**Linear Search**

```python
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 += 1
    if i == len(b):
        return -1
    return i
```

**Binary Search**

```python
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:
            j = mid
        elif b[mid] > v:
            i = mid
        else:
            return mid
    return -1
```

**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”

**Horizontal Notation**

- Want a pictoral 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

**Visualizing Sorting**

**Insertion Sort**

```plaintext
1 = 0
while 1 < n:
    # Push b[i] down into its
    # sorted position in b[0..i]
    i = i+1
```
Insertion Sort: Moving into Position

\[
\begin{align*}
&\text{i} = 0 \\
&\text{while i < n:} \\
&\quad \text{push_down(b, i)} \\
&\quad \text{i} = \text{i} + 1 \\
\end{align*}
\]

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

```
2  4  6  6  7
0  i

0  2  4  6  6  7
2  4  6  6  7
2  4  6  6  7
```

```
2  4  6  6  7  5
0  i

0  2  4  6  6  7
2  4  6  5  6
2  4  5  6  6
```

Remember the restrictions!