Linear search in unsorted lists

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

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

	h	i	k
inv: b	v not here	?	

Linear Search

	h	i	k
inv: b	v not here	?	

```
def linear_search(b, h, k, v):
    """(see previous)"""
    i = h
    # inv: v not in b[h..i-1]
    while i <= k and b[i] != v:
        i = i + 1
    # post: b[i] = v and v not in b[h..i-1], or,
    #       i = k+1 and v not in b[h..k]
    n = i if i <= k else -1
    return n
```

Analyzing the Loop

1. Does the initialization make **inv** true?
2. Is **post** true when **inv** is true and **condition** is false?
3. Does the loop body make progress?
4. Does the loop body keep **inv** true?

Binary search in *sorted* lists

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

Restated as postcondition: if n = -1, then v is not in b[h..k]. 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 ≥ v. Check in the middle.

	h	i	j	k
inv: b	<v	?	≥v	

	h	i	j	k
inv: b	<v	?	>=v	

```
def bin_search(b, h, k, v):
    """(see previous)"""
    Q1: (A) i = h; j = k   (B) i = h-1; j = k+1   (C) i = h-1; j = k   (D) i = h; j = k+1
    # inv: b[h..i-1] < v, b[j..k] >=v, i <= j
    while Q2:   (A) i == j   (B) i < j   (C) i <= j
        mid = (i + j)/2
        if b[mid] < v:
            Q3:   (A) i = mid   (B) i = mid+1
        else:
            j = mid   # may skip vast section of b
    return i if i <= k and b[i] == v else -1
    # post: b[n] = v and v not in b[h..n-1], or, n = -1 and v not in b[h..k]
```

Sorting: Selection Sort

	0	n
pre: b	?	

	0	n
post: b	sorted	

Selection Sort:

	0	i	n
inv: b	sorted, ≤ b[i..]	≥ b[0..i-1] or ? if i = 0	

INITIALIZE AND COMPLETE

```
def sort(b):
    """Sort list b in place."""
    ...
    while ...:
        j = min_index(b, i, n)
        # b[j] is min item in b[i..n-1]
        ...
```

	i	n
2 4 4 6 6	8 9 9 7 8 9	

Note the swap of the reds

	i	n
2 4 4 6 6	7 9 9 8 8 9	

	i	n
2 4 4 6 6 7	9 9 8 8 9	

```
def selection_sort(b):
    """Sort list b in place."""
    n = len(b)
    i = ?
    # inv: b[0..i-1] sorted;
    #      b[i..n-1] >= b[0..i-1]
    while ?
        j = min_index(b, i, n)
        # b[j] is min item in b[i..n-1]
        ?, ? = ?, ?
        i += 1
    # post: b[0..n-1] sorted
```

	0	i	n
inv: b	sorted, ≤ b[i..]	≥ b[0..i-1] or ? if i = 0	

Helper for Selection Sort

```
def min_index(b, i, n):
    """return index of min item in b[i...n-1]. Pre: i <= n."""
    j = ?
    k = ?
    # inv: ?
    while ?
        j = ?
        k += 1
    # post: ?
```

	i	k	n
inv: b	b[j] is min of these	?	

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"

	h			k
pre: b	?			

(values in h..k are unknown)

	h			k
post: b	<0	=0	>0	

	h	t	i	j	k
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

```
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)
```

Dutch National Flag Algorithm

```
def dnf(b, h, k):
    """Returns: partition indices as a tuple (i, j)"""
    t = h; i = k+1, j = k;
    # 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:
            b[i-1], b[t] = b[t], b[i-1]
            t = t+1
        elif b[i-1] == 0:
            i = i-1
        else:
            b[-1], b[j] = b[j], b[i-1
            i = i-1; j = j-1
    # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    return (i, j)
```

< 0		?			= 0		> 0	
h		t			i	j		k
-1	-2	3	-1	0	0	0	6	3

h		t		i		j		k
-1	-2	3	-1	0	0	0	6	3

h			t	i		j		k
-1	-2	-1	3	0	0	0	6	3

h			t			j		k
-1	-2	-1	0	0	0	3	6	3