CSE 548 / AMS 542: Analysis of Algorithms

Prerequisites Review 1
( Insertion Sort and Selection Sort )

Rezaul Chowdhury
Department of Computer Science
SUNY Stony Brook
Fall 2023
Insertion Sort

State 1
Input array

State 5
Suppose somehow we have this:

State 6
Now from state 5 we want to reach this:

Let State \( j \) be a state of the array \( A \) in which all numbers that were originally in \( A[1..j] \) are placed in sorted order (i.e., nondecreasing order of value) in \( A[1..j] \).
**Insertion Sort**

State 5
Suppose somehow we have this:

```
1 2 3 4 5 6 7 8 9 10
```

```
A
1 4 7 9 15
```

```
5 12 3 18 11
```

State 6
Now from state 5 we want to reach this:

```
1 2 3 4 5 6 7 8 9 10
```

```
1 4 5 7 9 15
```

```
A
5 12 3 18 11
```

**State 5**
Suppose somehow we have this:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>9</td>
<td>15</td>
<td>5</td>
<td>12</td>
<td>3</td>
<td>18</td>
</tr>
</tbody>
</table>

**State 6**


<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>9</td>
<td>5</td>
<td>15</td>
<td>12</td>
<td>3</td>
<td>18</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>9</td>
<td>15</td>
<td>12</td>
<td>3</td>
<td>18</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>15</td>
<td>12</td>
<td>3</td>
<td>18</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>15</td>
<td>12</td>
<td>3</td>
<td>18</td>
</tr>
</tbody>
</table>
## Insertion Sort

<table>
<thead>
<tr>
<th>Input array</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10</td>
<td>7 4 1 15 9 5 12 3 18 11</td>
</tr>
</tbody>
</table>

### State 1

* A[1] is trivially sorted

| 1 2 3 4 5 6 7 8 9 10 | 7 4 1 15 9 5 12 3 18 11 |


### State 2

* A[1..2] is now sorted

| 1 2 3 4 5 6 7 8 9 10 | 4 7 1 15 9 5 12 3 18 11 |

**Insert A[3] in sorted order into A[1..2]**

### State 3

* A[1..3] is now sorted

| 1 2 3 4 5 6 7 8 9 10 | 1 4 7 15 9 5 12 3 18 11 |


### State 4

* A[1..4] is now sorted

| 1 2 3 4 5 6 7 8 9 10 | 1 4 7 15 9 5 12 3 18 11 |


### State 5

* A[1..5] is now sorted

| 1 2 3 4 5 6 7 8 9 10 | 1 4 7 9 15 5 12 3 18 11 |
Insertion Sort

State 5
A[1..5] is now sorted

```
State 6
A[1..6] is now sorted

State 7
A[1..7] is now sorted

State 8
A[1..8] is now sorted

State 9
A[1..9] is now sorted

State 10
A[1..10] is now sorted
```
**Insertion Sort**

**Input:** An array $A[1 : n]$ of $n$ numbers.


**INSERTION-SORT** ($A$)

1. \*\*for\* $j = 2$ to $A.length$  
3. $i = j - 1$  
4. \*\*while $i > 0$ and $A[i] > A[i + 1]$ \*\*  
6. $i = i - 1$
### Worst Case Runtime of Insertion Sort (Upper Bound)

**Insertion-Sort (A)**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Cost ((c_i))</th>
<th>Times ((n))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>(for\ j = 2\ to\ A.\ length)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>(//\ insert\ A[j]\ into\ the\ sorted\ sequence\ A[1..j-1])</td>
<td>0</td>
<td>(n - 1)</td>
</tr>
<tr>
<td>3.</td>
<td>(i = j - 1)</td>
<td>(c_3)</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>(while\ i &gt; 0\ and\ A[i] &gt; A[i + 1])</td>
<td>(c_4)</td>
<td>(\sum_{2\leq j\leq n} j)</td>
</tr>
<tr>
<td>5.</td>
<td>(A[i + 1] \leftrightarrow A[i]) (//\ swap\ A[i]\ and\ A[i + 1])</td>
<td>(c_5)</td>
<td>(\sum_{2\leq j\leq n} (j - 1))</td>
</tr>
<tr>
<td>6.</td>
<td>(i = i - 1)</td>
<td>(c_6)</td>
<td></td>
</tr>
</tbody>
</table>

