The Binary Tree

- **Def.** A **binary tree** is a finite set, call it T, that is either empty or else has its elements partitioned in 3 distinguished subsets:
  - a singleton subset containing the root of T,
  - a left subtree, which is itself a binary tree,
  - a right subtree, which is again a binary tree.
- The objects of T are called nodes.

Parent - Child

- If x and y are nodes such that y is the root of the left subtree of the tree rooted at x, the we say that y is the **left child** of x and x is the **parent** of y.
- Similar definition for the **right child**.
- The **descendants** of a node x are all the nodes that make up the left and right subtrees.
- The **ancestors** of a node x are all the nodes in the tree for which x is a descendant.
- The root of the tree has no ancestors.
- A node with no descendant is called a **leaf node**.
**Tree Properties**

- The *height* of a tree is equal to
  - $-1$ if the tree is empty
  - $1 + \max(\text{height(left subtree)}, \text{height(right subtree)})$.
- The *level* of a node is equal to
  - $0$ if the node is the root of the tree
  - $1 + \text{level(parent)}$ for any other node.
- The height of the tree is the maximal level of any node in that tree.

---

**Minimum Height**

- What is the minimum height of a tree with $n$ nodes?
- If we denote that by $M(n)$, then we have
  - $M(n) = 1 + \min_{0 \leq i < n}\{\max(M(i), M(n-1-i))\}$
- From this we can deduce that
  - $M(n) = 1 + M(\lfloor n/2 \rfloor)$
  - $M(n) = \lceil \lg n \rceil$
**Number of Nodes**

- What is the minimum/maximum number of nodes in a tree of height \( h \)?
- Min: \( n(h) = h+1 \)
- Max: \( N(-1) = 0 \)
- \( N(h) = 1 + 2 \cdot N(h-1) \)
- \( N(h) = 2^{h+1} - 1 \)
- Number of empty subtrees: \( n+1 \).

**Number of Leaves**

- Minimum: 1 leaf.
- Maximum number of leaves in a tree with \( n \) nodes?
- Inverse problem: what is the minimum number of nodes in a tree with \( \lambda \) leaves?
- \( \lambda \) leaves \( \Rightarrow \) \( 2\lambda \) empty subtrees at least \( \Rightarrow \) at least \( 2\lambda - 1 \) nodes in the tree.
- \( n \geq 2\lambda - 1 \iff \lambda \leq (n+1)/2 \Rightarrow \)
- \( \lambda \leq \left\lfloor (n+1)/2 \right\rfloor = \left\lceil n/2 \right\rceil \)
void print_in_preorder (node_ptr p) {
    if (p != NULL) {
        cout << p->datum << endl;
        print_in_preorder (p->left);
        print_in_preorder (p->right);
    }
}

Tree Traversal

Complexity

- **Number of runtime stack frames**: the maximum number at any time is equal to the height of the tree+2.
- **Extra space** required:
  - min: $\lfloor \log n \rfloor + 2$
  - max: $n + 1$
- **Time** (number of operations): a constant times the total number of function calls.
- **Number of calls**: the number of nodes + the number of empty subtrees = $2n+1$.
- Complexity is $\Theta(n)$. 

C455 Algorithms Analysis
**Height of the Tree**

```c
int height (node_ptr p)
{
    if (p == NULL)
        return -1;
    else if (height(p->left)<=height(p->right))
        return 1 + height(p->right);
    else
        return 1 + height(p->left);
}
```

**Complexity**

\[
K(T) = \begin{cases} 
1 & \text{if } T \text{ is empty (}n=0) \\
1 + K(L_T) + K(R_T) + K(\text{taller}(L_T,R_T)) & \text{if } T \text{ is not empty} \\
1 & \text{if } T \text{ is empty (}n=0) \\
1 + K(L_T) + 2K(R_T) & \text{if } R_T \text{ is taller than } L_T \\
1 + 2K(L_T) + K(R_T) & \text{otherwise} 
\end{cases}
\]

- In the worst case, if the tree is a string, and we denote the complexity by \( T(n) \), then we have
  - \( T(0) = 1 \)
  - \( T(n) = 2 \cdot T(n-1) + 2 \)
  - So \( T(n) = \Theta(2^n) \)
Binary Search Trees

- **Def.** A *binary search tree* is a binary tree with the following properties:
  - a) Each node carries one object of some type containing a "key" value that distinguishes it from all objects stored in the tree.
  - b) For each node N in the tree, all the keys in the left subtree of N are smaller than the key in N, and all the keys in the right subtree of N are larger than the key in N.

