CSE 373

DECEMBER 4TH – ALGORITHM DESIGN

ASSORTED MINUTIAE

- **P3P3 scripts running right now**
  - Pushing back resubmission to Friday
- **Next Monday office hours**
  - 12:00-2:00 – last minute exam questions
  - Topics list and old practice exams out after class
  - Practice exam (hopefully tomorrow), by Wednesday night

ASSORTED MINUTIAE

- **Course evaluations**
  - Very important to this class and this department
  - Above all, they're very important to me
  - Should only take ~5 minutes, and it's very valuable feedback
  - 17 of you so far… and I'm going to bug you until it's above 75%
  - Save yourself the 15 emails and just fill it out

ALGORITHM DESIGN

- Solving well known problems is great, but how can we use these lessons to approach new problems?

ALGORITHM DESIGN

- **Solving well known problems is great, but how can we use these lessons to approach new problems?**
  - Guess and Check (Brute Force)
  - Linear Solving
  - Divide and Conquer
  - Greedy-first
  - *Randomization and Approximation*
  - *Dynamic Programming*

BRUTE FORCE

- Classic naïve approach to algorithm design

BRUTE FORCE

- **Classic naïve approach to algorithm design**
- **A Brute Force Algorithm revolves primarily around attempting all possible outcomes**
  - Bogo sort
  - Travelling salesman
  - Longest path

BRUTE FORCE

- **If the problem is very difficult, then brute force may not be the worst solution**
  - Cracking RSA
  - Low-reward problems
  - Small, non-time-constrained

LINEAR SOLVING

- **Basic linear approach to problem solving**

LINEAR SOLVING

- Basic linear approach to problem solving
- If the decider creates a set of correct answers, find one at a time

LINEAR SOLVING

- **Basic linear approach to problem solving**
- **If the decider creates a set of correct answers, find one at a time**
  - Selection sort: find the lowest element at each run through
- **Sometimes, the best solution**
  - Find the smallest element of an unsorted array

LINEAR SOLVING

- **Important to understand**
  - What piece of information brings you one step closer to the final answer?

LINEAR SOLVING

- **Important to understand**
  - What piece of information brings you one step closer to the final answer?
  - Exam problem – simple solution
  - Not always bad, O(n) problems lend themselves well to linear solving

DIVIDE AND CONQUER

- **Divide-and-conquer algorithms divide the work and perform work seperately (usually recursively)**
  - Works best for $O(n^k)$ problems
  - Why?

DIVIDE AND CONQUER

- **Divide-and-conquer algorithms divide the work and perform work seperately (usually recursively)**
  - Works best for $O(n^k)$ problems ($k>1$)
  - Why?
  - If an algorithm is n2 work, and we divide into two halves, we've halved the work!

DIVIDE AND CONQUER

- **Divide-and-conquer algorithms divide the work and perform work seperately (usually recursively)**
  - Works best for $O(n^k)$ problems (k>1)
  - Why?
  - If an algorithm is n2 work, and we divide into two halves, we've halved the work!
  - Recurrences are going to play a big role in this

GREEDY-FIRST

- **A Greedy-first algorithm is any algorithm that makes the move that seems best now**
  - These can be divide-and-conquer algorithms or linear algorithms
  - Dijkstra's and Ford-Fulkerson are both Greedy-first algorithms
  - Notice, however, Dijkstra's finds the correct answer easily, and Ford-Fulkerson requires some augmentation to guarantee correctness

ALGORITHM DESIGN

- Which approach should be used comes down to how difficult the problem is

ALGORITHM DESIGN

- **Which approach should be used comes down to how difficult the problem is**
- **How do we describe problem difficulty?**
  - P : Set of problems that can be solved in polynomial time

ALGORITHM DESIGN

- **Which approach should be used comes down to how difficult the problem is**
- **How do we describe problem difficulty?**
  - P : Set of problems that can be solved in polynomial time
  - NP : Set of problems that can be verified in polynomial time

ALGORITHM DESIGN

- **Which approach should be used comes down to how difficult the problem is**
- **How do we describe problem difficulty?**
  - P : Set of problems that can be solved in polynomial time
  - NP : Set of problems that can be verified in polynomial time
  - EXP: Set of problems that can be solved in exponential time

