### Invariants: Keep in mind

- At heart, an invariant is just a way to document *what you want your variables to mean.*
  This is why you want your code to keep the invariant true; you want to keep things consistent in your program, and in your head.

- In our notation, both $b[1..i-1]$ and $b[i..i-1]$ denote an empty sequence.

### Linear search in unsorted lists

**Goal:** Given unsorted list $b$, search range $h..k-1$ for $k \geq h$, and target value $v$, *return* index $n$ of $v$'s first occurrence in $b[h..k-1]$ (-1 if not found)

Restated as postcondition: if $n=-1$, then $v$ is not in $b[h..k-1]$. Otherwise, $v = b[n]$ and $v$ is not in $b[h..n-1]$.

**Idea:** keep an index $i$, marking position of next thing unchecked; everything to its left has been verified to not be $v$.

![Linear Search](image)

![Analyzing the Loop](image)

### Binary search in sorted lists

**Goal:** Given *sorted* list $b$, search range $h..k$ for $k \geq h$, and target value $v$, *return* index $n$ of $v$'s first occurrence in $b[h..k-1]$ (-1 if not found)

Restated as postcondition: if $n=-1$, then $v$ is not in $b[h..k-1]$. Otherwise, $v = b[n]$ and $v$ is not in $b[h..n-1]$.  

**Idea:** keep indices $i$ and $j$, marking position of next thing not known to be $<v$, and the first thing known to be $\geq v$. Check halfway between.

![Binary Search Implementation](image)
**Sorting: Selection Sort**

<table>
<thead>
<tr>
<th>pre: b</th>
<th>?</th>
<th>n</th>
<th>post: b</th>
<th>sorted</th>
<th>n</th>
</tr>
</thead>
</table>

**Selection Sort:**

| inv: b | sorted, a[b[0:n]] | i | j | a[b[0..i-1]] or ? if i = 0 | n |

**INITIALIZE AND COMPLETE**

```
while ...:
    j = b[0:n].index(min(b[i:n]))
```

```
Note the swap of the red
```

```
2 4 6 6 8 9 9 7 8 9
```

```
j 2 4 6 6 7 9 9 8 8 8 9
```

```
i 2 4 6 6 7 9 9 8 8 9
```

**Famous "Sort-Like" Example**

- Dutch national flag: tri-color
  - Sequence of h..k of red (<0), white (=0), blue (>0) "pixels"
  - Arrange to put <0 first, then =0, then >0, return "split pts"

| pre: b | ? | h | k | post: b | <0 | =0 | >0 |

| inv: b | <0 | ? | =0 | >0 |

| b[h..t-1] <0, b[t..i-1] unknown, b[i..j] =0, b[j+1..k] >0 |

**Dutch National Flag Algorithm**

```python
def dnf(b, h, k):
    """(DNF explanation omitted for space.)
    Returns: split-points as a tuple (i,j)"
    # init
    # inv: b[h..t-1] < 0, b[t..i-1] ?, b[i..j] = 0, b[j+1..k] > 0
    while t < i:
        if b[i-1] < 0:
            # what?
        elif b[i-1] == 0:
            # what?
        else:
            # what?
        # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
        return (i, j)
```

```python
# def dnf(b, h, k):
#     """(DNF explanation omitted for space.)
#     Returns: split-points as a tuple (i,j)"
#     # init
#     # inv: b[h..t-1] < 0, b[t..i-1] ?, b[i..j] = 0, b[j+1..k] > 0
#     while t < i:
#         if b[i-1] < 0:
#             # what?
#         elif b[i-1] == 0:
#             # what?
#         else:
#             # what?
#         # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
#         return (i, j)
```