Binary Search

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

<table>
<thead>
<tr>
<th>pre: $b$</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>post: $b$</td>
<td>$&lt;$ $v$</td>
</tr>
<tr>
<td>inv: $b$</td>
<td>$&lt;$ $v$</td>
</tr>
</tbody>
</table>

Example $b$: 0 1 2 3 3 3 4 4 4 6 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

New statement of the invariant guarantees that we get leftmost position of $v$ if found.

Insertion Sort: Moving into Position

- $b[0..i-1]$: $i$ 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: $0 + 1 + 2 + 3 + \ldots (n-1) = (n-1)n/2$
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:

<table>
<thead>
<tr>
<th>Complexity</th>
<th>n=10</th>
<th>n=100</th>
<th>n=1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0.01 s</td>
<td>0.1 s</td>
<td>1 s</td>
</tr>
<tr>
<td>n log n</td>
<td>0.016 s</td>
<td>0.32 s</td>
<td>4.79 s</td>
</tr>
<tr>
<td>n^2</td>
<td>0.1 s</td>
<td>10 s</td>
<td>16.7 m</td>
</tr>
<tr>
<td>n^3</td>
<td>1 s</td>
<td>16.7 m</td>
<td>11.6 d</td>
</tr>
<tr>
<td>2^n</td>
<td>1 s</td>
<td>4x10^69 y</td>
<td>3x10^80 y</td>
</tr>
</tbody>
</table>

Major Topic in 2110: Beyond scope of this course

Sorting: Changing the Invariant

pre: b[] <= x <= x post: b[] <= x <= x

i = 0
while i < n:
    j = index of min of b[i..n-1]
    swap(b, i, j)
    i = i+1

Selection Sort:

2  4  4  6  6  8  9  9  7  8  9

First segment always contains smaller values

Major Topic in 2110: Beyond scope of this course

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:

<table>
<thead>
<tr>
<th>Change:</th>
<th>into</th>
<th>or</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>3 5 4 1 6 2 3 8 1</td>
<td>b 1 2 3 1 3 4 5 6 8</td>
</tr>
<tr>
<td>h</td>
<td>i</td>
<td>k</td>
</tr>
<tr>
<td>j</td>
<td>1 2 1 3 4 6 3 8</td>
<td></td>
</tr>
</tbody>
</table>

x is called the pivot value
x is not a program variable
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:

<table>
<thead>
<tr>
<th>Change:</th>
<th>into</th>
<th>or</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>&lt;= x</td>
<td>&gt;= x</td>
</tr>
<tr>
<td>h</td>
<td>i+1</td>
<td>k</td>
</tr>
<tr>
<td>j</td>
<td>1 2 3 1 3 4 5 6 8</td>
<td></td>
</tr>
</tbody>
</table>

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+1..k]
    # Sort b[h..j-1] and b[j+1..k]
    quick_sort(b, h, j-1)
    quick_sort(b, j+1, k)
```

Worst Case:
array already sorted
Or almost sorted

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
- Implementation:
  - An algorithm in a specific language
  - Many times, not the “hard part”
- Higher Level Computer Science courses:
  - We teach advanced algorithms (pictures)
  - Implementation you learn on your own

List Diagrams
Demo Code