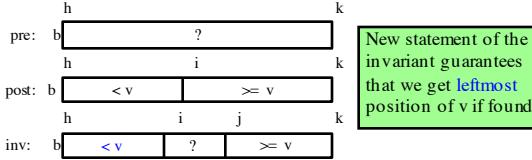


Binary Search

- Look for value v in **sorted** segment b[h..k]

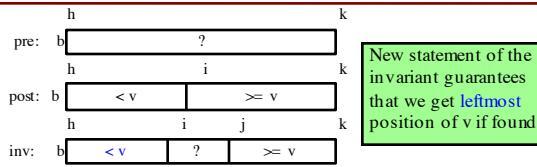


Example b [3 3 3 3 4 4 6 7 7]

- if v is 3, set i to 0
- if v is 4, set i to 5
- if v is 5, set i to 7
- if v is 8, set i to 10

h k
0 1 2 3 4 5 6 7 8 9

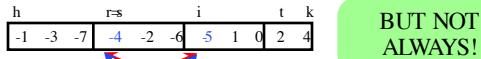
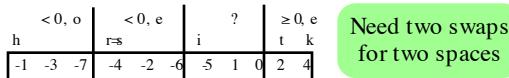
Binary Search



i = h; j = k+1;
while i != j:

Looking at b[i] gives **linear search from left**.
Looking at b[j-1] gives **linear search from right**.
Looking at middle: b[(i+j)/2] gives **binary search**.

Flag of Mauritius

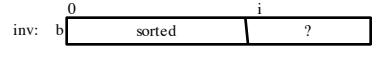


Have to check if second swap is okay

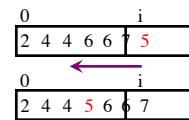
Sorting: Arranging in Ascending Order



Insertion Sort:

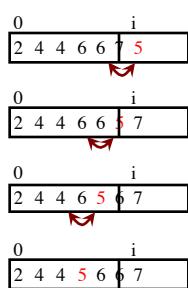


i = 0
while i < n:
Push b[i] down into its
sorted position in b[0..i].
i = i+1



Insertion Sort: Moving into Position

```
i = 0
while i < n:
| push_down(b,i)
| i = i+1
```



```
def push_down(b, i):
| j = i
| while j > 0:
| | if b[j-1] > b[j]:
| | | swap(b[j-1],j)
| j = j-1
```

swap shown in the lecture about lists

Insertion Sort: Performance

```
def push_down(b, i):
    """Push value at position i into sorted position in b[0..i-1]"""
    j = i
    while j > 0:
        if b[j-1] > b[j]:
            swap(b[j-1],j)
        j = j-1
```

Insertion sort is an n^2 algorithm

- b[0..i-1]: i elements
- Worst case:
 - i = 0: 0 swaps
 - i = 1: 1 swap
 - i = 2: 2 swaps
- Pushdown is in a loop
 - Called for i in 0..n
 - i swaps each time

Total Swaps: $0 + 1 + 2 + 3 + \dots + (n-1) = (n-1)*n/2$

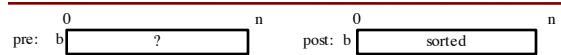
Algorithm “Complexity”

- Given: a list of length n and a problem to solve
- Complexity: rough number of steps to solve worst case
- Suppose we can compute 1000 operations a second:

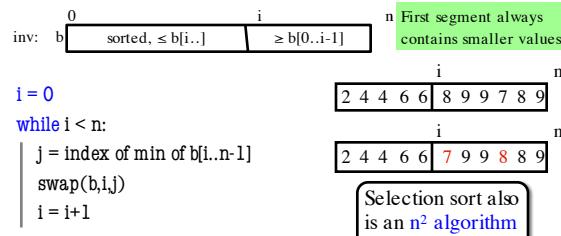
Complexity	$n=10$	$n=100$	$n=1000$
n	0.01 s	0.1 s	1 s
$n \log n$	0.016 s	0.32 s	4.79 s
n^2	0.1 s	10 s	16.7 m
n^3	1 s	16.7 m	11.6 d
2^n	1 s	4×10^{19} y	3×10^{90} y

Major Topic in 2110: Beyond scope of this course

Sorting: Changing the Invariant

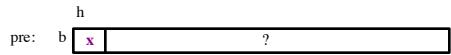


Selection Sort:



Partition Algorithm

- Given a list segment $b[h..k]$ with some value x in $b[h]$:



- Swap elements of $b[h..k]$ and store in j to truthify post:

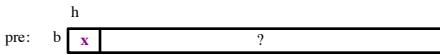


change:
into
or

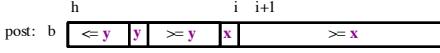
b	$3 \ 5 \ 4 \ 1 \ 6 \ 2 \ 3 \ 8 \ 1$	$\bullet \ x$ is called the pivot value
b	$1 \ 2 \ 1 \ 3 \ 5 \ 4 \ 6 \ 3 \ 8$	$\bullet \ x$ is not a program variable
b	$1 \ 2 \ 3 \ 1 \ 3 \ 4 \ 5 \ 6 \ 8$	$\bullet \ x$ denotes value initially in $b[h]$

Sorting with Partitions

- Given a list segment $b[h..k]$ with some value x in $b[h]$:



- Swap elements of $b[h..k]$ and store in j to truthify post:



Partition Recursively

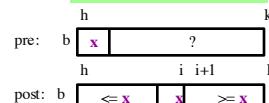
Recursive partitions = sorting

- Called **QuickSort** (why???)
- Popular, fast sorting technique

QuickSort

```
def quick_sort(b, h, k):
    """Sort the array fragment b[h..k]"""
    if b[h..k] has fewer than 2 elements:
        return
    j = partition(b, h, k)
    # b[h..j-1] <= b[j] <= b[j..k]
    # Sort b[h..j-1] and b[j..k]
    quick_sort(b, h, j-1)
    quick_sort(b, j+1, k)
```

- Worst Case:** array already sorted
 - Or almost sorted
 - n^2 in that case
- Average Case:** array is scrambled
 - $n \log n$ in that case
 - Best sorting time!



Final Word About Algorithms

Algorithm:

- Step-by-step way to do something
- Not tied to specific language

List Diagrams

Implementation:

- An algorithm in a specific language
- Many times, not the “hard part”

Demo Code

Higher Level Computer Science courses:

- We teach advanced algorithms (pictures)
- Implementation you learn on your own