Foundations Notes

Searching in Databases

- Searching is the most common operation performed by database systems
- The SELECT statement in SQL is particularly versatile and complex
- Linear search is the baseline for efficiency comparison:
    - Start at the beginning of a list and check each element sequentially
    - Continues until either the target is found or the list is exhausted

Database Records and Collections

- **Record**: A collection of values for attributes of a single entity instance (a row in a table)
- **Collection**: A set of records of the same entity type (a table)
- **Search Key**: A value for an attribute from the entity type
    - Could consist of one or more attributes

Storage Structures for Lists of Records

Memory Requirements

- If each record takes up x bytes of memory, n records require n*x bytes

Contiguously Allocated Lists (Arrays)

- All n*x bytes are allocated as a single "chunk" of memory
- Advantages: Fast random access
- Disadvantages: Slow for insertions anywhere except at the end

Linked Lists

- Each record needs x bytes plus additional space for memory addresses (pointers)
- Individual records are linked together in a chain using memory addresses
- Advantages: Fast for insertions anywhere in the list
- Disadvantages: Slow for random access

Binary Search Algorithm

- **Input**: Array of values in sorted order, target value

- **Output**: The location (index) of the target or an indicator it wasn't found
- Uses a divide-and-conquer approach by repeatedly dividing the search interval in half
- Implementation shown in Python in the slides

Time Complexity Comparison

Linear Search

- Best case: O(1) - target found at first element
- Worst case: O(n) - target not in array, requiring n comparisons

Binary Search

- Best case: O(1) - target found at midpoint
- Worst case: $O(\log_2 n)$ - target not in array
- Much more efficient for large datasets

Database Searching Challenges

- If data is stored on disk sorted by a primary key (like ID), searching by that key is fast
- Searching by other columns (like specialVal) requires a linear scan - inefficient
- Cannot efficiently store data sorted by multiple columns simultaneously
- Need for external data structures to enable efficient searches on secondary attributes

Potential Solutions

1. **Array of tuples (specialVal, rowNumber) sorted by specialVal**
    - Enables binary search for quick lookups
    - Problem: Insertions are slow due to the need to maintain sorted order
2. **Linked list of tuples (specialVal, rowNumber) sorted by specialVal**
    - Fast insertions
    - Problem: Searching requires a linear scan
3. **Binary Search Tree**
    - Proposed as a solution offering both fast insertion and fast search
    - A binary tree where every node in the left subtree is less than its parent
    - Every node in the right subtree is greater than its parent
    - Provides a compromise between the speed advantages of arrays and linked lists