Principled Programming
Introduction to Coding in Any Imperative Language

Tim Teitelbaum
Emeritus Professor
Department of Computer Science
Cornell University

Binary Search
If you want to find the definition of the word **proboscis** in a 512-page dictionary, you wouldn’t use Sequential Search starting on the first page, say, with **aardvark**. Rather, you would start roughly in the middle. From there, you would:

- Repeatedly halve the portion of the dictionary that remains under consideration, doing so by looking at the middle page of the region in hand, and discarding whichever half is revealed thereby to not contain **proboscis**.
- Once the search has been narrowed to a single page, you would look on that page to see if **proboscis** is there.
- If it is, you found its definition; otherwise, it isn’t in the dictionary.

The method is called **Binary Search**, and is an example of a **Divide and Conquer algorithm**. Binary Search is astoundingly fast.
**Application:** Search for a value \( v \) in an unordered array \( A[0..n-1] \).

/* Given array \( A[0..n-1] \), \( n \geq 0 \), and value \( v \), let \( k \) be an index of \( A \) where \( A[k] == v \), or \( n \) if no \( v \) in \( A \). */

---

☞ **A statement-comment says exactly what code must accomplish, not how it does so.**
Application: Search for a value v in an ordered array A[0..n-1].

/* Given ordered array A[0..n-1], n≥0, and value v, let k be an index of A where A[k]==v, or n if no v in A. */
Application: Search for a value $v$ in an ordered array $A[0..n-1]$.

/* Given ordered array $A[0..n-1]$, $n \geq 0$, and value $v$, let $k$ be an index of $A$ where $A[k] == v$, or $n$ if no $v$ in $A$. */

int $k = 0$;
while ( $A[k] != v$ && $k < n$ ) $k++$;

Master stylized code patterns, and use them.

Sequential search works, but ignores the order. We can do better.
Application: Search for a value v in an ordered array A[0..n-1].

/* Given ordered array A[0..n-1], n≥0, and value v, let k be an index of A where A[k]==v, or n if no v in A. */

__________
while ( _____ ) ___________
__________

☞ If you “smell a loop”, write it down.
Application: Search for a value v in an ordered array A[0..n-1].

/* Given ordered array A[0..n-1], n≥0, and value v, let k be an index of A where A[k]==v, or n if no v in A. */

while (______) _________

Invent (or learn) diagrammatic ways to express concepts.
Application: Search for a value $v$ in an ordered array $A[0..n-1]$.

/* Given ordered array $A[0..n-1]$, $n \geq 0$, and value $v$, let $k$ be an index of $A$ where $A[k] == v$, or $n$ if no $v$ in $A$. */

```plaintext
while ( ______ ) ___________

L  n

A  v in here, if $v$ in $A[0..n-1]$

INVARIANT
```

To get to POST iteratively, choose a weakened POST as IN Variant.
Application: Search for a value v in an ordered array A[0..n-1].

/* Given ordered array A[0..n-1], n≥0, and value v, let k be an index of A where A[k]==v, or n if no v in A. */
int L = ______; int R = ______;
while ( ______ ) ___________

Introduce program variables whose values describe “state”. 
**Application:** Search for a value $v$ in an ordered array $A[0..n-1]$. 

```c
/* Given ordered array $A[0..n-1]$, $n \geq 0$, and value $v$, let $k$ be an index of $A$ where $A[k]=v$, or $n$ if no $v$ in $A$. */
int L = ______; int R = ______;
while ( ______ )
    if ( ______ ) __________ else __________
```

**A Case Analysis in the loop body is often needed for characterizing different ways in which to decrease the loop variant while maintaining the loop invariant.**
Application: Search for a value $v$ in an ordered array $A[0..n-1]$.

/* Given ordered array $A[0..n-1]$, $n \geq 0$, and value $v$, let $k$ be an index of $A$ where $A[k] == v$, or $n$ if no $v$ in $A$. */

```c
int L = ______; int R = ______;
while ( ______ )
    if ( ________ )
        R = ______; // Select left "half".
    else L = ______; // Select right "half".
```

A Case Analysis in the loop body is often needed for characterizing different ways in which to decrease the loop variant while maintaining the loop invariant.
**Application:** Search for a value $v$ in an ordered array $A[0..n-1]$.

```c
/* Given ordered array $A[0..n-1]$, $n \geq 0$, and value $v$, let $k$ be an index of $A$ where $A[k] == v$, or $n$ if no $v$ in $A$. */
int L = _____; int R = _____;
M = _____; // Compute "midpoint".
while ( _____ )
    if ( _____ )
        R = _____; // Select left "half".
    else L = _____; // Select right "half".
```

---

**VARIANT:** $R-L$

**INVARIANT**

---

A Case Analysis in the loop body is often needed for characterizing different ways in which to **decrease the loop variant** while maintaining the loop invariant.
**Application:** Search for a value v in an ordered array A[0..n-1].

