Linear Search

**pre:** b

**post:** b

**inv:** b

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

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.

Binary Search

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

**pre:** b

**post:** b

**inv:** b

Example b

12/1/19

Insertion Sort: Moving into Position

i = 0

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

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

Insertion Sort: Performance

def push_down(b, i):

• 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: $0 + 1 + 2 + 3 + \ldots + (n-1) = \frac{(n-1)n}{2}$
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>4x10^9 y</td>
<td>3x10^99 y</td>
</tr>
</tbody>
</table>

Major Topic in 2110: Beyond scope of this course

Sorting: Changing the Invariant

<table>
<thead>
<tr>
<th>pre:</th>
<th>post:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ![ sorted, x b[0..i-1] ]</td>
<td>n First segment always contains smaller values</td>
</tr>
</tbody>
</table>

Selection Sort:

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

Selection sort also is an \( n^2 \) algorithm

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:

```python
def quick_sort(b, h, k):
    """Sort the array fragment [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)
```

QuickSort

- 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

- **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