ALGORITHM DESIGN

- **Some problems are provably difficult**

ALGORITHM DESIGN

- **Some problems are provably difficult**
  - Humans haven’t beaten a computer in chess in years, but computers are still far away from “solving” chess

ALGORITHM DESIGN

- **Some problems are provably difficult**
  - Humans haven’t beaten a computer in chess in years, but computers are still far away from “solving” chess
  - At each move, the computer needs to approximate the best move

ALGORITHM DESIGN

- **Some problems are provably difficult**
  - Humans haven't beaten a computer in chess in years, but computers are still far away from "solving" chess
  - At each move, the computer needs to approximate the best move
  - Certainty always comes at a price

APPROXIMATION DESIGN

- What is approximated in the chess game?

APPROXIMATION DESIGN

- **What is approximated in the chess game?**
  - Board quality – If you could easily rank which board layout in order of quality, chess is simply choosing the best board

APPROXIMATION DESIGN

- **What is approximated in the chess game?**
  - Board quality – If you could easily rank which board layout in order of quality, chess is simply choosing the best board
  - It is very difficult, branching factor for chess is ~35

APPROXIMATION DESIGN

- **What is approximated in the chess game?**
  - Board quality – If you could easily rank which board layout in order of quality, chess is simply choosing the best board
  - It is very difficult, branching factor for chess is ~35
  - Look as many moves into the future as time allows to see which move yields the best outcome

APPROXIMATION DESIGN

- **Recognize what piece of information is costly and useful for your algorithm**

APPROXIMATION DESIGN

- **Recognize what piece of information is costly and useful for your algorithm**
  - Consider if there is a cheap way to estimate that information

APPROXIMATION DESIGN

- **Recognize what piece of information is costly and useful for your algorithm**
  - Consider if there is a cheap way to estimate that information
  - Does your client have a tolerance for error?
  - Can you map this problem to a similar problem?
  - "Greedy" algorithms are often approximators

RANDOMIZATION DESIGN

- **Randomization is also another approach**

RANDOMIZATION DESIGN

- **Randomization is also another approach**
  - Selecting a random pivot in quicksort gives us more certainty in the runtime

RANDOMIZATION DESIGN

- **Randomization is also another approach**
  - Selecting a random pivot in quicksort gives us more certainty in the runtime
  - This doesn't impact correctness, a randomized quicksort still returns a sorted list

RANDOMIZATION DESIGN

- **Randomization is also another approach**
  - Selecting a random pivot in quicksort gives us more certainty in the runtime
  - This doesn't impact correctness, a randomized quicksort still returns a sorted list
- **Two types of randomized algorithms**
  - Las Vegas – correct result in random time

RANDOMIZATION DESIGN

- **Randomization is also another approach**
  - Selecting a random pivot in quicksort gives us more certainty in the runtime
  - This doesn’t impact correctness, a randomized quicksort still returns a sorted list
- **Two types of randomized algorithms**
  - Las Vegas – correct result in random time
  - Montecarlo – estimated result in deterministic time

RANDOMIZATION DESIGN

- **Can we make a Montecarlo quicksort?**

RANDOMIZATION DESIGN

- **Can we make a Montecarlo quicksort?**
  - Runs O(n log n) time, but not guaranteed to be correct

RANDOMIZATION DESIGN

- **Can we make a Montecarlo quicksort?**
  - Runs O(n log n) time, but not guaranteed to be correct
  - Terminate a random quicksort early!

RANDOMIZATION DESIGN

- **Can we make a Montecarlo quicksort?**
  - Runs O(n log n) time, but not guaranteed to be correct
  - Terminate a random quicksort early!
  - If you haven’t gotten the problem in some constrained time, just return what you have.

RANDOMIZATION DESIGN

- **How *close* is a sort?**
- **If we say a list is 90% sorted, what do we mean?**

RANDOMIZATION DESIGN

- **How *close* is a sort?**
- **If we say a list is 90% sorted, what do we mean?**
  - 90% of elements are smaller than the object to the right of it?

