# Algorithm Design and Analysis

shereen@pacificu.edu

# Overview

- Where does this class fit?
- Topics
  - Data structures
  - Big Oh
  - Searching
  - Sorting
  - Graphs
  - Proofs

# What is an Algorithm?

- A sequence of computational steps that transforms the ***input*** into the desired ***output***

# Do Algorithms Matter?

A Story

- Once upon a time …

From Algorithms in a Nutshell. O'Reilly

# Do Algorithms Matter?

A Story

- Danny's Idea: write a program to find memory leaks
- Built a small library that wrapped the OS's memory allocation and deallocation routines with new functions
- These functions recorded each allocation and deallocation in a data structure that would be queried at the end of the program

# Do Algorithms Matter?

A Story

- Problem: Program ran really slowly
- Heidi to the rescue!

# Do Algorithms Matter?

A Story

- Heidi: Describe the problem and solution
- Danny: ….
- Heidi : Is there a difference in the performance of the programs?
- Danny: Small programs run in acceptable time, regardless if they had memory leaks. Programs that did a lot of processing and had memory leaks ran disproportionately slow

# Experiments: Program A

A Story

```
int main(int argc, char **argv)
{
  for(int i = 0; i < 1000000; i++)
  {
    malloc(32);
  }
  exit(0);
}
```

# Experiments: Program B

A Story

```
int main(int argc, char **argv)
{
  for(int i = 0; i < 1000000; i++)
  {
    void *x = malloc(32);
    free(x);
  }
  exit(0);
}
```

# Experiments: Program C

A Story

```
int main(int argc, char **argv)
{
  void *addrs[1000000];
  for(int i = 0; i < 1000000; i++)
  {
    addrs[i] = malloc(32);
  }
  for(int i = 0; i < 1000000; i++)
  {
    free(addrs[i]);
  }
  exit(0);
}
```

# New Insight



- It’s not the number of memory allocations open at the end of the program that affected performance.
- Instead, it’s …

# Algorithms Matter!

A Story

- Heidi: How do you track allocated memory?
- Danny: A binary search tree. Each node is a struct containing:
  - Pointers to children
  - Address of allocated memory
  - Size allocated
  - Place in program where allocation was made
- Memory address is the key for the nodes

A Story

- Binary Search Tree is a good choice
- Key is memory address
  - malloc allocates memory from the heap in order of increasing memory address
  - What happens if addresses are 1-15 (for the sake of argument)?
- What is the problem with Danny's code?

# How do we evaluate algorithms?

- Performance
- And …
  - o
  - o
  - o
  - o
  - o

# Why Study Algorithms?

- 
- 
- 
- 
- 
- 
- 

# Correctness

- For any algorithm, we must prove that it *always* returns the desired output for all legal *instances* of the problem.
- What does this mean for a sorting algorithm?

# 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.

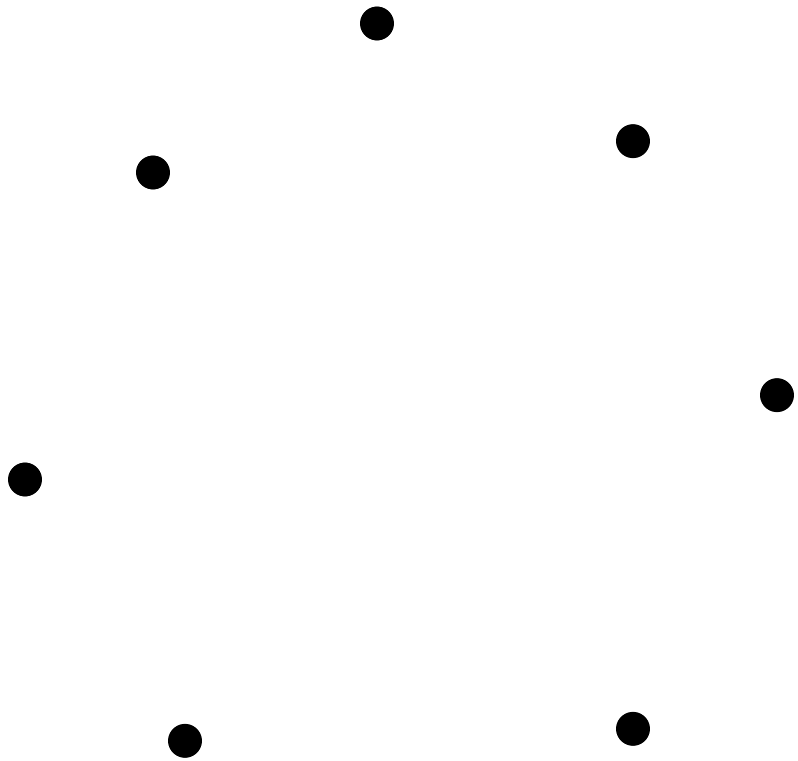
# Correctness is Not Obvious!

- Suppose you have a robot arm equipped with a tool, say a soldering iron. To enable the robot arm to do a soldering job we must construct an ordering of the contact points so the robot visits (and solders) the first contact point, then visits the second point, third, and so forth until the job is done.
- Since robots are expensive, we need to find the order which minimizes the time (ie. travel distance) it takes to assemble the circuit board.

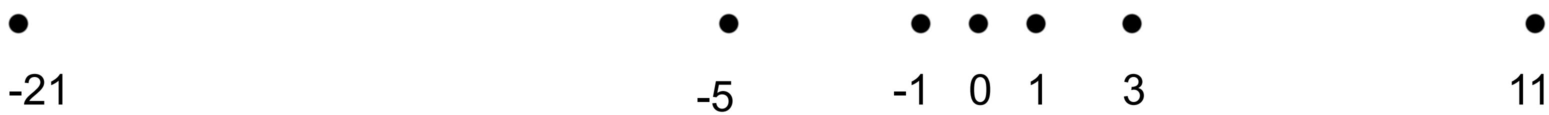
# Correctness is Not Obvious!

- 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

# Best, Worst, and Average-Case

- Worst case:

- Best case:

- Average case:

# 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. 18)

```
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

# 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

# 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

# For Next Time

- Read Chapters 1 and 2 from the book.