Running time, \(T(n) \leq c_1 n + c_3 (n - 1) + c_4 \sum_{j=2}^{n} j + c_5 \sum_{j=2}^{n} (j - 1) + c_6 \sum_{j=2}^{n} (j - 1)\)

\[
= 0.5(c_4 + c_5 + c_6)n^2 + 0.5(2c_1 + 2c_3 + c_4 - c_5 - c_6)n - (c_3 + c_4)
\]

\(\Rightarrow T(n) = O(n^2)\)
**Best Case Runtime of Insertion Sort (Lower Bound)**

**Insertion-Sort (A)**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Cost</th>
<th>Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td><code>for j = 2 to A.length</code></td>
<td>$c_1$</td>
<td>$n$</td>
</tr>
<tr>
<td>2.</td>
<td><code>// insert A[j] into the sorted sequence A[1..j-1]</code></td>
<td>$c_3$</td>
<td>$n - 1$</td>
</tr>
<tr>
<td>3.</td>
<td>$i = j - 1$</td>
<td>$c_5$</td>
<td>$0$</td>
</tr>
<tr>
<td>4.</td>
<td><code>while i &gt; 0 and A[i] &gt; A[i + 1]</code></td>
<td>$c_4$</td>
<td>$0$</td>
</tr>
<tr>
<td>6.</td>
<td>$i = i - 1$</td>
<td>$c_2$</td>
<td>$0$</td>
</tr>
</tbody>
</table>

Running time, $T(n) \geq c_1 n + c_3 (n - 1) + c_4 (n - 1)$

$$= (c_1 + c_3 + c_4) n - (c_3 + c_4)$$

$$\Rightarrow T(n) = \Omega(n)$$
**Insertion Sort**  
*(Slightly Optimized but Same Asymptotic Bounds)*

**Input:** An array $A[1: n]$ of $n$ numbers.


```plaintext
**INSERTION-SORT ( A )**

1. for $j = 2$ to $A.length$
2. key = $A[j]$
3. // insert $A[j]$ into the sorted sequence $A[1..j-1]$
4. $i = j - 1$
5. while $i > 0$ and $A[i] > key$
7. $i = i - 1$
8. $A[i + 1] = key$
```
Selection Sort

Input array

Let **State j** be a state of the array $A$ in which the smallest $j$ numbers of $A[1..n]$ are placed in sorted order (i.e., nondecreasing order of value) in $A[1..j]$, and the remaining numbers placed in arbitrary order in $A[j + 1..n]$.

**State 0**
Input array

**State 4**
Suppose somehow we have this:

**State 5**
Now from state 4 we want to reach this:
**State 4**
Suppose somehow we have this:

```
A  1 3 4 5 9 15 12 7 18 11
```


**State 5**
Now from state 4 we want reach this:

```
A  1 3 4 5 7 15 12 9 18 11
```
Selection Sort

State 4
Suppose somehow we have this:

\[ A = [1, 3, 4, 5, 9, 15, 12, 7, 18, 11] \]

- smallest 4 numbers in sorted order
- remaining 6 numbers

\[ \text{min} = 5 \]

\[ A[6] \geq A[\text{min}] \]
State 4
Suppose somehow we have this:

\[ \text{smallest 4 numbers in sorted order} \]
\[ \text{remaining 6 numbers} \]

\[ m_{\text{in}} = 5 \]

\[ A[6] \geq A[m_{\text{in}}] \]
Move on
**State 4**
Suppose somehow we have this:

```
A
1 3 4 5 9 15 12 7 18 11
```

