Algorithms

Overview

Algorithms

- A generic recipe for computation
  - Should work on broad category of computers
    - E.g. Algorithms for quantum computers, biological computers are / would be different

Standard Model of Computing

- What is presented to the programmer:
  - Computer reads instructions from memory
  - Computer acts on instructions by changing memory locations
    - Example: addi x, 5
      - Load x into accumulator, load 5 into a register, add results, move accumulator results back into memory where x is located

Standard Model of Computing

- Modern systems pretend that instructions are executed serially
  - Compilers move instructions around without telling
  - Compilers change instructions
  - Most instructions are not atomic
  - Caches allow two different threads to have different views of the memory contents
  - Memory system prioritizes reads over writes

Standard Model of Computing

- Contract between system and programmer:
  - System does what programmer wants, but in a different faster way
  - With a few exceptions, which makes multi-threaded computing so challenging

Standard Model of Computing

- Turns out that the optimizations of modern computing systems **do not** create genuine new capabilities
- We can ***emulate*** a modern system using an old one
- We can even ***emulate*** a modern system using a model of computing used in the 30s and 40s to model what Mathematics can compute:
  - Turing machine

DNA Computing

- DNA can store vast amounts of information in a very small space.
  - Store data (key-value pair) by encoding in DNA sub-sequences
  - To look up by key:
    - Introduce the compliment of the key's substring affixed to a magnetic bead
    - Compliment bonds to DNA molecules with that key
    - Extract these DNA molecules magnetically
    - Sequence them for the result

Quantum Computing

- Uses quantum phenomena for computing
    - Especially super-position and entanglement
  - Can be analog or digital
  - Digital quantum computing uses quantum gates
  - Difficulty now is getting up the number of q-bits in a system
- Could be faster than classical computers
  - Example: Shor's algorithm for factoring integers, Boson sampling
- Will almost certainly force current cryptography to use much larger keys

Algorithms

- Algorithms $\neq$ Implementation
  - An algorithm can be implemented more or less efficiently
  - You can measure the speed of an implementation on a given system fairly accurately
  - You can derive the performance of an algorithm using a computing model

Algorithms

- Correctness
  - Can we prove that the answer given by an algorithm is correct?
    - via Automated proof methods
    - via human reasoning
  - Often involves pseudo-code

Algorithms

- Performance
  - Needs to be measured independently of implementation
  - Depends on the "instance size"
    - Many problems in CS become proportionally more difficult as they grow
    - Use an "asymptotic" notation to capture behavior as we "scale up"

Performance

- Computing uses resources
  - Space: How much storage is needed
  - Time: How many instructions are needed
- But it becomes more interesting:
  - Some problems need to use storage (flash / disks)
    - Storage is much slower
    - Performance measurement: How many times does the algorithm need to access storage

Performance

- Parallel / Multi-threaded performance
  - Almost all computers have limited capability to execute instructions in parallel
  - E.g.: Develop data structures that are
    - thread-safe
    - lock-free (no locking of shared resources needed)
    - wait-free (no waiting for a thread to access a data structure)

Data Structures

- Way to organize data for algorithms
  - Correctness:
    - Provide a clearly defined interface
      - Abstract Data Structure
    - Provides capability to argue about programs
    - Allows independent development
      - Both are examples of the benefits of modularization

Data Structures

- An ADS is defined by its interface
- Possible to mathematically prove certain properties from the definition of the interface
    - In reality, mathematical proofs are rare
    - But they become more important when things become more difficult:
        - Arguing about thread safety

Data Structures

- Performance of ADT
  - Measured usually in time and space
  - Different implementations favor different operations
    - E.g. Inserts / Deletes at tail
      - If they are important: cyclic double linked
      - If they are not: single linked list
    - Inserts into a Python list
      - Fast at the end, slow at the beginning
      - Suffer if lists are large
      - Eventually, linked lists are better