Some of the fastest known algorithms for certain tasks rely on chance.

Stochastic/Randomized Algorithms

Two common variations
- Monte Carlo
- Las Vegas
- We have already encountered some of both in this class

Monte Carlo algorithms have a consistent time complexity, but there is a probability of returning an incorrect/poor answer.
- Can flexibly increase time to improve chance of a correct answer.

Las Vegas algorithms return a correct answer, but there is a probability that it could take a relatively long time or not return an answer at all.
- Why "Las Vegas"? - Because the house always wins (will get a correct answer for the house) even though it might take a while.
Monte Carlo Algorithm – Fermat Test

- If a number is prime then it will pass the Fermat primality test 
  \[ a^{c-1} \equiv 1 \mod c \] for a random \( a \) such that \( 1 \leq a < c \)
- There is less than a 50% probability that a composite number \( c \) passes the Fermat primality test
- So try the test \( k \) times
- This is called *Amplification of Stochastic Advantage*

```plaintext
function primality2(N)
Input: Positive integer N
Output: yes/no

Choose \( a_1 \ldots a_k \) \( (k < N) \) random integers between 1 and \( N-1 \)
if \( a_i^{N-1} \equiv 1 \mod N \) for all \( a_i \) then
  return yes with probability \( 1 - 1/(2^k) \)
else:
  return no
```
Dealing with Local Optima

- Assume an algorithm has a probability $p$ of finding the optimal answer for a certain problem
  - Just run it multiple times with different random start states
  - If chance of hitting a true optima is $p$, then running the problem $k$ times gives probability $1-(1-p)^k$ of finding an optimal solution
  - This is a Monte Carlo approach

- Another approach is to add some randomness to the algorithm and occasionally allow it to move to a neighbor which *increases* the objective, allowing it to potentially escape local optima
  - This (like GA) mixes both Monte Carlo and Las Vegas approaches
Monte Carlo Algorithms – General Approach

- Powerful approach for many tasks with no efficient deterministic algorithm
  - Especially useful in complicated tasks with a large number of coupled variables
  - Also useful when there is uncertainty in the inputs

1. Define a domain of possible inputs
2. Generate inputs randomly from the domain using an appropriate specified probability distribution (sampling)
3. Perform a deterministic computation using the sampled inputs
4. Aggregate the results of the individual computations into the final result
Monte Carlo Algorithm to Calculate $\pi$

Want to calculate the value of $\pi$ using darts and a blindfold.
Monte Carlo Algorithm to Calculate $\pi$

Want to calculate the value of $\pi$ using darts and a blindfold. Ratio of the area of the inscribed circle to the area of the square is

$$\frac{\pi r^2}{(2r)^2} = \frac{\pi}{4}$$

for every 4 darts thrown about $\pi$ of them will land in the circle.
Monte Carlo Algorithm to Calculate $\pi$

Want to calculate the value of $\pi$ using darts and a blindfold

Ratio of the area of the inscribed circle to the area of the square is

$$\frac{\pi r^2}{(2r)^2} = \frac{\pi}{4}$$

for every 4 darts thrown about $\pi$ of them will land in the circle

Choose $n$ points in the square from a uniform random distribution (e.g. throw $n$ darts with a blindfold, etc.)

$n\pi/4$ darts should land in the circle

$$\pi = 4(\text{number of darts in circle})/n$$

- Does this follow the general form?
- How does the accuracy of the approximation change with $n$?
- Make sure you sample from the representative distribution.
Monte Carlo Integration

- Solving complex definite integrals
- Symbolic Integration software
  - How long would that take?
  - Solve all integrals?
- What would be a Monte Carlo solution?
  - How hard to code?
  - How general and how well would it do?
- Multi-variate, high dimensional, non-uniform probability integrals, etc.
Markov Chain Monte Carlo Techniques

- Sampling complex probability distributions?
- MCMC - Markov Chain Monte Carlo algorithms
  - Start from an initial arbitrary state
  - Travel a Markov chain representing the distribution
    - Probability of next state is just a function of the current state
    - Follow this chain of states, sampling as you go, until space is "sufficiently" sampled according to the probability distribution
    - Need a short "burn-in" phase where we discard early samples to avoid dependency on the arbitrary initial state
  - Good approach for solving Bayesian Networks, etc.
- Monte Carlo methods commonly used in computational physics, molecular biology, applied statistics, finance, optimization, machine learning, etc.
Las Vegas Algorithms

- Quicksort sorts in place, using *partitioning*
  - Example: Pivot about a random element (e.g. first element (3))
  - $3 \ 1 \ 4 \ 1 \ 5 \ 9 \ 2 \ 6 \ 5 \ 3 \ 5 \ 8 \ 9$ --- before
  - $2 \ 1 \ 3 \ 1 \ 3 \ 9 \ 5 \ 6 \ 5 \ 4 \ 5 \ 8 \ 9$ --- after

- At most $n$ swaps
  - Pivot element ends up in it’s final position
  - No element left or right of pivot will flip sides again
    - Sort each side independently
    - Recursive Divide and Conquer approach

- Average case complexity is $O(n \log n)$ but empirically better constants than other sorting algorithms

- Speed depends on how well the random pivot splits the data

- Worst case is $O(n^2)$ – Thus a Las Vegas algorithm

- Selection algorithm for median finding is also a Las Vegas algorithm
When to use what Algorithm?

- Not always just one best approach for a task
- Can depend on what style you like
- However, some algorithms will usually fit a task more efficiently
- Have given you an initial toolkit:
  - Divide and Conquer
  - Graph Algorithms
  - Greedy Algorithms
  - Dynamic Programming
  - Linear Programming
  - Intelligent Search
  - Local Search
  - Stochastic Algorithms
- TSP, Sort, Shortest Path, Knapsack, Multiplication, …
- New Problem? can find a similar common problem and its solutions
When to use what Algorithm?

- **Divide and Conquer**
  - Problem has natural hierarchy with independent branches
  - Speed-up happens when we can find short-cuts during partition/merge that can be taken because of the divide and conquer paradigm

- **Graph Algorithms**
  - When finding reachability, paths, and properties of tasks represented as graphs, often fall under other approaches

- **Greedy**
  - Common simple and fast approximation approach, occasionally optimal

- **Dynamic Programming**
  - Overlapping subproblems (given by a recursive definition) that are only slightly (constant factor) smaller than the original problem, solved with the proper ordering

- **Linear Programming**
  - Any optimization with linear objective and constraints

- **Intelligent Search**
  - Effective when we have some heuristic knowledge of the search space to allow pruning

- **Local Search**
  - Simple optimization technique for many complex search spaces – local optima issues

- **Stochastic Algorithms**
  - Sampling problems, amplification of stochastic advantage, takes advantage of fast computers, etc.
Algorithms

- Can often use a combination of different algorithms
  - Divide and Conquer followed by different algorithm on subproblems
  - Stochastic algorithms can often improve many of the base algorithms for certain tasks
  - Greedy components within algorithms
  - etc.

Be Creative!

- Don't get stuck on just one "Hammer"
- The ways in which you apply these algorithmic techniques will not always be initially obvious

You now have a powerful toolbox of algorithmic techniques and philosophies which will prove beneficial as you solve complex tasks in the future