Computer Memory Structure

1. Processor
    a. CPU
        i. Super fast
        ii. Super expensive
        iii. Tiny capacity
    b. CPU Cache (L1 - L3 Cache)
        i. Not as fast as CPU
        ii. Expensive
        iii. Small capacity
2. RAM
    a. Physical Memory (Random Access Memory)
        i. Not as fast as CPU cache
        ii. Priced reasonably
        iii. Average Capacity
3. Solid State Drives (Solid State Memory)
    a. Non-Volatile Flash-Based Memory
        i. Average Speed
        ii. Priced reasonable
        iii. Average capacity
4. Mechanical Hard Drives (Virtual Memory)
    a. File-Based Memory
        i. Slow
        ii. Cheap
        iii. Large Capacity

- For fast databases, we want to minimize access to solid state drives and hard drives
- A 64-bit integer takes up 8 bytes of memory
- Say you have a 2048 byte block size. To get 1, 64-bit (8 byte) integer out of memory, the DB has to read 2048 bytes.
- In an AVL node containing a key-value pair of two 64-bit integers, we are using 32 bytes of memory
    - 8 bytes for key, 8 bytes for value, 8 bytes for right child pointer, 8 bytes for left child pointer
    - Therefore, we are not optimizing the amount of memory we have available. We can increase the performance by decreasing the height of the tree
- Consider the following case: We have a sorted array of 128 integers

    - 128 * 8 = 1024 bytes
    - In the worst case, binary search o 128 integers is still much faster than a single additional disk access
    - Reading only one block of memory from a hard drive is significantly faster than reading for a second time
    - What if we could have two keys per node? Or three or four...?
        - As you add more keys to each node, the tree gets shallower (smaller height) and searching gets faster
        - The main point: in databases, we want to minimize hard drive access
- The ideal structure or databases and indexing are B+ trees
    - Each node can have up to 128, 256, etc values (keys) for indexing.
    - For n-1 keys, we can have n children
    - Searching for a child is still much faster than another disk read
    - This way, we can store more values and keys in much fewer levels
- The goal of indexing structures = minimize the height of the tree