Algorithm Design and Analysis

chadd@pacificu.edu

Office Hours:
MW 11am-noon
Friday 1-3pm

Syllabus

- Book
- Schedule
- Grading Assignments/Exams/Quizzes
- Policies
    - Late Policy
    - Grade Complaints
- Moodle

Overview

- Topics
- Goals
- Where does this class fit?
  - Data Structures → Algo ← Discrete Math
    - Big Oh, Data Structs Graphs, Proofs, Logic
  - Theory ↔ Algo

What is an Algorithm? chapter 1

-

How do we discuss algorithms?

How to we evaluate algorithms?

- Performance
- and ...

Why Study Algorithms?

Correctness

Demonstrating Incorrectness

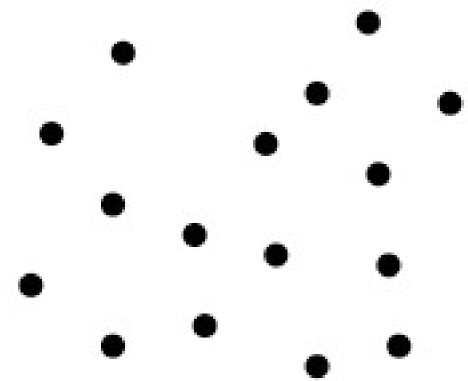
- Searching for counterexamples is the best way to disprove the correctness of a heuristic.
- Think about all small examples.
- Think about examples with ties on your decision criteria (e.g. pick the nearest point).
- Think about examples with extremes of big and small.

Induction and Recursion

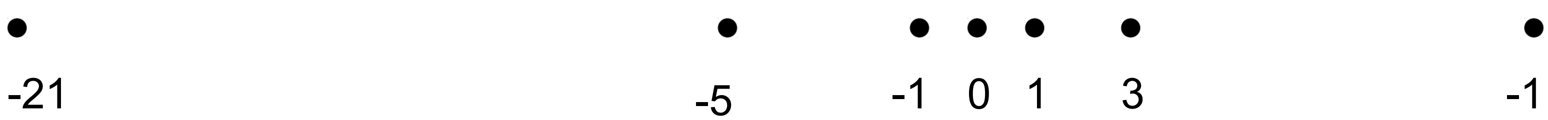
- Failure to find a counterexample to a given algorithm does not mean “it is obvious” that the algorithm is correct.
- Mathematical induction is a very useful method for proving the correctness of recursive algorithms.
- Recursion and induction are the same basic idea: (1) basis case, (2) general assumption, (3) general case.

Correctness is Not Obvious!