/* Given ordered array A[0..n-1], n≥0, and value v, let k be an index of A
where A[k]==v, or n if no v in A. */

int L = ______; int R = ______;
M = ______; // Compute “midpoint”.
while ( ______ )
    if ( ________ )
        R = ______; // Select left “half”.
    else L = ______; // Select right “half”.

If you object to A[L..R] straddling the midpoint of A[0..n-1], understand that in
“schematic diagrams”, the exact locations of boundaries are immaterial.
Application: Search for a value v in an ordered array A[0..n-1].

/* Given ordered array A[0..n-1], n≥0, and value v, let k be an index of A where A[k]==v, or n if no v in A. */
int L = _____; int R = _____;
M = _____; // Compute “midpoint”.
while ( _____ )
  if ( _____ )
    R = _____; // Select left “half”.
  else L = _____; // Select right “half”.

Be alert to high-risk coding steps associated with binary choices.

Recognize that regions of even and odd lengths may need distinct treatments.
Application: Search for a value \( v \) in an ordered array \( A[0..n-1] \).

/* Given ordered array \( A[0..n-1] \), \( n \geq 0 \), and value \( v \), let \( k \) be an index of \( A \) where \( A[k] == v \), or \( n \) if no \( v \) in \( A \). */

```
int L = ______;  // Select left “half”.
int R = ______;  // Select right “half”.
```

// Compute “midpoint”.

```
M = (L+R)/2;
```

while ( ______ )
```
if ( ________ )
    R = ______;  // Select left “half”.
else L = ______;  // Select right “half”.
```

Be alert to high-risk coding steps associated with binary choices.

Recognize that regions of even and odd lengths may need distinct treatments, but hope for a uniform treatment.
Application: Search for a value $v$ in an ordered array $A[0..n-1]$.

/* Given ordered array $A[0..n-1]$, $n \geq 0$, and value $v$, let $k$ be an index of $A$ where $A[k] == v$, or $n$ if no $v$ in $A$. */
int $L = \ldots$; int $R = \ldots$;
M = (L+R)/2; // Compute “midpoint”.
while ( \ldots )
    if ( \ldots )
        $R = M$; // Select left “half”.
    else $L = M+1$; // Select right “half”.

☞ Be alert to high-risk coding steps associated with binary choices.
**Application**: Search for a value $v$ in an ordered array $A[0..n-1]$.

/* Given ordered array $A[0..n-1]$, $n \geq 0$, and value $v$, let $k$ be an index of $A$ where $A[k] == v$, or $n$ if no $v$ in $A$. */

```c
int L = ______; int R = ______;
M = (L+R)/2; // Compute “midpoint”.
while ( ______ )
    if ( v __ A[M] )
        R = M; // Select left “half”.
    else L = M+1; // Select right “half”.
```

Be alert to high-risk coding steps associated with binary choices.
Application: Search for a value $v$ in an ordered array $A[0..n-1]$.

/* Given ordered array $A[0..n-1]$, $n\geq 0$, and value $v$, let $k$ be an index of $A$ where $A[k]==v$, or $n$ if no $v$ in $A$. */
int $L = ____$; int $R = ____$
M = (L+R)/2; // Compute “midpoint”.
while ( ____ )
    if ( $v \leq A[M] $ )
        $R = M$; // Select left “half”.
    else $L = M+1$; // Select right “half”.

☞ Be alert to high-risk coding steps associated with binary choices.
Application: Search for a value \( v \) in an ordered array \( A[0..n-1] \).

/* Given ordered array \( A[0..n-1] \), \( n \geq 0 \), and value \( v \), let \( k \) be an index of \( A \) where \( A[k] == v \), or \( n \) if no \( v \) in \( A \). */

```c
int L = ______; int R = ______;
M = (L+R)/2;           // Compute “midpoint”.
while ( ______ )
    if ( v <= A[M] )
        R = M;           // Select left “half”.
    else L = M+1;       // Select right “half”.
```

Be alert to high-risk coding steps associated with binary choices.
Application: Search for a value v in an ordered array A[0..n-1].

/* Given ordered array A[0..n-1], n≥0, and value v, let k be an index of A where A[k]==v, or n if no v in A. */

int L = ______; int R = ______;
M = (L+R)/2;  // Compute “midpoint”.
while ( ______ )
    if ( v <= A[M] )
        R = M;  // Select left “half”.
    else L = M+1;  // Select right “half”.

