Search as in problem set: b is sorted

<table>
<thead>
<tr>
<th>pre: b</th>
<th>post: b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ? b.length</td>
<td>0 &lt;= v &gt; v</td>
</tr>
</tbody>
</table>

$
\begin{align*}
\text{inv: } & 0 \leq h < t \\
\text{Methodology: } & \text{1. Draw the invariant as a combination of pre and post} \\
\text{2. Develop loop using 4 loopy questions.} \\
\text{Practice doing this!}
\end{align*}
$

Search as in problem set: b is sorted

<table>
<thead>
<tr>
<th>pre: b</th>
<th>post: b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ? b.length</td>
<td>0 &lt;= v &gt; v</td>
</tr>
</tbody>
</table>

$
\begin{align*}
\text{inv: } & 0 \leq h < t \\
\text{Methodology: } & \text{1. Draw the invariant as a combination of pre and post} \\
\text{2. Develop loop using 4 loopy questions.} \\
\text{Practice doing this!}
\end{align*}
$

Binary search: an $O(\log n)$ algorithm

<table>
<thead>
<tr>
<th>pre: b</th>
<th>post: b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 h</td>
<td>e = (h + t) / 2; n = 2**k</td>
</tr>
</tbody>
</table>

$
\begin{align*}
\text{inv: } & b \leq v < e \\
\text{Each iteration cuts the size of the } & \text{? segment in half.}
\end{align*}
$

Miscellaneous

- Pinned Piazza note on Supplemental study material. @472. Contains material that may help you study certain topics. It also talks about how to study.
Looking at execution speed  
Process an array of size n

<table>
<thead>
<tr>
<th>Number of operations executed</th>
<th>2n+2 ops, n+2, n are all “order n” O(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Called linear in n, proportional to n</td>
</tr>
<tr>
<td></td>
<td>n*n ops</td>
</tr>
<tr>
<td></td>
<td>Constant time</td>
</tr>
<tr>
<td></td>
<td>size n</td>
</tr>
</tbody>
</table>

```
InsertionSort
```

**Pre:** b[i..b.length]

```
inv: b                 post: b sorted
0                b.length
```

**Inv:** b[i..b.length]

```
for (int i= 0; i < b.length; i= i+1) { maintain invariant }
```

Each iteration, i= i+1; How to keep inv true?

```
inv: b sorted i b.length
0 i b.length
```

```
e.g. b
0 2 5 5 5 3
```

```
b 2 3 5 5 5
```

Push b[i] down to its shortest position in b[0..i], then increase i

Will take time proportional to the number of swaps needed

```
What to do in each iteration?
```

```
inv: i
0 i b.length
```

```
e.g. b
0 2 5 5 5 3
```

```
Loop body
inv true before and after
```

```
0 2 5 5 5 3
```

```
k= k–1;
```

```
K: b[0..i] is sorted
Push b[i] down to its sorted position in b[0..i], then increase i
```

```
start? stop? progress?
```

```
inv: b[k..b.length]
```

```
start? stop? progress?
```

```start
```
How to write nested loops

```java
// sort b[], an array of int
// inv: b[0..i-1] is sorted
for (int i = 0; i < b.length; i++) {
  Push b[i] down to its sorted position in b[0..i]
}
```

If you are going to show implementation, put in the "WHAT TT DO" as a comment.

Present algorithm like this.

```java
while (k > 0 && b[k] < b[k-1]) {
  k = k - 1;
// sort b[], an array of int
// inv: b[0..i-1] is sorted
for (int i = 0; i < b.length; i++) {
  Push b[i] down to its sorted position in b[0..i]
}
```

InsertionSort

```java
// sort b[], an array of int
// inv: b[0..i-1] is sorted
for (int i = 0; i < b.length; i++) {
  Push b[i] down to its sorted position in b[0..i]
}
```

InsertionSort

```java
// sort b[], an array of int
// inv: b[0..i-1] is sorted
for (int i = 0; i < b.length; i++) {
  Push b[i] down to its sorted position in b[0..i]
}
```

SelectionSort

```java
// sort b[], an array of int
// inv: b[0..i-1] sorted AND b[0..i-1] <= b[i..]
for (int i = 0; i < b.length; i++) {
  int m = index of minimum of b[i..];
  Swap b[i] and b[m];
}
```

SelectionSort

```java
// sort b[], an array of int
// inv: b[0..i-1] sorted AND b[0..i-1] <= b[i..]
for (int i = 0; i < b.length; i++) {
  int m = index of minimum of b[i..];
  Swap b[i] and b[m];
}
```

Swapping b[i] and b[m]

```java
// Swap b[i] and b[m]
int t = b[i];
b[i] = b[m];
b[m] = t;
```

Swapping b[i] and b[m]

```java
// Swap b[i] and b[m]
int t = b[i];
b[i] = b[m];
b[m] = t;
```

Partition algorithm of quicksort

```java
// Swap array values around until b[h..k] looks like this:
pre: h h+1 ... k
post: <= x x >= x
```
Partition algorithm

Initially, with \(j = h\) and \(t = k\), this diagram looks like the start diagram.

