Searching:

- Searching is the most common operation performed by a database system
- In SQL, the SELECT statement is arguably the most versatile / complex (they can be recursive and there can even be select statements in select statements)

General Vocab:

- Record: a collection of values for attributes of a single entity instance; a row of a table
- Collection: a set of records of the same entity type; a table (trivially stored in some sequential order like a list)
- Search Key: a value for an attribute from the entity type (could be >= 1 attribute)

id	specVal
1	55
2	87
3	50
4	108

- Assume data is stored on disk by column id's value, searching for a specific id is fast
- But searching for specific specialVal since data is unsorted, the only option is linear scan the column
- Can't store data on disk sorted by both id and specialVal at the same time so the data would have to be duplicated and there'd be inefficient space
    - Therefore we need an external data structure to support faster searching by specialVal than a linear search

- *What to do?*
    - An array of tuples (specialVal, rowNumber) sorted by specialVal
        - We could use Binary Search to quickly locate a particular specialVal and find its corresponding row in the table
        - But, every insert into the table would be like inserting into a sorted array - slow…
    - OR A linked list of tuples (specialVal, rowNumber) sorted by specialVal
        - searching for a specialVal would be slow - linear scan required
        - But inserting into the table would theoretically be quick to also add to the list… INSTEAD USE BINARY SEARCH TREE

Linear Search:

- Baseline for efficiency where you start at the beginning of a list and proceed element element by element until you either find what you're looking for or get to the last element and haven't found it aka O(n)
- If each record takes up x bytes of memory, then for n records, we need n*x bytes of memory
- There are 2 different Data Structures for Linear Search:
    1. Contiguously Allocated List (aka Array): all n*x are allocated as a single "chunk" of memory

        - Pro: Faster for random access
        - Con: Slow for inserting anywhere but the end
    2. Linked List: each record needs x bytes and additional space for 1 or 2 memory addresses - individual records are linked together in a type of chain using memory addresses
        - Pro: Faster for inserting anywhere in the list
        - Con: Slower for random access

- Best Case: target is found at the first element where only 1 comparison is needed
- Worst Case: target is not in the array; n comparisons - O(n) time complexity is for worst case

Binary Search:

- Input: array of values in sorted order, target value
- Output: location (index) of where target is located or some value indicating target was not found

- Best Case: target is found at mid; 1 comparison (inside the loop)
- Worst Case: target is not in the array; log2n comparisons - O(log2n) time complexity is for worse case

Binary Search Tree:

- A binary tree where every node in the left subtree is less than its parent and every node in the right subtree is greater than its parent