Be alert to high-risk coding steps associated with binary choices.

Duplicate instances of v in A[L..R] may escape, but not all of them.
**Application**: Search for a value \( v \) in an ordered array \( A[0..n-1] \).

/* Given ordered array \( A[0..n-1] \), \( n \geq 0 \), and value \( v \), let \( k \) be an index of \( A \) where \( A[k] == v \), or \( n \) if no \( v \) in \( A \). */

```
int L = ______; int R = ______;
M = (L+R)/2; // Compute "midpoint".
while ( L != R )
    if ( v <= A[M] )
        R = M; // Select left "half".
    else L = M+1; // Select right "half".
```

<table>
<thead>
<tr>
<th>Coding order</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) body</td>
</tr>
<tr>
<td>(2) termination</td>
</tr>
<tr>
<td>(3) initialization</td>
</tr>
<tr>
<td>(4) finalization</td>
</tr>
<tr>
<td>(5) boundary conditions</td>
</tr>
</tbody>
</table>
Application: Search for a value v in an ordered array A[0..n-1].

/* Given ordered array A[0..n-1], n≥0, and value v, let k be an index of A where A[k]==v, or n if no v in A. */
int L = ______; int R = ______;
M = (L+R)/2;       // Compute “midpoint”.
while ( L != R )
    if ( v <= A[M] )
        R = M;       // Select left “half”.
    else L = M+1;   // Select right “half”.

Confirm that the **VARIANT** decreases on every iteration.
**Application**: Search for a value \( v \) in an ordered array \( A[0..n-1] \).

```plaintext
/* Given ordered array \( A[0..n-1] \), \( n \geq 0 \), and value \( v \), let \( k \) be an index of \( A \) where \( A[k] == v \), or \( n \) if no \( v \) in \( A \). */
int L = ______; int R = ______;
M = (L+R)/2;  // Compute "midpoint".
while ( L != R )
    if ( v <= A[M] )
        R = M;    // Select left "half".
    else L = M+1;  // Select right "half".
```

**Confirm that the VARIANT decreases on every iteration.**
Application: Search for a value v in an ordered array A[0..n-1].

/* Given ordered array A[0..n-1], n≥0, and value v, let k be an index of A where A[k]==v, or n if no v in A. */
int L = _____; int R = _____;
M = (L+R)/2; // Compute “midpoint”.
while ( L != R )
  if ( v <= A[M] )
    R = M; // Select left “half”.
  else L = M+1; // Select right “half”.

Confirm that the **VARIANT** decreases on every iteration.
**Application**: Search for a value $v$ in an ordered array $A[0..n-1]$.

/* Given ordered array $A[0..n-1]$, $n \geq 0$, and value $v$, let $k$ be an index of $A$ where $A[k] == v$, or $n$ if no $v$ in $A$. */

```c
int L = 0; int R = n-1;
M = (L+R)/2;           // Compute "midpoint".
while ( L != R )
    if ( v <= A[M] )
        R = M;           // Select left "half".
    else L = M+1;       // Select right "half".
```

### Coding order

<table>
<thead>
<tr>
<th>Coding order</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) body</td>
</tr>
<tr>
<td>(2) termination</td>
</tr>
<tr>
<td>(3) initialization</td>
</tr>
<tr>
<td>(4) finalization</td>
</tr>
<tr>
<td>(5) boundary conditions</td>
</tr>
</tbody>
</table>
Application: Search for a value $v$ in an ordered array $A[0..n-1]$.

/* Given ordered array $A[0..n-1]$, $n \geq 0$, and value $v$, let $k$ be an index of $A$ where $A[k] == v$, or $n$ if no $v$ in $A$. */

```c
int L = 0; int R = n-1;
M = (L+R)/2;           // Compute "midpoint".
while ( L != R )
    if ( v <= A[M] )
        R = M;           // Select left "half".
    else L = M+1;       // Select right "half".
if ( A[L] == v ) k = L; else k = n;
```

Coding order

<table>
<thead>
<tr>
<th>Coding order</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) body</td>
</tr>
<tr>
<td>(2) termination</td>
</tr>
<tr>
<td>(3) initialization</td>
</tr>
<tr>
<td>(4) finalization</td>
</tr>
<tr>
<td>(5) boundary conditions</td>
</tr>
</tbody>
</table>
Application: Search for a value v in an ordered array A[0..n-1].