---

Search for a Key in a BST

```cpp
node_ptr location(node_ptr p, key_type target)
{
    while (p) {
        if (target == key(p->datum))
            return p;
        else if (target < key(p->datum))
            p = p->left;
        else
            p = p->right;
    }
    return NULL;
}
```

---

C455 Algorithms Analysis
Creating a New Node

// The node contains an integer
// datum and no label.
node_ptr new_node(otype x)
{
    node_ptr result = new node;
    result->left = NULL;
    result->right = NULL;
    result->datum = x;
    return result;
}

Insert a New Key in a BST

bool insert(node_ptr & p, otype x)
{
    if (!p) {
        p = new_node(key);
        return true;
    }
    else if (key(x) < key(p->datum))
        return insert(p->left, x);
    else if (key(x) > key(p->datum))
        return insert(p->right, x);
    else
        return false;
}
Reading From A File

while unread data remain in the file
{
    read one data object from the file into a temporary variable;
    insert the object into the binary search tree;
    advance to the next unread data object in the data file;
}

- File 1: Jill, Cody, Tina, Drew, Beth, Pete, Ruth
- File 2: Jill, Tina, Pete, Cody, Beth, Ruth, Drew
- File 3: Tina, Beth, Ruth, Cody, Pete, Drew, Jill

Height Balanced Trees

- **Def.** A binary tree is **height balanced** if for every node in the tree, the height of the left subtree and the right subtree is at most 1.
- Otherwise, we can define it recursively:
  - $|\text{height(left subtree)} - \text{height(right subtree)}| \leq 1$.
- the left subtree and the right subtree are both height balanced.
- A BST which is height-balanced is an AVL tree.
Height of Height Balanced Trees

- What is the minimum and maximum height of a height balanced tree with \( n \) nodes?
- Minimum: minimum height of a tree with \( n \) nodes, \( \lceil \lg n \rceil \).
- Let \( H(n) \) be the maximum height of a height-balanced tree.
- \( H(0) = -1, H(1) = 0, H(2) = 1 \), etc.
- \( H(n) = 1 + \max_{0 \leq i < n} \{H(i): i \text{ is such that } H(i) \text{ and } H(n-i-1) \text{ differ by at most 1} \} \)

Inverse Problem

- Given a height balanced tree of height \( h \), what is the minimum number of nodes in the tree, \( M(h) = n \)?
- \( M(0) = 1, M(1) = 2, M(2) = 1 + M(1) + M(0) \)
- In general we can write:
- \( M(h) = 1 + M(h-1) + M(h-2) \)
- \( M(h) = a((1+\sqrt{5})/2)^h + b((1-\sqrt{5})/2)^h + c \)
- \( h = \Theta(\lg n) \)
- This means that in general the height of a height balanced tree is \( \Theta(\lg n) \).
Comparison-Based Sorting Algorithms

template <class otype>
void linear_insertion_sort (otype a[], int n)
{
    for (int k = 2; k <= n; ++k)
    {
        otype temp = a[k];
        for (int i = k-1; i >= 1 &&
            a[i] > temp; --i)
        {
            a[i+1] = a[i];
            a[i+1] = temp;
        }
    }
}

Theorem

- **I1 Theorem.** For every comparison based sorting algorithm and every set of \( n \) distinct objects, there is a way of arranging the objects initially so that the number of object comparisons that will be made by the given algorithm while sorting the collection is at least \( \lceil \log(2(n!)) - 1 \rceil \).

- Consequently, if \( T_{\max}(n) \) denotes the maximum possible execution time for a comparison based sorting algorithm on an array of length \( n \), then \( T_{\max}(n) = \Theta(n \log n) \).