Course Objective

• Introduce a toolkit of modern algorithmic techniques
• Theory of algorithms developed around developing efficient algorithms for a natural class of problems with respect to a standard model of computation (1950s through 1980s)
• Newer work in algorithms focuses on using different models (changing the rules) and bringing in a new set of mathematical tools
My background

• PhD, Stanford (1985)
  • Thesis: *The Complexity of Parallel Algorithms*

• Post Doc (1985-86)
  Mathematical Science Research Institute, Berkeley

• University of Washington (since 1986)
  • Broad range of work: Algorithms, Software Engineering, Educational Technology, Computing for Development

• Sabbatical 1993-1994
  • Indian Institute of Science, Bangalore
  • Parallel Algorithms

• Sabbatical 2001-2002
  • Microsoft Research, Redmond
  • Learning Science and Technology

• Sabbatical 2008-2009
  • PATH, Seattle
  • Digital solutions for global health
Preview of Course Topics

• Distinct from (my) undergraduate courses [see CSE 417/421]
• Work in progress (but drawing on Karlin/Lee CSEP 521)

• Randomized Algorithms and Average Case Analysis
  • Randomization is a powerful technique and worst case analysis sometimes misses the point

• Hashing

• Streaming Algorithms

• High dimensional searching

• Linear Algebra Techniques
PMP Course Philosophy

• Students working full time in industry
• Varying backgrounds and length of time from prerequisite courses
• Interests in both relevance and broadening

• Aim for relatively uniform workload and clear expectations
• Ability to work independently and figure things out and seek out resources and assistance

• We’ve got to make the best of remote teaching
Course mechanics

• Zoom
• Recorded lectures
• Course website: https://courses.cs.washington.edu/courses/csep521/21wi/
• Office hours, zoom links on website
  • RJA: Monday 11am, Friday 2pm
  • Oscar: Wednesday 11 am, Friday 11 am
• Homework Assignments
  • Weekly assignments
  • Electronic turn in on gradescope
  • Mix of written problems and programming experiments
  • Programming assignments
    • Your choice of language (Java, C#, Python) and environment
• No exams, no course project
Making the best of Zoom

• Attendance is class sessions strongly encouraged
• Slides will be available before class
• Feel free to ask questions
• Oscar will monitor chat
  • Useful for clarification questions
• You will need to tolerate some technical glitches
  • “I think the instructor can’t hear us”
• Turning your camera on is optional
• We will use break out rooms for discussion questions
• The course will have office hours, four hours per week
Policies

• Weekly homework assignments, due Tuesdays, 11:59 PM
  • Homework posted on the website (HW1 is available)
  • Late homework accepted with penalty
    • 10% per day
    • Maximum 50% reduction
    • 5 free late days
    • Late day computation will be done at the end of the course
  • Homework received after grading has started may not receive feedback
  • Course grade based on top 9 of 10 assignments

• Collaboration policy
  • Its fine to work together (but not required)
  • Independent write ups
  • Acknowledge collaborators
Models of computation

• Problem: Instance, Solution, Constraints, Value
• Computation Model: Idealized Computer, Unit Cost per Instruction
• Runtime function (for Algorithm A)
  • Runtime on instance I, $T_A(I)$, number of steps to compute solution to I
  • Runtime function, $T_A(n)$, maximum runtime over all instances of length n
Asymptotic Analysis

• Ignore constant factors
  • Constant factors are arbitrary and tedious

• Express run time as $O(f(n))$

• $T(n)$ is $O(f(n))$ \([T : \mathbb{Z}^+ \rightarrow \mathbb{R}^+]\)
  • If $n$ is sufficiently large, $T(n)$ is bounded by a constant multiple of $f(n)$
  • Exist $c$, $n_0$, such that for $n > n_0$, $T(n) < c f(n)$

• Emphasize algorithms with slower growth rates
Randomized Computation

• Assume a source of random bits
• Compute using the random bits
• Multiple types of results are possible
  • Always correct, just the run time varies
  • One sided error
  • Two sided error

• Amplification of probability of correctness
  • Try multiple times
Why randomization

• Actually, lots of reasons
  • Foiling an adversary
  • Random sampling
  • Witnesses
  • Fingerprinting
  • Hashing
  • Re-ordering to avoid bad inputs
  • Load balancing
  • Convergence in Markov chains
  • Symmetry breaking
  • The Probabilistic Technique
Breakout Groups!

