**Searching a binary tree**

```java
public BinaryTreeNode search(BinaryTreeNode n, Comparable obj) {
    int comparison;
    while (n != null) {
        comparison = obj.compareTo(n.key);
        switch(comparison) {
            case -1:  // obj less than node key
                n = n.left;
                break;
            case  0:  // obj equals node key
                return n;
                break;
            case  1:  // obj greater than node key
                n = n.right;
        }
    }
    return null;
}
```

**Complexity of searching a binary tree**

- Best case: find item in root
- Worst case: not in tree, or in one of the leaves
- Average case: ???

**Searching a binary tree: average case**

- Complexity depends on the shape of the tree
- In general, complexity of finding a node at position $p$ in the tree is the path length to $p$ plus 1
- For average complexity, need to compute the *internal path length (IPL)*
  - sum of all path lengths in the tree
“Flat” tree

- This is the worst case
- Flat tree = linked list
- $IPL = 0 + 1 + 2 + \ldots + n = n(n+1)/2$
- Average complexity = $IPL/n \Rightarrow O(n)$

Complete binary tree

- This is the best case
- $IPL = 0 + (1)(2) + (2)(4) + (3)(8) + \ldots + N(2^N)$
- Simplifies to $(h-2)2^h + 2$
- Number of nodes ($N$) in a complete tree $= 2^h - 1$
- Average complexity = $IPL/N \approx h - 2$
- $h = \lg(n + 1)$
- $\Rightarrow O(\lg n)$

Other binary trees

- Average complexity of searching a binary tree is between $(n-1)/2$ and $\lg(n+1) - 2$
- Formula:
  - $(2/n^2) \sum_{i=1}^{n-1} i (\text{path}_i + 1)$
  - reduces to $1.4 \lg n \Rightarrow O(\lg n)$
- So in general, average complexity for searching any binary tree is $O(\lg n)$

Traversing a binary tree

- Recall that traverse means “visit every node in the tree exactly once”
- Q: How many ways can you come up with to traverse a binary tree?