
\]
Insertion Sort: Moving into Position

```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
```

### Diagram:

1. Initial array: 0 2 4 4 6 6 7
2. After 1st pass: 0 2 4 4 6 6 7
3. After 2nd pass: 0 2 4 4 6 5 7
4. After 3rd pass: 0 2 4 4 5 6 7

Swap shown in the lecture about lists.

11/17/22
Searching & Sorting
## Insertion Sort: Moving into Position

\[ i = 0 \]

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

### def push_down(b, i):

\[ j = i \]

```python
while j > 0:
    if b[j-1] > b[j]:
        swap(b, j-1, j)
    j = j - 1
```

---

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

*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
```

VS

```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
```

Can you understand all this code below?

11/17/22

Searching & Sorting
Measuring Performance

• Performance is a tricky thing to measure
  ▪ Different computers run at different speeds
  ▪ Memory also has a major effect as well

• Need an independent way to measure
  ▪ Measure in terms of “basic steps”
  ▪ Example: Searching counted # of checks

• For sorting, we measure in terms of swaps
  ▪ Three assignment statements
  ▪ Present in all sorting algorithms
Insertion Sort: Performance

```python
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: \[ 0 + 1 + 2 + 3 + \ldots (n-1) = \frac{(n-1)\times n}{2} = \frac{n^2-n}{2} \]
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>log n</td>
<td>0.003 s</td>
<td>0.006 s</td>
<td>0.01 s</td>
</tr>
<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>n²</td>
<td>0.1 s</td>
<td>10 s</td>
<td>16.7 m</td>
</tr>
<tr>
<td>n³</td>
<td>1 s</td>
<td>16.7 m</td>
<td>11.6 d</td>
</tr>
<tr>
<td>2ⁿ</td>
<td>1 s</td>
<td>4x10¹⁹ y</td>
<td>3x10²⁹⁰ y</td>
</tr>
</tbody>
</table>
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>$\log n$</td>
<td>0.006 s</td>
<td>0.01 s</td>
<td></td>
</tr>
<tr>
<td>$n$</td>
<td>0.1 s</td>
<td>1 s</td>
<td></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>10 s</td>
<td>16.7 m</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>
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>log n</td>
<td>0.003 s</td>
<td>0.036 s</td>
<td>0.01 s</td>
</tr>
<tr>
<td>n</td>
<td>1 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n log n</td>
<td>4.79 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n²</td>
<td>16.7 m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n³</td>
<td>1 s</td>
<td>16.7 m</td>
<td>11.6 d</td>
</tr>
<tr>
<td>2ⁿ</td>
<td>1 s</td>
<td>4x10¹⁹ y</td>
<td>3x10²⁹⁰ y</td>
</tr>
</tbody>
</table>

Major Topic in 2110: Beyond scope of this course
Insertion Sort is Not Great

• Typically $n^2$ is okay, but not great
  ▪ Will perform horribly on large data
  ▪ Very bad when performance critical (games)

• We would like to do better than this
  ▪ Can we get $n$ swaps (no)?
  ▪ How about $n \log n$ (maybe)

• This will require a new algorithm
  ▪ Let’s return to horizontal notation
# A New Algorithm

**Start:**

<table>
<thead>
<tr>
<th>b</th>
<th>?</th>
</tr>
</thead>
</table>

**Goal:**

| b | sorted |

**In-Progress:**

| b | sorted, ≤ b[i..] | ≥ b[0..i-1] |

*First segment always contains smaller values*
Selection Sort

\[ i = 0 \]
\[ \text{while } i < n: \]
\[ \# \text{ Find minimum in } b[i..] \]
\[ \# \text{ Move it to position } i \]
\[ i = i + 1 \]

Remember the restrictions!
Selection Sort

How fast is this?

\[ i = 0 \]

while \( i < n \):

\[ j = \text{index of min of } b[i..n-1] \]

\[ \text{swap}(b, i, j) \]

\[ i = i + 1 \]
Selection Sort

This is also \( n^2 \)!

\[ i = 0 \]

\textbf{while} \( i < n \):  

\[ j = \text{index of min of } b[i..n-1] \]

swap(\( b, i, j \)) \[ \text{This is } n \text{ steps} \]

\[ i = i + 1 \]
What is the Problem?

• Both insertion, selection sort are nested loops
  ▪ Outer loop over each element to sort
  ▪ Inner loop to put next element in place
  ▪ Each loop is n steps. \( n \times n = n^2 \)

• To do better we must eliminate a loop
  ▪ But how do we do that?
  ▪ What is like a loop? Recursion!
  ▪ Will see how to do this next lecture