- Smallest 4 numbers in sorted order: 1, 3, 4, 5
- Remaining 6 numbers: 9, 15, 12, 7, 18, 11

$\text{min} = 5$

$A[7] \geq A[\text{min}]$
**State 4**
Suppose somehow we have this:

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>9</th>
<th>15</th>
<th>12</th>
<th>7</th>
<th>18</th>
<th>11</th>
</tr>
</thead>
</table>

*smallest 4 numbers in sorted order*

*A*

*remaining 6 numbers*

\[ m_{\text{in}} = 5 \]

\[ A[7] \geq A[m_{\text{in}}] \]

Move on
State 4
Suppose somehow we have this:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td>15</td>
<td>12</td>
<td>7</td>
<td>18</td>
<td>11</td>
</tr>
</tbody>
</table>

- **smallest 4 numbers in sorted order**: 1, 3, 4, 5
- **remaining 6 numbers**: 6, 7, 8, 9, 10, 11

\[ \text{min} = 5 \]

\[ A[8] < A[\text{min}] \]
State 4
Suppose somehow we have this:

\[ A = \begin{array}{cccccccc}
1 & 3 & 4 & 5 & 9 & 15 & 12 & 7 & 18 & 11
\end{array} \]

**selection sort**

**smallest 4 numbers in sorted order**

**remaining 6 numbers**

\[ m_{\text{in}} = 8 \]

\[ A[8] < A[m_{\text{in}}] \]

\[ m_{\text{in}} = 8 \]

Move on
State 4
Suppose somehow we have this:

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 9 & 15 & 12 & 7 & 18 & 11 \\
\end{array}
\]

\[m_{\text{in}} = 8\]

\[A[9] \geq A[m_{\text{in}}]\]
Selection Sort

State 4
Suppose somehow we have this:

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 9 & 15 & 12 & 7 & 18 & 11 \\
\end{array}
\]

\(\text{smallest 4 numbers in sorted order}\)
\(\text{remaining 6 numbers}\)

\(m_{\text{in}} = 8\)

\(A[9] \geq A[m_{\text{in}}]\)
Move on
State 4
Suppose somehow we have this:

\[ \min = 8 \]

\[ A[10] \geq A[\min] \]
State 4
Suppose somehow we have this:

\[ \text{smallest 4 numbers in sorted order} \]
\[ \text{remaining 6 numbers} \]

\[ m_{\text{in}} = 8 \]
\[ A[10] \geq A[m_{\text{in}}] \]
Done scanning
State 4
Suppose somehow we have this:

\[ \text{smallest 4 numbers in sorted order} \]
\[ \text{remaining 6 numbers} \]

\[ m_{\text{in}} = 8 \]

Selection Sort

State 5

smallest 5 numbers in sorted order

remaining 5 numbers

\[ m_{\text{in}} = 8 \]

\[ \text{Swap } A[5] \text{ and } A[m_{\text{in}}] \]
Selection Sort

State 0
Input array

\[1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10\]

\[\text{A} \ 7 \ 4 \ 1 \ 15 \ 9 \ 5 \ 12 \ 3 \ 18 \ 11\]

\[\downarrow\]
Swap \text{A}[1] with the smallest number in \text{A}[1..10]

State 1
\text{A}[1] is now sorted

\[1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10\]

\[\text{A} \ 1 \ 4 \ 7 \ 15 \ 9 \ 5 \ 12 \ 3 \ 18 \ 11\]

\[\downarrow\]
Swap \text{A}[2] with the smallest number in \text{A}[2..10]

State 2
\text{A}[1..2] is now sorted

\[1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10\]

\[\text{A} \ 1 \ 3 \ 7 \ 15 \ 9 \ 5 \ 12 \ 4 \ 18 \ 11\]

\[\downarrow\]
Swap \text{A}[3] with the smallest number in \text{A}[3..10]