Terminate when \(j = t\), so the "?" segment is empty, so diagram looks like result diagram.

QuickSort procedure

Worst case quicksort: pivot always smallest value

QuickSort

QuickSort developed by Sir Tony Hoare (he was knighted by the Queen of England for his contributions to education and CS).

81 years old.

Developed QuickSort in 1958. But he could not explain it to his colleague, so he gave up on it.

Later, he saw a draft of the new language Algol 58 (which became Algol 60). It had recursive procedures. First time in a procedural programming language. "Ah!" he said. "I know how to write it better now." 15 minutes later, his colleague also understood it.
**Best case quicksort: pivot always middle value**

<table>
<thead>
<tr>
<th>&lt;= x0</th>
<th>x0</th>
<th>&gt;= x0</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= x1</td>
<td>x1</td>
<td>&gt;= x0</td>
</tr>
<tr>
<td>&lt;= x1</td>
<td>x1</td>
<td>&lt;= x2</td>
</tr>
</tbody>
</table>

Depth 0: 1 segment of size \(-n\) to partition.

Depth 2: 2 segments of size \(-n/2\) to partition.

Depth 3: 4 segments of size \(-n/4\) to partition.

Max depth: about \(\log n\). Time to partition on each level: \(-n\)

Total time: \(O(n \log n)\).

Average time for Quicksort: \(n \log n\). Difficult calculation

**QuickSort procedure**

```java
/** Sort b[h..k]. */
public static void QS(int[] b, int h, int k) {
    if (b[h..k] has < 2 elements) return;
    int j = partition(b, h, k);
    // We know b[h...j-1] <= b[j] <= b[j+1..k]
    // Sort b[h...j-1] and b[j+1..k]
    QS(b, h, j-1);
    QS(b, j+1, k);
}
```

Worst-case: quadratic

Average-case: \(O(n \log n)\)

Worst-case space: \(O(n^2)\) -- depth of recursion can be \(n\)

Can rewrite it have space \(O(\log n)\)

Average-case: \(O(n \log n)\)

**Partition algorithm**

**Key issue:** How to choose a pivot?

- **Choosing pivot**
  - Ideal pivot: the median, since it splits array in half
  - But computing median of unsorted array is \(O(n)\), quite complicated
  - Popular heuristics: Use
    - first array value (not good)
    - middle array value
    - median of first, middle, last, values GOOD!
    - Choose a random element

**Quicksort with logarithmic space**

Problem is that if the pivot value is always the smallest (or always the largest), the depth of recursion is the size of the array to sort.

Eliminate this problem by doing some of it iteratively and some recursively.

```java
/** Sort b[h..k]. */
public static void QS(int[] b, int h, int k) {
    int h1 = h; int k1 = k;
    // invariant b[h..k] is sorted if b[h1..k1] is sorted
    while (b[h1..k1] has more than 1 element) {
        // Reduce the size of b[h1..k1], keeping inv true
        ...
    }
}
```

**QuickSort with logarithmic space**

Problem is that if the pivot value is always the smallest (or always the largest), the depth of recursion is the size of the array to sort.

Eliminate this problem by doing some of it iteratively and some recursively. We may show you this later. Not today!
QuickSort with logarithmic space

```java
/** Sort b[h..k]. */
public static void QS(int[] b, int h, int k) {
    int h1 = h, k1 = k;
    // invariant b[h..k] is sorted if b[h1..k1] is sorted
    while (b[h1..k1] has more than 1 element) {
        int j = partition(b, h1, k1);
        // b[h1..j-1] <= b[j] <= b[j+1..k1]
        if (b[h1..j-1] smaller than b[j+1..k1]) {
            QS(b, h, j-1); h1 = j+1;
        } else {
            QS(b, j+1, k1); k1 = j-1;
        }
    }
}
```

Only the smaller segment is sorted recursively. If b[h1..k1] has size n, the smaller segment has size < n/2. Therefore, depth of recursion is at most log n

Binary search: find position h of v = 5

<table>
<thead>
<tr>
<th>pre: array is sorted</th>
<th>t = 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>h = -1</td>
<td>t = 11</td>
</tr>
<tr>
<td>1 4 4 5 6 6 8 8 10 11 12</td>
<td></td>
</tr>
<tr>
<td>h = -1</td>
<td>t = 5</td>
</tr>
<tr>
<td>1 4 4 5 6 6 8 8 10 11 12</td>
<td></td>
</tr>
<tr>
<td>h = 2</td>
<td>t = 5</td>
</tr>
<tr>
<td>1 4 4 5 6 6 8 8 10 11 12</td>
<td></td>
</tr>
<tr>
<td>h = 3</td>
<td>t = 5</td>
</tr>
<tr>
<td>1 4 4 3 6 6 8 8 10 11 12</td>
<td></td>
</tr>
<tr>
<td>h = 3</td>
<td>t = 4</td>
</tr>
<tr>
<td>1 4 4 3 6 6 8 8 10 11 12</td>
<td></td>
</tr>
</tbody>
</table>

post: <= v h > v

Loop invariant:
- b[0..h] <= v
- b[t..] > v
- B is sorted