http://www.cs.cornell.edu/courses/cs1110/2019sp

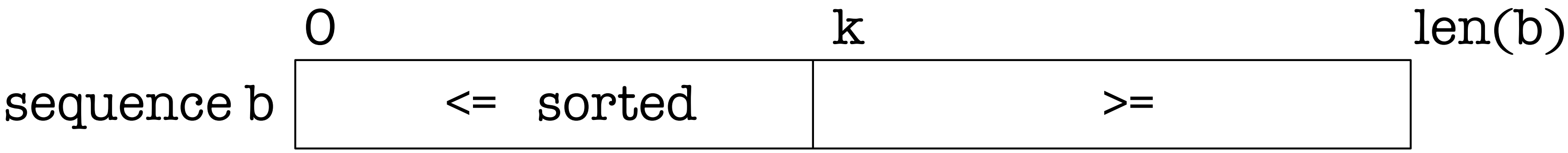
Lecture 25: Sequence Algorithms

CS 1110

Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]

Box Notation for Sequences



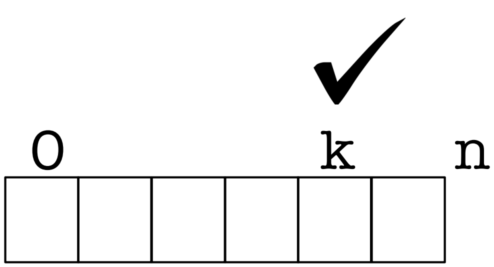
Graphical assertion about sequence b. It asserts that:

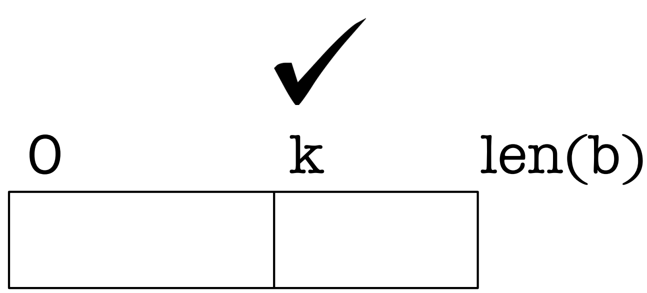
1. b[0..k–1] is sorted (values are in ascending order)
2. all of b[0..k–1] is $\leq$ all of b[k..len(b)–1]

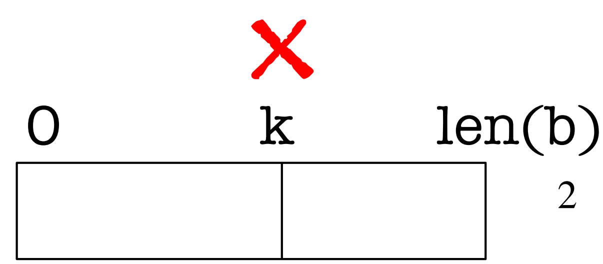
Pro Tip #1:

index always goes *above a box*, never above a line

(just like house numbers go on a house not between the houses)

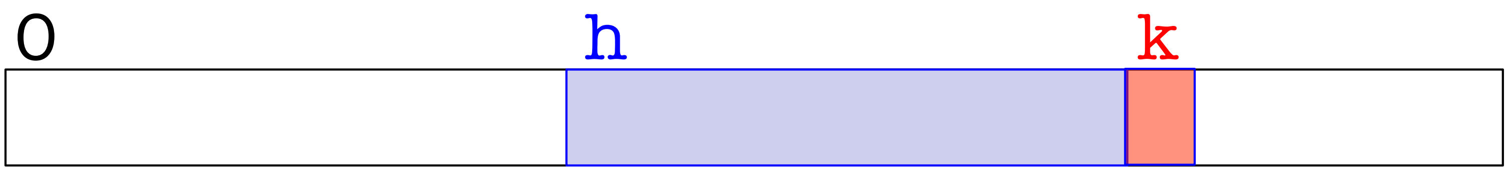






Q: Indices for Box Notation

sequence b



Given:

- index **h** of the first element of a segment
- index **k** of the element that follows that segment,

Questions:

1. How many values are in segment b[h .. k – 1]
2. How many values are in b[h .. h – 1] ?
3. How many values are in b[h .. h + 1] ?

