**Binary Search Tree (BST):**

A Binary Search Tree (BST) is a type of binary tree where each node follows a specific order: the left child has values smaller than its parent node, and the right child has values larger. This organization allows for efficient searching.

**YouTube Example (BST):**

Imagine YouTube uses a Binary Search Tree to organize videos by upload date:

* **Root Node**: Represents a video uploaded in January 2023.
* **Left Child**: Contains videos uploaded before January 2023, like December 2022.
* **Right Child**: Contains videos uploaded after January 2023, such as February 2023.

This structure enables YouTube to quickly search for videos by date, making it easy for users to find content from a specific time frame.

**Example**

A diagram of a number and a number

Description automatically generated with medium confidence

**Basic Operations on BST:**

* Insertion
* Searching
* Deletion
* Traversals – Inorder, Preorder, Post Order

In case of binary search trees (BST), **Inorder traversal** gives nodes in non-decreasing order. We visit the left child first, then the root, and then the right child.

## **Advantages of Binary Search Trees:**

1. Time Complexity:
2. Balanced Structure:
3. **Self-Balancing**:
4. **Height-Balanced**:
5. **Fast Searches**:
6. **Not Skewed**:

## **Disadvantages of Binary Search Trees:**

1. Unbalanced Trees:
2. Complex Balancing:
3. More Space Required:
4. Not Constant Time Access

**Unbalanced Trees**:

If you keep adding contacts to your phone in sorted order (like by last name), the contact list can become unbalanced. This means that when you try to search for a contact, the process can take longer, as the system might need to sift through many entries. As the list grows, the time it takes to find someone increases, leading to frustration. A balanced structure would help keep search times consistent and quick.

If you add items in sorted order, the tree can become unbalanced, making searches and updates slower.

**Complex Balancing**:

In a system that manages a playlist of songs, maintaining balance can be tricky if the list constantly changes. Each time a song is added or removed, the system might need to reorganize itself to keep everything efficient. This balancing act requires extra work and can lead to complications, especially if there are many songs. Users might experience delays or glitches during this process, impacting their enjoyment of the app.

Keeping the tree balanced can be complicated and requires extra work, especially with certain types of trees that adjust themselves.

**More Space Required**:

In a digital note-taking app, each note you create needs to link to other notes or tags for easy navigation. This means that more memory is required to store these connections, which can be an issue on devices with limited storage. As you create more notes, the space used increases significantly due to the additional information. Users may find themselves running low on space quicker than expected.

Binary search trees (BSTs) need more memory because they store extra information, like links to child nodes.

**Not Constant Time Access**:

When using an online shopping site, searching for a specific product often involves looking through multiple categories or filters. Unlike a system that allows instant access to items, you may need to check various sections before finding what you need. This process can be time-consuming and frustrating, especially if the site has many products. Quick access would enhance the shopping experience and make it more enjoyable.

Unlike hash tables, BSTs don’t allow for quick access to items. You often have to check multiple nodes to find what you need, which takes longer.

### Applications of AVL Trees

1. **Maps and Sets**: They help store key-value pairs and unique items, making it easy to find, add, or remove things quickly.
2. **Priority Queues**: They can be used to manage items based on priority, like organizing tasks that need to be done first.
3. **File Systems**: Help efficiently track files and folders.
4. **In-Memory Databases**: Facilitate fast data storage and retrieval.
5. **Graphics and Game Development**: Useful for handling tasks like detecting collisions between objects or finding paths in games.