/* Given ordered array A[0..n-1], n≥0, and value v, let k be an index of A
where A[k]==v, or n if no v in A. */
int L = 0; int R = n-1;
M = (L+R)/2; // Compute “midpoint”.
while ( L != R )
    if ( v <= A[M] )
        R = M; // Select left “half”.
    else L = M+1; // Select right “half”.
if ( A[L]==v ) k = L; else k = n;
Application: Search for a value $v$ in an ordered array $A[0..n-1]$. 

/* Given ordered array $A[0..n-1]$, $n \geq 0$, and value $v$, let $k$ be an index of $A$ where $A[k] == v$, or $n$ if no $v$ in $A$. */
if ( n==0 ) $k = 0$;
else {
    int $L = 0$; int $R = n-1$;
    $M = (L+R)/2$; // Compute “midpoint”.
    while ( $L != R$ )
        if ( $v <= A[M]$ )
            $R = M$; // Select left “half”.
        else $L = M+1$; // Select right “half”.
    if ( $A[L] == v$ ) $k = L$; else $k = n$;
}

Coding order

<table>
<thead>
<tr>
<th>Coding order</th>
<th>Code order</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) body</td>
<td>for body</td>
</tr>
<tr>
<td>(2) termination</td>
<td>for termination</td>
</tr>
<tr>
<td>(3) initialization</td>
<td>for initialization</td>
</tr>
<tr>
<td>(4) finalization</td>
<td>for finalization</td>
</tr>
<tr>
<td>(5) boundary conditions</td>
<td>for boundary conditions</td>
</tr>
</tbody>
</table>
Application: Search for a value $v$ in an ordered array $A[0..n-1]$.

/* Given ordered array $A[0..n-1]$, $n\geq0$, and value $v$, let $k$ be an index of $A$ where $A[k]=v$, or $n$ if no $v$ in $A$. */

if ( n==0 ) $k = 0$;

else {
    int $L = 0$; int $R = n-1$;
    int $M = (L+R)/2$; // Compute "midpoint".
    while ( $L != R$ )
        if ( $v \leq A[M]$ )
            $R = M$; // Select left "half".
        else $L = M+1$; // Select right "half".
        if ( $A[L]==v$ ) $k = L$; else $k = n$;
    }

Coding order

<table>
<thead>
<tr>
<th>(1) body</th>
<th>(2) termination</th>
<th>(3) initialization</th>
<th>(4) finalization</th>
<th>(5) boundary conditions</th>
</tr>
</thead>
</table>

```
L
R
A

0 1 2 3 4 5 6

k

0

found
```
Application: Search for a value $v$ in an ordered array $A[0..n-1]$. 

/* Given ordered array $A[0..n-1]$, $n \geq 0$, and value $v$, let $k$ be an index of $A$ where $A[k]=v$, or $n$ if no $v$ in $A$. */
if ( n==0 ) k = 0;
else {
    int L = 0; int R = n-1;
    M = (L+R)/2;               // Compute “midpoint”.
    while ( L != R )
        if ( v <= A[M] )
            R = M;                // Select left “half”.
        else L = M+1;           // Select right “half”.
    if ( A[L]==v ) k = L; else k = n;
}

⚠️ Is it a problem that $k==0$ represents both “not found” and “found in $0^{th}$ element”?

<table>
<thead>
<tr>
<th>Coding order</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) body</td>
</tr>
<tr>
<td>(2) termination</td>
</tr>
<tr>
<td>(3) initialization</td>
</tr>
<tr>
<td>(4) finalization</td>
</tr>
<tr>
<td>(5) boundary conditions</td>
</tr>
</tbody>
</table>
Application: Search for a value `v` in an ordered array `A[0..n-1].`

/* Given ordered array `A[0..n-1]`, `n≥0`, and value `v`, let `k` be an index of `A` where `A[k]==v`, or `n` if no `v` in `A`. */
if ( n==0 ) k = 0;
else {
    int L = 0; int R = n-1;
    M = (L+R)/2;           // Compute “midpoint”.
    while ( L != R )
        if ( v <= A[M] )
            R = M;           // Select left “half”.
        else L = M+1;       // Select right “half”.
    if ( A[L]==v ) k = L; else k = n;
}
if ( k<n ) /* Found. */ else /* Not found. */

No. What matters is whether `k<n`, not whether `k==0`. 
Binary Search is astoundingly fast. If \( n=512 \), just 9 iterations to termination!

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>VARIANT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>512</td>
</tr>
<tr>
<td>1</td>
<td>256</td>
</tr>
<tr>
<td>2</td>
<td>128</td>
</tr>
<tr>
<td>3</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>

Running time is logarithmic in \( n \), and independent of whether \( v \) is in \( A \) or not.