RANDOMIZATION DESIGN

- **How *close* is a sort?**
- **If we say a list is 90% sorted, what do we mean?**
  - 90% of elements are smaller than the object to the right of it?
  - The longest sorted subsequence is 90% of the length?

RANDOMIZATION DESIGN

- **How *close* is a sort?**
- **If we say a list is 90% sorted, what do we mean?**
  - 90% of elements are smaller than the object to the right of it?
  - The longest sorted subsequence is 90% of the length?
- **Analysis for these problems can be very tricky, but it's an important approach**

RANDOMIZATION

- Guess and check

RANDOMIZATION

- **Guess and check**
  - How bad is it?

RANDOMIZATION

- **Guess and check**
  - How bad is it?
    - Necessary for some hard problems

RANDOMIZATION

- **Guess and check**
  - How bad is it?
    - Necessary for some hard problems
    - Still can be useful for some easier problems

RANDOMIZATION

- **Guess and check**
  - How bad is it?
    - Necessary for some hard problems
    - Still can be useful for some easier problems
    - Hugely dependent on how “good” the checker is

RANDOMIZATION

- If an algorithm has a chance P of returning the correct answer to an NP-complete problem in $O(n^k)$ time

RANDOMIZATION

- **If an algorithm has a chance P of returning the correct answer to an NP-complete problem in $O(n^k)$ time**
  - P is our success probability

RANDOMIZATION

- **If an algorithm has a chance P of returning the correct answer to an NP-complete problem in $O(n^k)$ time**
  - P is our success probability
  - NP-complete means we can check a solution in $O(n^k)$ time, but we can find the exact solution in $O(k^n)$ time – very bad
  - Suppose we want to have a confidence equal to α, how do we get this?

RANDOMIZATION

- Even if P is low, we can increase our chance of finding the correct solution by running our randomized estimator multiple times

RANDOMIZATION

- **Even if P is low, we can increase our chance of finding the correct solution by running our randomized estimator multiple times**
  - We can verify solutions in polynomial time, so we can just guess-and-check.

RANDOMIZATION

- **Even if P is low, we can increase our chance of finding the correct solution by running our randomized estimator multiple times**
  - We can verify solutions in polynomial time, so we can just guess-and-check.
  - How many times do we need to run our algorithm to be sure our chance of error is less than α?

RANDOMIZATION

- **Even if P is low, we can increase our chance of finding the correct solution by running our randomized estimator multiple times**
  - We can verify solutions in polynomial time, so we can just guess-and-check.
  - How many times do we need to run our algorithm to be sure our chance of error is less than α?

RANDOMIZATION

$(1-p)^k = \alpha$

RANDOMIZATION

$(1-p)^k = \alpha$

$k*\ln(1-p) = \ln \alpha$

$$k = \frac{(\ln \alpha)}{(\ln(1-p)}$$

$k = \log_{(1-p)} \alpha$

RANDOMIZATION

- Cool, I guess… but what does this mean?

RANDOMIZATION

- Cool, I guess… but what does this mean?
- Suppose P = 0.5 (we only have a 50% chance of success on any given run) and $\alpha = 0.001$, we only tolerate a 0.1% error

RANDOMIZATION

- Cool, I guess… but what does this mean?
- Suppose P = 0.5 (we only have a 50% chance of success on any given run) and $\alpha = 0.001$, we only tolerate a 0.1% error
- How many runs do we need to get this level of confidence?