• I’m going to try to use breakout groups for discussions / problems, but this first one is just a chance to get to know each other. Introduce yourselves!
Generating a random permutation

• Permutation: Bijection from [1..n] to [1..n]
  • [2, 5, 8, 1, 10, 3, 4, 7, 6, 9]
• n! permutations on [1..n]
• Random permutation – generate permutations with each having probability of exactly 1/n!
Generating a random permutation

```java
public static int[] Permutation(int n, Random rand) {
    int[] arr = IdentityPermutation(n);

    for (int i = 1; i < n; i++) {
        int j = rand.Next(0, i + 1); // Random number in range 0..i
        int temp = arr[i]; // Swap arr[i], arr[j]
        arr[i] = arr[j];
        arr[j] = temp;
    }

    return arr;
}
```
Correctness proof

- **Invariant**: arr[0] \ldots arr[i] is a random permutation of 0 \ldots i at the end of the loop
- **Base case**: i = 0
- **Induction**: Suppose arr[0] \ldots arr[i-1] is a random permutation of 0..i-1 at the start of the loop

Let P be a permutation on i+1 elements, show \( \text{Prob}[P] = 1/(i+1)! \) at end of loop

Suppose \( P[i] = x \) and \( P[j] = i \)

P is created from \( P' \) (on i elements), by moving x from location j to location i

\[
\text{Prob[Swapping i and j]} = 1/(i+1) \\
\text{Prob}[P'] = 1/i! \text{ By the induction hypothesis} \\
\text{Hence } \text{Prob}[P] = 1/(i+1)!
\]
Quicksort

QSort(A)

If |A| <= 1 return A

Choose element x from A
S_1 = \{y \in A \mid y < x\}
S_2 = \{y \in A \mid y > x\}
S_3 = \{y \in A \mid y = x\}
return QSort(S_1) + S_3 + QSort(S_2)
Basic QS facts

• Considered one of best sorts in practice with careful implementation
• Usual runtime $O(n \log n)$, Worst case $O(n^2)$
• Sort \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, \ldots, N\}
• Avoiding the worst case
  • Change pivot selection
  • Randomly permute before sorting
• Compute average case run time

• Mathematically tractable, but not easy
Finding the k-th largest

• Given n numbers and an integer k, find the k-th largest
• If k = n/2 this is computing the median
• Obviously, we can solve this problem by sorting (in time O(n log n)) but can we do better
• Quicksort idea, but only one recursive call
• This will still have the same pathological case
QSelect(A, k)

QSelect(A, k){
    Choose element x from A
    S_1 = {y in A | y < x}
    S_2 = {y in A | y > x}
    S_3 = {y in A | y = x}
    if (|S_2| >= k)
        return QSelect(S_2, k)
    else if (|S_2| + |S_3| >= k)
        return x
    else
        return QSelect(S_1, k - |S_2| - |S_3|)
}
Quick Select

• Same worst case as Quick Sort
• Random pivot will give an $O(n)$ solution (Expected time)
• Deterministic solution (due to BFPRT) not practical

• Exact evaluation of Quick Select recurrence is very difficult
Evaluating Quick Select

• Deterministic recurrence: $T(1) = 1; \ T(N) = T(aN) + bN$ for $a < 1$

• What is the chance that for a random pivot, the subproblem is of size at most $aN$ (for an appropriate $a < 1$)
Analysis

• A random pivot reduces the problem to less then $\frac{3}{4}$ the size with probability at least $\frac{1}{2}$

• Recurrence $T(n) \leq \frac{1}{2} T(\frac{3n}{4}) + \frac{1}{2} T(n) + cn$

• Recurrence for expected number of steps

• This recurrence gives an upper bound on the number of steps for randomized selection
Selection results

• $O(n)$ expected time median algorithm
• Practical algorithm

• How many comparison to find the median?