A: 0
B: 1
C: 2
D: k - h
E: k + h

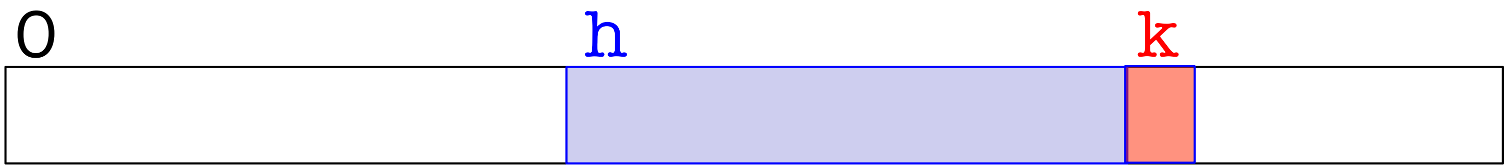
Pro Tip #2:

Size is "Follower minus First"

Follower: next thing outside the specified range

Clicker Answer: Indices for Box Notation

sequence b



Given:

- index **h** of the first element of a segment
- index **k** of the element that follows that segment,

Questions:

1. How many values are in segment `b[h .. k – 1]` **D**
2. How many values are in `b[h .. h – 1]` ? **A**
3. How many values are in `b[h .. h + 1]` ? **C**

A: `0`
B: `1`
C: `2`
D: `k - h`
E: `k + h`

count num adjacent equal pairs

Approach #1: compare s[k] to the character in front of it (s[k-1])

```
# set n_pair to # adjacent equal pairs in s


n_pair = 0
k = 1

while  k < len(s):
    if s[k-1] == s[k]:
        n_pair += 1
    k = k + 1
```

count num adjacent equal pairs

Approach #1: compare s[k] to the character in front of it (s[k-1])

```
# set n_pair to # adjacent equal pairs in s
```

0 n

pre: seq s | ? (unknown values) | n >= 0, n_pair = 0

```
n_pair = 0
k = 1
```

0 k n

INV: seq s | processed | ?(unknown) | n_pair = num adjacent pairs in s[0..k-1]

```
while  k < len(s):
    if s[k-1] == s[k]:
        n_pair += 1
    k = k + 1
```

0 n

post: seq s | processed | n_pair = num adjacent pairs in s[0..n-1]

find the max of a seq

Task: find the maximum of a sequence s

```
k = 1
big = s[0]

while  k < len(s):
    big = max(big, s[k])
    k = k + 1
```

find the max of a seq

Task: find the maximum of a sequence s

big is the max of this segment (s[0])

0 n

pre: s | | *? (unknown values)* | n > 0, big = s[0]

```
k = 1
big = s[0]
```

0 k n

inv: s | *big is max of this segment* | ? | n > 0, big = s[0..k-1]

```
while  k < len(s):
    big = max(big, s[k])
    k = k + 1
```

0 n

post: s | *big is the max of this segment* | k = n, big = max of s[0..n-1]

Developing Algorithms on Sequences

- Specify the algorithm by giving its precondition and postcondition as pictures.
- Draw the invariant by drawing another picture that "moves from" the precondition to the postcondition
  - The invariant is true at the beginning and at the end
- The four loop design questions
  1. How does loop start (how to make the invariant true)?
  2. How does it stop (is the postcondition true)?
  3. How does the body make progress toward termination?
  4. How does the body keep the invariant true?

Invariants: separate + from – in a list

Task: Put negative values before nonnegative ones and return the split index

5	-7	2	2	-1	8	-3	9	3

-7	-1	-3	2	5	8	2	9	3

`k = 3`

Invariants: separate + from – in a list

Task: Put negative values before nonnegative ones and return the split index

5	-7	2	2	-1	8	-3	9	3

-7	-1	-3	2	5	8	2	9	3

k = 3

pre: s — 0 … n: ? *(unknown)* — n >= 1

```
k = 0
j = n
```

inv: s — 0 … k→ … ←j … n: < 0 | ? | >= 0

s[0..k-1] negative
s[j..n-1] zero or +
s[k..j-1] unknown

```
while  k < j:

    <body goes here>
```

post: s — 0 … k … n: < 0 | >= 0 — k = j

x — Shaded elements have been processed

High Level Approach

-7	5	2	2	-1	8	-3	-9	3	
k									j

inspect s[k]: stays where it is
→ just increment k

-7	5	2	2	-1	8	-3	-9	3	
	k								j

inspect s[k]: needs to be moved
inspect s[j-1]: stays where it is
→ just decrement j

-7	5	2	2	-1	8	-3	-9	3
	k							j

inspect s[k]: needs to be moved
inspect s[j-1]: needs to be moved
→ swap the elements,
increment k, decrement k

-7	-9	2	2	-1	8	-3	5	3
		k					j	

inspect s[k]: needs to be moved
inspect s[j-1]: needs to be moved
→ swap the elements,
increment k, decrement k

-7	-9	-3	2	-1	8	2	5	3
			k			j		

Body: separate + from – in a list

```
k = 0
j = n
```

inv: s

0	k→ ←j	n
< 0	?	>= 0

s[0..k-1] negative
s[j..n-1] zero or +
s[k..j-1] unknown

```
while  k < j:
    if s[k] < 0: # kth elem stays where it is
        k = k + 1
    elif s[j-1] >= 0: # (j-1)th elem stays where it is
        j = j - 1
    else: # both elements in the wrong place
    swap(s, k, j-1)
        k = k + 1
        j = j - 1
```

post: s

0	k	n
< 0	>= 0	

k = j