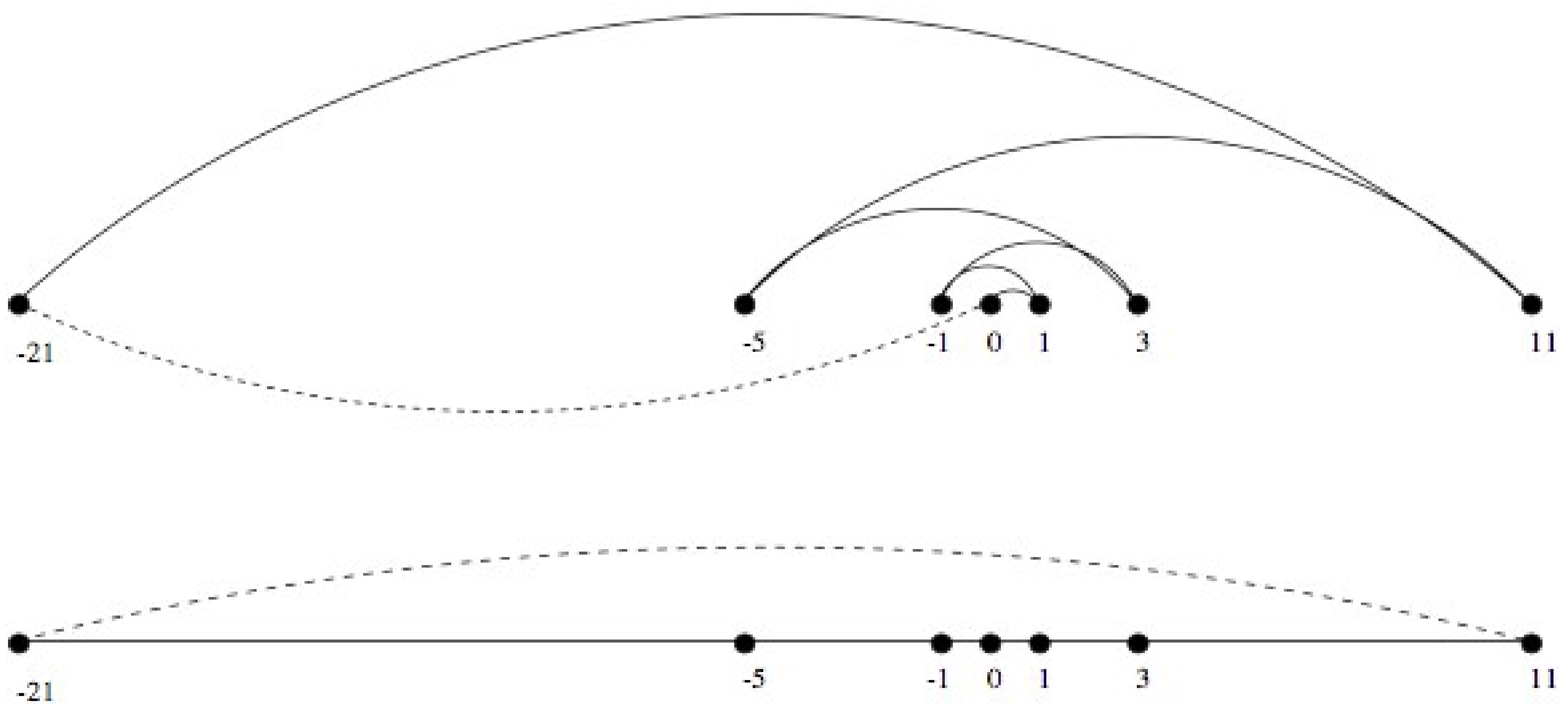
- Visit each point once, minimizing the distance moved



Nearest Neighbor Tour



Nearest Neighbor Tour



A Correct Algorithm

Why Not Use a Supercomputer

- A faster algorithm running on a slower computer will always win for sufficiently large instances
- Usually, problems don’t have to get that large before the faster algorithm wins

Expressing Algorithms

- What are the possible ways to express an algorithm?
  - English
  - Pseudocode
  - Programming Language

The RAM Model, section 2.2

- Algorithms can be studied in a machine and language independent way.
- Each “simple” operation (+, -, =, if, call) takes exactly one step.
- Loops and subroutines are not simple operations.
- Each memory access takes one step.

Tuning

- But what about registers, cache, RAM, Virtual Memory, pipelining, branch prediction, etc.

Best, Worst, and Average-Case

- Worst case:
- Best case:
- Average case *:

rate of growth   OR order of growth

Example: Sorting

- **Input:** A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$
- **Output:** A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$
- We seek algorithms that are *correct* and *efficient*

Insertion Sort, p 26

- INSERTION-SORT(A) // A is an array

```
for j = 2 to A.length
  key = A[j]
  // Insert A[j] in to the correct location
   i = j - 1
   while i > 0 and A[i] > key
      A[i+1] = A[i]
      i = i - 1
   A[i+1] = key
```

Example

- How would insertion sort work on the following numbers?
  - 3 1 7 4 8 2 6

Insertion Sort

- Is the algorithm correct?
- How efficient is the algorithm?
- How does insertion sort do on sorted permutations?
- How about unsorted permutations?

Analysis of Insertion Sort

- Best Case

Analysis of Insertion Sort

- Worst Case

- We generally care about this analysis.
  - gives the upper bound time
  - average case is often closer to worst than best
    - and average case is often very hard to compute
  - unless we know we are in a best case

Your Turn

- Problem: How would insertion sort work on the following characters to sort them alphabetically (from A -> Z)? Show each step.

  S O R T E D

  this is a good exam question.

For Next Time

- Read Chapters 1 and 2 from the book.