A typical hierarchy

Every desktop/laptop/server is different but here is a plausible configuration these days

CPU

L1 Cache: 128KB = 2^17
  get data in L1: 2^22/sec = 2 insns

L2 Cache: 2MB = 2^21
  get data in L2: 2^25/sec = 30 insns

Main memory: 2GB = 2^31
  get data in main memory: 2^30/sec = 250 insns

Disk: 1TB = 2^40
  get data from "new place" on disk:
    2^18/sec =8,000,000 insns
    "streamed": 2^31/sec

Morals

It is much faster to do:
  5 million arithmetic ops
  2500 L2 cache accesses
  400 main memory accesses

Than:
  1 disk access
  1 disk access
  1 disk access

Why are computers built this way?
  – Physical realities (speed of light, closeness to CPU)
  – Cost (price per byte of different technologies)
  – Disks get much bigger not much faster
    • Spinning at 7200 RPM accounts for much of the slowness
      and unlikely to spin faster in the future
  – Speedup at higher levels makes lower levels relatively slower

Usually, it doesn’t matter . . .

The hardware automatically moves data into the caches from main memory for you
  – Replacing items already there
  – So algorithms much faster if “data fits in cache” (often does)

Disk accesses are done by software (e.g., ask operating system to open a file or database to access some data)

So most code “just runs” but sometimes it’s worth designing algorithms / data structures with knowledge of memory hierarchy
  – And when you do, you often need to know one more thing...

Connection to data structures

• An array benefits more than a linked list from block moves
  – Language (e.g., Java) implementation can put the list nodes anywhere, whereas array is typically contiguous memory

• Suppose you have a queue to process with 2^23 items of 2^7 bytes each on disk and the block size is 2^10 bytes
  – An array implementation needs 2^20 disk accesses
  – If “perfectly streamed”, > 4 seconds
  – If “random places on disk”, 8000 seconds (> 2 hours)
  – A list implementation in the worst case needs 2^23 “random” disk accesses (> 16 hours) – probably not that bad

• Note: “array” doesn’t mean “good”
  – Binary heaps “make big jumps” to percolate (different block)
BSTs?

- Looking things up in balanced binary search trees is $O(\log n)$, so even for $n = 2^{58}$ (512GB) we need not worry about minutes or hours.
- Still, number of disk accesses matters:
  - AVL tree could have height of 55
  - So each find could take about 0.5 seconds or about 100 finds a minute
  - Most of the nodes will be on disk: the tree is shallow, but it is still many gigabytes big so the tree cannot fit in memory
  - Even if memory holds the first 25 nodes on our path, we still need 30 disk accesses

Note about numbers; moral

- All the numbers in this lecture are “ballpark” “back of the envelope” figures
- Even if they are off by, say, a factor of 5, the moral is the same: If your data structure is mostly on disk, you want to minimize disk accesses.
- A better data structure in this setting would exploit the block size and relatively fast memory access to avoid disk accesses...