Lecture 26

Searching & Sorting
Announcements for This Lecture

**Lab/Finals**
- **Lab 26** is the final lab
  - Can use Consulting hours
  - Due **next Wednesday 9:30**
- **Final: Dec 12\textsuperscript{th} 2-4:30 pm**
  - Study guide is posted
  - Announce reviews next week.
- **Conflict with Final time?**
  - Submit to conflict to CMS by **next TUESDAY**!

**Assignments**
- **A6** still not graded
  - Will be done by Thursday
  - **SEVERAL** AI hearings
- **A7** is due **Tuesday Dec. 7**
  - Extensions are possible
  - Contact your lab instructor

11/30/21

Searching & Sorting
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
    """

    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?
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
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

Higher!
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 75 100

Lower!
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

Higher!
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
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 \textbf{will} be sorted
  ▪ Part of the list will \textbf{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 \textbf{without} slicing!
  ▪ Slicing makes a \textbf{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 = 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 <= n+1
  - So you can assume that even if we do not say it
  - If m = n+1, the range has 0 values
Horizontal Notation

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

- 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

\[ h \quad h+1 \]

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

- Label regions with properties
  - **Example:** Sorted or ???

  ![Diagram](image)

  - $b[0..k-1]$ is sorted
  - $b[k..n-1]$ **unknown** (might be sorted)

- Picture allows us to track progress
Visualizing Sorting

Start: 0 \[\text{b}\] n

Goal: 0 \[\text{b}\] sorted

In-Progress: 0 \[\text{b}\] sorted i ? n
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

\[ i = 0 \]

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

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

\[ \begin{array}{cccccc}
0 & 2 & 4 & 4 & 6 & 6 & 7 \\
\end{array} \]

\[ \text{swap shown in the lecture about lists} \]
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
Insertion Sort: Moving into Position

\[ i = 0 \]

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

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

swap shown in the lecture about lists

0  2  4  4  6  6  7  5

0  2  4  4  6  6  5  7

2  4  4  6  5  6  7

11/30/21  Searching & Sorting
Insertion Sort: Moving into Position

\[ i = 0 \]

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

\[ \text{def push\_down}(b, i): \]
\[ \quad j = i \]
\[ \quad \text{while } j > 0: \]
\[ \quad \quad \text{if } b[j-1] > b[j]: \]
\[ \quad \quad \quad \text{swap}(b,j-1,j) \]
\[ \quad \quad j = j - 1 \]
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 = 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?
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:** $0 + 1 + 2 + 3 + \ldots + (n-1) = \frac{(n-1) \times n}{2} = \frac{n^2 - n}{2}$
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: \(0 + 1 + 2 + 3 + \ldots (n-1) = \frac{(n-1)*n}{2} = \frac{n^2-n}{2}\)
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>( 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>4 \times 10^{19} \text{ y}</td>
<td>3 \times 10^{290} \text{ 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>0.016 s</td>
<td>0.32 s</td>
<td>4.79 s</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>1 s</td>
<td>16.7 m</td>
<td>11.6 d</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>$4 \times 10^{19}$ y</td>
<td>$3 \times 10^{290}$ y</td>
</tr>
</tbody>
</table>

*Binary Search*  
*Linear Search*  
*Insertion Sort*
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></td>
<td></td>
<td>1 s</td>
</tr>
<tr>
<td>( n \log n )</td>
<td></td>
<td></td>
<td>4.79 s</td>
</tr>
<tr>
<td>( n^2 )</td>
<td></td>
<td></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>4 \times 10^{19} y</td>
<td>3 \times 10^{290} 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: b

Goal: b

In-Progress: b

First segment always contains smaller values
Selection Sort

Selection Sort

\[
\begin{array}{ccc}
0 & i & n \\
\text{b} & \text{sorted, } \leq b[i..] & \geq b[0..i-1] \\
\end{array}
\]

\[
i = 0
\]

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

\[
\text{# Find minimum in } b[i..]
\]

\[
\text{# Move it to position } i
\]

\[
i = i + 1
\]

Remember the restrictions!
Selection Sort

How fast is this?

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

This is also \( n^2 \)!

\[ i = 0 \]

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

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

\textbf{swap}(b,i,j)

\( i = i + 1 \)
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