- Searching
    - Most common operation performed by database system
    - Baseline for efficiency is Linear Search
        - Start at beginning of list and then go through each element
            - Until you find desired elements
            - Get to end at don't find it
    - Record
        - Collection of values for attributes of a single entity instance (row of a table)
    - Collection
        - Set of records (table)
    - Search key
        - Id of what element you are searching for
- If each record takes up x bytes of memory, n*x bytes of memory needed
- Contiguously allocated list
    - All n*x bytes are allocated in a single 'chunk' of memory
- Linked list
    - Each record needs x bytes plus additional space for 1 or 2 memory addresses
    - Not continuous, jumping back and forth In the array in a particular order
- Arrays are faster for random access (lookup an element), but slow for inserting anywhere but the end
- Linked lists (similar to what lists are in python) are faster for inserting anywhere, but slower for random access (you have to go in the particular order that the lists (mini elements) are linked in)
- Binary Search
    - Input
        - Array of values in sorted order, target value
    - Output
        - Location (index) of where the target is located or some value indicating target wasn't found
    - Start in the middle of the array, check to see if it is less or more than target value, and move to that direction until you reach the value
    - Divide data by half each time you search, log base 2 of n
- Time Complexity
    - Linear search
        - Best case --> first element is correct, 1 comparison
        - Wors case --> not in the array, n comparisons O(n)
    - Binary search
        - Best case --> target is found at middle (1 comparison)
        - Worst case --> target isn't in array
        - Less efficient to linked lists with binary search
- Can't both store a sorted dataset by special id and val, linear not that efficient
- We need an external data structure

    - Array of tuples (special val, row
    - Need fast insert and fast search
        - Binary search tree
            - 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
            - Recursive data structure
            - What is used is most database structures when you are indexing
- Data:
    - A json
    - Preprocessed text
    - Organized by month
        - Many json files inside
- Key is a word, value is whichever articles have that word in it
- Can remove duplicate words
- Use string comparisons to compare words?
- Going over BST code
    - Done recursively
    - Insert a word, the value is the article/page associated with it
- Going over AVL Tree
    - Also inserts recursively
    - Same insert logic as BST
    - Check for balance and rotate
        - Rotate left ex:
            - Take out nodes you need for rotation
            - Reassign node that is being rotated to the left
            - Find the heights again
    - Ignore order traversal commented out code at the bottom
- Alphabetically

    - Transaction --> unit of work for a database
        - Sequence of operations that are performed as a unit
        - Either entire sequence succeeds or fails
    - Benefits of relational model
        - One standard language mostly
        - Works well with highly structured data
        - One relational database can handle a lot of data
        - Very well understood
    - Column oriented storage --> data stored by column, it's easier to look through a particular attribute with a lot of data
    - Isolation --> two transactions happening at the same time can't affect each other
        - Ensured through locking
        - Can lock for read/write, there are diff levels

- Dirty read
    - Transaction T2 is able to read a row that has been modified by another transaction T1 that hasn't yet executed a commit
- Non repeatable read
    - Two queries execute a single transaction but get different values because one has changed data
- Phantom reads
    - When a transaction adds or deletes rows from the data another transaction is using
- Durability
    - Once a transaction is completed and committed, changes are permanent, even in case of system failure

- Joins can be expensive --> also more tables while more normalized (space is cheaper than computing power) take more to bring back together
- Scalability
    - Scaling vertically means more power on one machine, it's easier than horizontal
- Distributed system
    - Computers that operate concurrently
    - Computers fail independently
    - No shared global clock
    - Appear to user as one computer
- Distributed storage
    - Replication --> make copies, if one dies you have backup
    - Sharding --> splitting data up by some variable, they operate on an independent system, controlled by some manager system
- Distributed data stores
    - Data stored on more than one node is typically replicated

- Acid transactions assumes that if something can go wrong it will, conflicts are prevented by locking resources until transaction is complete
- low conflict systems
    - Backups, analytical dbs
    - Optimistic concurrency
- NoSQL --> look at higher performance for certain tasks