Algorithm Problem Solving (APS): Divide-and-Conquer

Niema Moshiri
UC San Diego SPIS 2019
What is an algorithm?
Goals of Algorithm Problem Solving (APS)
Goals of Algorithm Problem Solving (APS)

- Introduction to basic *algorithmic strategies* for solving problems
Goals of Algorithm Problem Solving (APS)

- Introduction to basic algorithmic strategies for solving problems
- Emphasis on writing solutions precisely and coherently
Goals of Algorithm Problem Solving (APS)

- Introduction to basic *algorithmic strategies* for solving problems
- Emphasis on writing solutions *precisely* and *coherently*
- Practice *discovering* algorithms and *describing* them
Goals of Algorithm Problem Solving (APS)

- Introduction to basic **algorithmic strategies** for solving problems
- Emphasis on writing solutions **precisely** and **coherently**
- Practice **discovering** algorithms and **describing** them
- **Analyze** algorithms
Example: The Largest Integer Problem
Example: The Largest Integer Problem

- **Input:** A list of integers \textit{ints}
Example: The Largest Integer Problem

- **Input:** A list of integers \textit{ints}

\begin{center}
\begin{tabular}{|c|c|c|c|c|}
\hline
7 & 25 & 0 & 42 & -9 \\
\hline
\end{tabular}
\end{center}
Example: The Largest Integer Problem

- **Input:** A list of integers *ints*
- **Output:** An integer $x$ in *ints* such that, for all integers $y$ in *ints*, $x \geq y$

| 7 | 25 | 0 | 42 | -9 |
Example: The Largest Integer Problem

- **Input:** A list of integers $ints$
- **Output:** An integer $x$ in $ints$ such that, for all integers $y$ in $ints$, $x \geq y$
  - In other words, $x$ is a largest integer in $ints$

<table>
<thead>
<tr>
<th>7</th>
<th>25</th>
<th>0</th>
<th>42</th>
<th>-9</th>
</tr>
</thead>
</table>
Example: The Largest Integer Problem

- **Input**: A list of integers \( ints \)

- **Output**: An integer \( x \) in \( ints \) such that, for all integers \( y \) in \( ints \), \( x \geq y \)
  - In other words, \( x \) is a largest integer in \( ints \)

\[
7 \quad 2 \quad 0 \quad 4 \quad -9 \quad 5 \quad 1 \quad -4 \quad 3 \quad 8 \quad -2 \quad -7 \quad ... \quad -1 \quad -8 \quad 6 \quad -3 \quad -6 \quad 9 \quad -5 \quad 2
\]
Easier Example: The Peanut Butter & Jelly Problem
Easier Example: The Peanut Butter & Jelly Problem

- **Input:** A closed jar of peanut butter *jar_pb*, a closed jar of jelly *jar_jelly*, a closed bag of toast *bag_toast*, and a knife *knife*
Easier Example: The Peanut Butter & Jelly Problem

- **Input:** A closed jar of peanut butter $\textit{jar\_pb}$, a closed jar of jelly $\textit{jar\_jelly}$, a closed bag of toast $\textit{bag\_toast}$, and a knife $\textit{knife}$

- **Output:** A peanut butter & jelly sandwich $\textit{pbj}$
Easier Example: The Peanut Butter & Jelly Problem

- **Input:** A closed jar of peanut butter `jar_pb`, a closed jar of jelly `jar_jelly`, a closed bag of toast `bag_toast`, and a knife `knife`

- **Output:** A peanut butter & jelly sandwich `pbj`

Let’s solve the problem!
Easier Example: The Peanut Butter & Jelly Problem

1. Open bag_toast
2. Remove 2 pieces of toast x and y from bag_toast
3. Close bag_toast
4. Open jar_pb
5. Insert knife into jar_pb
6. Remove knife from jar_pb
7. Spread knife onto x
8. Wipe knife
9. Close jar_pb
10. ...
What is **Algorithm Problem Solving** (APS)?
What is **Algorithm Problem Solving (APS)**?

- An *algorithm describes* a series of operations to perform some task
What is **Algorithm Problem Solving (APS)**?

- An *algorithm* *describes* a series of operations to perform some task
- A *program* is a computer-understandable formulation of an algorithm
What is Algorithm Problem Solving (APS)?

- An algorithm *describes* a series of operations to perform some task
- A program is a computer-understandable formulation of an algorithm
- APS is the process of discovering the algorithm in the first place
What is **Algorithm Problem Solving (APS)**?

- An **algorithm** describes a series of operations to perform some task
- A **program** is a computer-understandable formulation of an algorithm
- **APS** is the process of discovering the algorithm in the first place

**APS → Algorithm → Program**
Example: The Largest Integer Problem

- **Input:** A list of integers \(\text{ints}\)

- **Output:** An integer \(x\) in \(\text{ints}\) such that, for all integers \(y\) in \(\text{ints}\), \(x \geq y\)
  
  ○ In other words, \(x\) is a largest integer in \(\text{ints}\)

Let’s solve the problem!
Example: The Largest Integer Problem

Algorithm `largest_number(ints)`:

\[
x \leftarrow \text{negative infinity}
\]

For every integer \( y \) in \( \text{ints} \):

\[
\text{if } y > x:
\]

\[
x \leftarrow y
\]

Return \( x \)
Example: The Largest Integer Problem

- Our algorithm is correct (can you prove it?)
Example: The Largest Integer Problem

● Our algorithm is correct (can you prove it?)