State 3
\text{A}[1..3] is now sorted

\[1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10\]

\[\text{A} \ 1 \ 3 \ 4 \ 15 \ 9 \ 5 \ 12 \ 7 \ 18 \ 11\]

\[\downarrow\]
Swap \text{A}[4] with the smallest number in \text{A}[4..10]

State 4
\text{A}[1..4] is now sorted

\[1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10\]

\[\text{A} \ 1 \ 3 \ 4 \ 5 \ 15 \ 9 \ 12 \ 7 \ 18 \ 11\]

\[\downarrow\]
Swap \text{A}[5] with the smallest number in \text{A}[5..10]

State 5
\text{A}[1..5] is now sorted

\[1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10\]

\[\text{A} \ 1 \ 3 \ 4 \ 5 \ 7 \ 15 \ 12 \ 9 \ 18 \ 11\]
## Selection Sort

**State 5**

$A[1..5]$ is now sorted


<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>18</td>
<td>11</td>
</tr>
</tbody>
</table>

**State 6**

$A[1..6]$ is now sorted


<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**State 7**

$A[1..7]$ is now sorted


<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td>11</td>
<td>15</td>
<td>18</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**State 8**

$A[1..8]$ is now sorted


<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td>12</td>
<td>18</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**State 9**

$A[1..9]$ is now sorted

Do nothing!

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**State 10**

$A[1..10]$ is now sorted

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Do nothing!
Selection Sort

**Input:** An array $A[1 \ldots n]$ of $n$ numbers.

**Output:** Elements of $A[1 \ldots n]$ rearranged in non-decreasing order of value.

```
FUNCTION SELECTION-SORT (A)
    1. for $j = 1 \rightarrow A.length - 1$
    2.     // find the index of an entry with the smallest value in $A[j..A.length]$
    3.     $min = j$
    4.     for $i = j + 1 \rightarrow A.length$
    5.         if $A[i] < A[min]$
    6.             $min = i$
    7.     // swap $A[j]$ and $A[min]$
```
[ Optional ]
Proof of Correctness of Insertion Sort
Loop Invariants

We use loop invariants to prove correctness of iterative algorithms.

A loop invariant is associated with a given loop of an algorithm, and it is a formal statement about the relationship among variables of the algorithm such that

- **[Initialization]** It is true prior to the first iteration of the loop.
- **[Maintenance]** If it is true before an iteration of the loop, it remains true before the next iteration.
- **[Termination]** When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.
Loop Invariants for Insertion Sort

**INSERTION-SORT ( A )**

1. **for** $j = 2$ **to** $A.length$
2. $key = A[j]$
3. // insert $A[j]$ into the sorted sequence $A[1..j-1]$
4. $i = j - 1$
5. **while** $i > 0$ **and** $A[i] > key$
7. $i = i - 1$
8. $A[i + 1] = key$
Loop Invariants for Insertion Sort

**INSERTION-SORT (A)**

1. **for** \( j = 2 \) **to** \( A.length \)
   
   **Invariant 1:** \( A[1..j-1] \) consists of the elements originally in \( A[1..j-1] \), but in sorted order

2. \( key = A[j]\)

3. // insert \( A[j] \) into the sorted sequence \( A[1..j-1] \)

4. \( i = j - 1 \)

5. **while** \( i > 0 \) **and** \( A[i] > key \)


7. \( i = i - 1 \)

8. \( A[i + 1] = key \)
Loop Invariants for Insertion Sort

**INSERTION-SORT** (A)

1. \( \text{for } j = 2 \text{ to } A\.\text{length} \)
   
   **Invariant 1:** \( A[1..j-1] \) consists of the elements originally in \( A[1..j-1] \), but in sorted order

2. \( \text{key} = A[j] \)

3. // insert \( A[j] \) into the sorted sequence \( A[1..j-1] \)

4. \( i = j - 1 \)

5. \( \text{while } i > 0 \text{ and } A[i] > \text{key} \)

   **Invariant 2:** \( A[i..j] \) are each \( \geq \text{key} \)


7. \( i = i - 1 \)

8. \( A[i+1] = \text{key} \)
Loop Invariant 1: Initialization

At the start of the first iteration of the loop (in lines 1 – 8): \( j = 2 \)


The subarray consisting of a single element is trivially sorted.