RANDOMIZATION

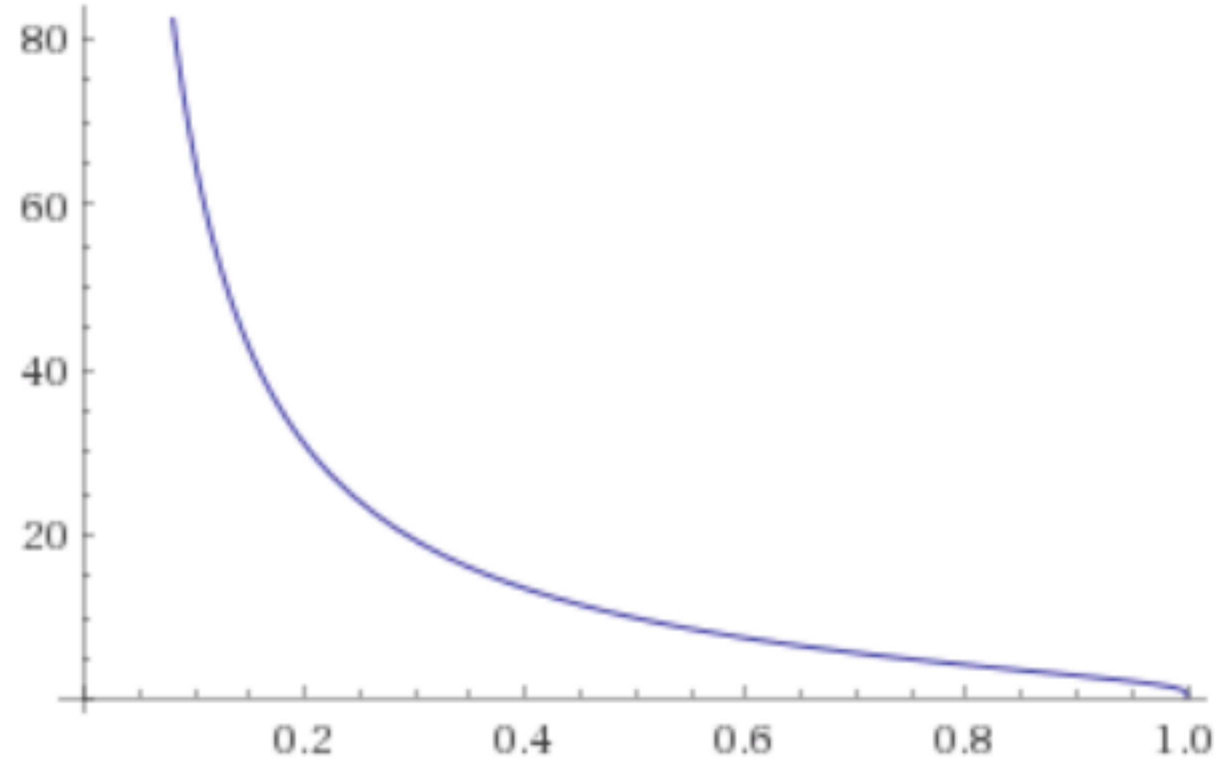
- **Cool, I guess… but what does this mean?**
- **Suppose P = 0.5 (we only have a 50% chance of success on any given run) and α = 0.001, we only tolerate a 0.1% error**
- **How many runs do we need to get this level of confidence?**
  - Only 10! This is a constant multiple

RANDOMIZATION

- In fact, suppose we always want our error to be 0.1%, how does this change with p?

RANDOMIZATION

- **In fact, suppose we always want our error to be 0.1%, how does this change with p?**



RANDOMIZATION

- Even if p is 0.1, only a 10% chance of success, we only need to run the algorithm 80 times to get a 0.001 confidence level

RANDOMIZATION

- Even if p is 0.1, only a 10% chance of success, we only need to run the algorithm 80 times to get a 0.001 confidence level
- What does this mean?

RANDOMIZATION

- **Even if p is 0.1, only a 10% chance of success, we only need to run the algorithm 80 times to get a 0.001 confidence level**
- **What does this mean?**
  - Randomized algorithms don't have to be complicated, if you can create a *reasonable* guess and can verify it in a short amount of time, then you can get good performance just from running repeatedly.

RANDOMIZATION CONCLUSION

- **Good for estimating difficult problems in constrained time**

RANDOMIZATION CONCLUSION

- Good for estimating difficult problems in constrained time
- Relies on the quality of the guess

RANDOMIZATION CONCLUSION

- Good for estimating difficult problems in constrained time
- Relies on the quality of the guess
- Important approach to consider in modern computing

CONCLUSION

- **Be prepared for the algorithm design question on the final**
  - Understand how to go about getting the solution
  - Rigorous analysis, of both runtime and memory
  - Defend all design decisions
  - More points for explanation than for cleverness

CONCLUSION

- **Course evaluations**
  - https://uw.iasystem.org/survey/183488