● However, a single “person” has to look at every integer
Example: The Largest Integer Problem

- Our algorithm is correct (can you prove it?)
- However, a single “person” has to look at every integer
- Even if we had more “people,” they have no way of helping
Example: The Largest Integer Problem

- Our algorithm is correct (can you prove it?)
- However, a single “person” has to look at every integer
- Even if we had more “people,” they have no way of helping
- Can we think of a way to speed things up by working in parallel?
Recursion

- Algorithm that depends on smaller subproblems of itself
Recursion

- Algorithm that depends on smaller subproblems of itself
- Typically composed of two “types” of cases:
Recursion

- Algorithm that depends on smaller subproblems of itself

- Typically composed of two “types” of cases:
  - **Base Case:** Can be solved directly
Recursion

● Algorithm that depends on smaller subproblems of itself

● Typically composed of two “types” of cases:
  ○ **Base Case:** Can be solved directly
  ○ **Recursive Case:** Can be solved using solutions of subproblems
Example: Counting People Recursively

Algorithm `num_people(person)`:

If `person` is at the front of the line:

Return 1

Else:

`neighbor ←` the person in front of `person`

Return `num_people(neighbor) + 1`
Divide-and-Conquer Algorithms
Divide-and-Conquer Algorithms

- **Divide** a given problem into several subproblems
Divide-and-Conquer Algorithms

- **Divide** a given problem into several subproblems
- **Solve** each subproblem recursively
Divide-and-Conquer Algorithms

- **Divide** a given problem into several subproblems
- **Solve** each subproblem recursively
- **Combine** the solutions of the subproblems to solve the problem
Divide-and-Conquer Algorithms

- **Divide** a given problem into several subproblems
- **Solve** each subproblem recursively
- **Combine** the solutions of the subproblems to solve the problem
- **Tip**: Try to balance the sizes of the subproblems as much as possible
A Protocol for Solving Problems
A Protocol for Solving Problems

1. Articulate the problem
A Protocol for Solving Problems

1. Articulate the problem

2. Work out concrete examples, making note of boundary cases
A Protocol for Solving Problems

1. Articulate the problem
2. Work out concrete examples, making note of boundary cases
3. Brainstorm about the algorithm
A Protocol for Solving Problems

1. Articulate the problem
2. Work out concrete examples, making note of boundary cases
3. Brainstorm about the algorithm
4. Design an algorithm
A Protocol for Solving Problems

1. Articulate the problem
2. Work out concrete examples, making note of boundary cases
3. Brainstorm about the algorithm
4. Design an algorithm
5. Analyze the algorithm
A Protocol for Solving Problems

1. Articulate the problem
2. Work out concrete examples, making note of boundary cases
3. Brainstorm about the algorithm
4. Design an algorithm
5. Analyze the algorithm
6. Write the solution
A Protocol for Solving Problems

1. Articulate the problem
2. Work out concrete examples, making note of boundary cases
3. Brainstorm about the algorithm
4. Design an algorithm
5. Analyze the algorithm
6. Write the solution
7. Revise
Example: The Largest Integer Problem

• **Input:** A list of integers \( \text{ints} \)

• **Output:** An integer \( x \) in \( \text{ints} \) such that, for all integers \( y \) in \( \text{ints}, x \geq y \)
  
  ○ In other words, \( x \) is a largest integer in \( \text{ints} \)

Let’s solve the problem!
Example: The Largest Integer Problem

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
</tr>
</tbody>
</table>

largest_integer(ints, start, end)
Example: The Largest Integer Problem

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
</tr>
</tbody>
</table>

largest_integer(ints, 0, 7)
Example: The Largest Integer Problem

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td></td>
</tr>
</tbody>
</table>

largest_integer(ints, 0, 3)
Example: The Largest Integer Problem

```
largest_integer(ints, 0, 1)
```
Example: The Largest Integer Problem

largest_integer(ints, 0, 0)
Example: The Largest Integer Problem

```
largest_integer(ints, 0, 0)
```

```
0 1 2 3 4 5 6 7
a b c d e f g h
```
Example: The Largest Integer Problem

```
largest_integer(ints, 1, 1)
```

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td></td>
</tr>
</tbody>
</table>
```

b
Example: The Largest Integer Problem

```
largest_integer(ints, 0, 1)
i = max(a, b)
```
Example: The Largest Integer Problem

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\text{i} & \text{c} & \text{d} & \text{e} & \text{f} & \text{g} & \text{h} \\
\end{array}
\]

\text{largest_integer(ints, 2, 3)}
Example: The Largest Integer Problem

```
largest_integer(ints, 2, 2)
C
```
Example: The Largest Integer Problem

```
largest_integer(ints, 3, 3)
```
Example: The Largest Integer Problem

```
largest_integer(ints, 2, 3)
j = \max(c,d)
```
Example: The Largest Integer Problem

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```python
largest_integer(ints, 0, 3)  
k = max(i,j)
```
Example: The Largest Integer Problem

```
largest_integer(ints, 4, 5)
l = max(e, f)
```
Example: The Largest Integer Problem

largest_integer(ints, 6, 7)

\[ m = \max(g, h) \]
Example: The Largest Integer Problem

```
largest_integer(ints, 4, 7)  
n = max(l, m)
```
Example: The Largest Integer Problem

```
largest_integer(ints, 0, 7)
```

```
n = max(k, n)
```
Algorithm largest_number(ints, start, end):

    If start equals end:
        Return ints[start]

    Else:
        mid ← floor((start + end) / 2)
        left ← largest_number(ints, start, mid)
        right ← largest_number(ints, mid+1, end)
        Return max(left, right)