Hence, the invariant holds initially.
Loop Invariant 1: Maintenance

We assume that invariant 1 holds before the start of the current iteration. Hence, the following holds: $A[1..j-1]$ consists of the elements originally in $A[1..j-1]$, but in sorted order.

For invariant 1 to hold before the start of the next iteration, the following must hold at the end of the current iteration:

$A[1..j]$ consists of the elements originally in $A[1..j]$, but in sorted order.

We use invariant 2 to prove this.
Loop Invariant 1: Maintenance
Loop Invariant 2: Initialization

At the start of the first iteration of the loop (in lines 5 – 7): \( i = j - 1 \)

Hence, subarray \( A[i..j] \) consists of only two entries: \( A[i] \) and \( A[j] \).

We know the following:

\[ \text{— } A[i] > \text{key } (\text{explicitly tested in line 5}) \]
\[ \text{— } A[j] = \text{key } (\text{from line 2}) \]

Hence, invariant 2 holds initially.
We assume that invariant 2 holds before the start of the current iteration.

Hence, the following holds: $A[i..j]$ are each $\geq key$.

Since line 6 copies $A[i]$ which is known to be $> key$ to $A[i + 1]$ which also held a value $\geq key$, the following holds at the end of the current iteration: $A[i + 1..j]$ are each $\geq key$.

Before the start of the next iteration the check $A[i] > key$ in line 5 ensures that invariant 2 continues to hold.
Observe that the inner loop (in lines 5 − 7) does not destroy any data because though the first iteration overwrites $A[j]$, that $A[j]$ has already been saved in $key$ in line 2.

As long as $key$ is copied back into a location in $A[1..j]$ without destroying any other element in that subarray, we maintain the invariant that $A[1..j]$ contains the first $j$ elements of the original list.
Loop Invariant 1: Maintenance
Loop Invariant 2: Termination

When the inner loop terminates we know the following.

— $A[1..i]$ is sorted with each element $\leq key$
  
  - if $i = 0$, true by default
  - if $i > 0$, true because $A[1..i]$ is sorted and $A[i] \leq key$

— $A[i+1..j]$ is sorted with each element $\geq key$ because the following held before $i$ was decremented: $A[i..j]$ is sorted with each item $\geq key$

— $A[i+1] = A[i + 2]$ if the loop was executed at least once, and $A[i + 1] = key$ otherwise
Loop Invariant 1: Maintenance
Loop Invariant 2: Termination

When the inner loop terminates we know the following.

- $A[1..i]$ is sorted with each element $\leq key$
- $A[i + 1..j]$ is sorted with each element $\geq key$

Given the facts above, line 8 does not destroy any data, and gives us $A[1..j]$ as the sorted permutation of the original data in $A[1..j]$. 
Loop Invariant 1: Termination

When the outer loop terminates we know that $j = A.\text{length} + 1$.

Hence, $A[1..j - 1]$ is the entire array $A[1..A.\text{length}]$, which is sorted and contains the original elements of $A[1..A.\text{length}]$. 

```plaintext
INSERTION-SORT (A)
1. for $j = 2$ to $A.\text{length}$
   Invariant 1: $A[1..j - 1]$ consists of the elements originally in $A[1..j - 1]$, but in sorted order
2. key = $A[j]$
3. // Insert $A[j]$ into the sorted sequence $A[1..j - 1]$
4. $i = j - 1$
5. while $i > 0$ and $A[i] > key$
   Invariant 2: $A[i..j]$ are each $\geq key$
7. $i = i - 1$
8. $A[i + 1] = key$
```