Algorithm
Bourbour

Basic Info

- Instructor: Sara Bourbour (sarabourbour@aut.ac.ir)

- Course stuff (syllabus, lecture notes, assignments, grades) will
- be posted on Model

- More about the course and the syllabus later ...

What Are Algorithms?

- Algorithms are the essential ideas behind computer programs.
- An algorithm is what stays the same whether the program is in Pascal running on a Cray in New York or is in BASIC running on a Mac in Kathmandu!
- If hardware is the brawns of computing, algorithms are the brains.

Algorithms are a type of technology

- Algorithms (like software) aren’t “tangible”, so can be hard to convince non-CS people.
- E.g., typical person feels less guilty pirating SW (bits) than stealing a TV set (atoms).
- Nevertheless, algorithms ARE a form of technology.
- One company’s competitive advantage over another is often that it uses better algorithms.

Why you should take this course seriously

- Builds a library of (problem, solution) pairs in your brain.
- Helps you to reason about correctness & efficiency of programs.
- “Makes Computer Scientists out of Programmers”
- Helps you answer interview questions! Seriously!

https://www.youtube.com/watch?v=k4RRi_ntQc8

Syllabus Review

- Grading rubric
- Academic integrity
- Participation
- Final Grades
- Text book

Algorithms & Problems

- To be interesting, an algorithm has to solve a general, specified problem.
- An algorithmic problem is specified by describing the set of instances (inputs) it must work on and what desired properties the output must have

Don’t confuse problems and algorithms

- An algorithm is the solution to an algorithmic problem.
- Many algorithms possible for a given problem
- E.g., the sorting problem can be solved using bubblesort, insertion sort, selection sort, quicksort, mergesort, heap sort, radix sort, etc, algorithms.

Example: The Sorting Problem

- Input: A sequence of N numbers $a_1 \ldots a_N$
  Output: the permutation (reordering) of the input sequence such that $a_1' \leq a_2' \ldots \leq a_N'$.
  A Legal Instance: [7, 6, 11, − 3, 56]
  How about: [bill, mike, bob]?
  Generalize the problem statement:
  - "number" ⇒ "key"
  - Should be able to define a suitable comparison operator (e.g., need transitivity).

  Want algorithms that are correct and efficient.

Correctness

For any algorithm, we must prove that it always returns the desired output for all legal instances of the problem.
For sorting, this means even if

1. the input is already sorted, or
2. it contains repeated elements

Other Properties of Good Algorithms?

- Correct

- Efficient (we should be more specific about what this means).
- Can you think of others?

- How about Easy to Understand & Implement?

Expressing Algorithms

We need some way to express the sequence of steps comprising an algorithm.
Our options, in increasing order of precision

1. English
2. Pseudocode ("a programming lang that doesn't complain about syntax errors")
3. Real programming languages.

Unfortunately, ease of expression moves in the reverse order. We'll describe the ideas of an algorithm in English.
Use pseudocode (or real code) to clarify details.

Insertion Sort: a high-level view

- Start with an empty list, then successively insert new elements in the proper position:
- $a_1 \leq a_2 \leq \ldots \leq a_k \mid a_{k+1} \ldots a_n$
- At each stage, you (i) start with a sorted list (ii) insert an element and get a bigger sorted list.
  After n insertions, you have sorted the input.
  Above is an example of inductive or recursive or incremental reasoning.

Thought for the Day

“A computer scientist is a mathematician who only knows how to prove things by induction!”

Induction and Recursion

- 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.

Quick Review of Induction

- Classical induction problem: prove that

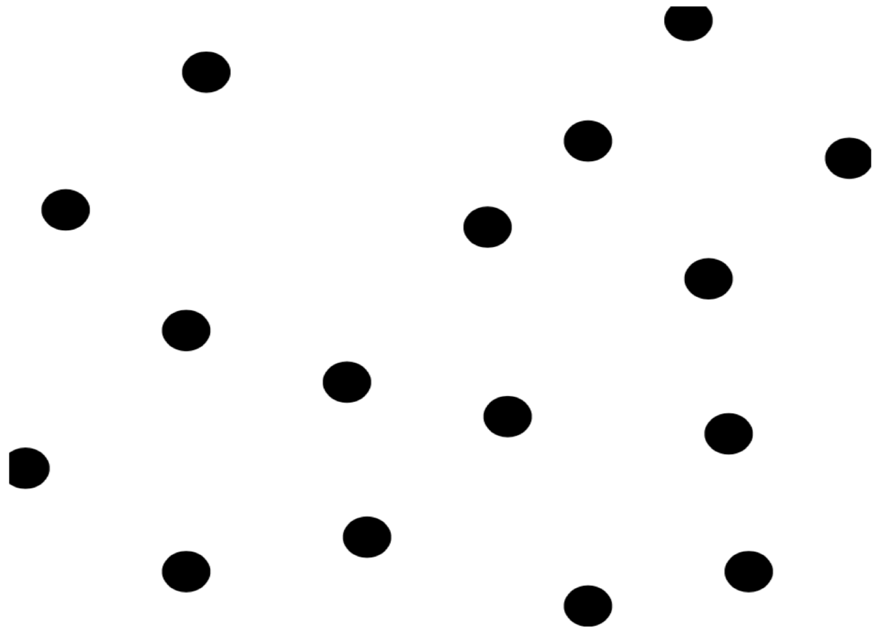
$$\text{P(n)}: \sum_{i=1}^{n} = n(n+1)/2$$

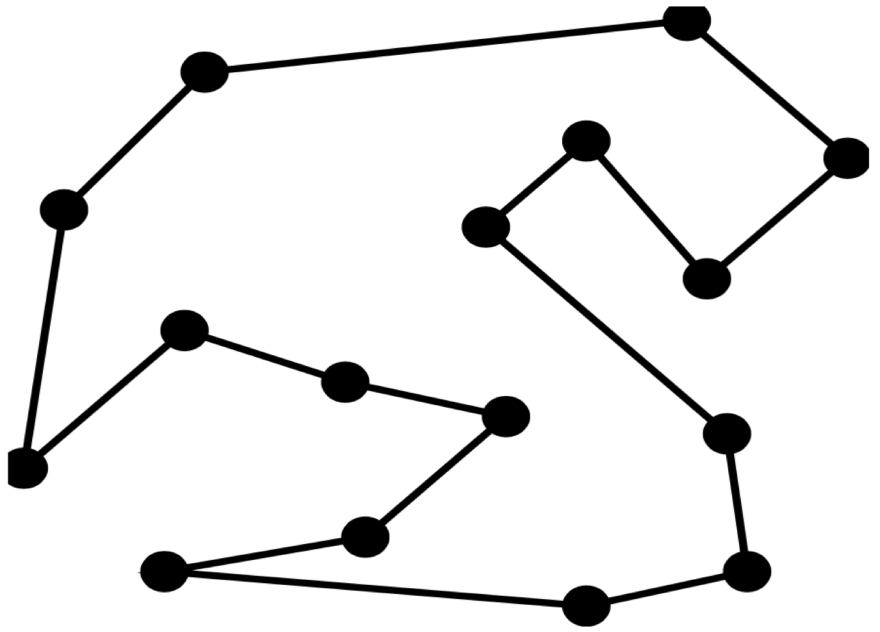
- Proving ∞ assertions!
- Review mechanical aspects of proof.
- Review philosophy: why it works for any positive integer

An Optimization problem

Input: a set of points
Output: a tour that starts at any point, visits all points, returns to starting point. Furthermore, want tour of least distance.

Find the Shortest Tour





Give me an algorithm to find the best tour!

Nearest Neighbor Tour

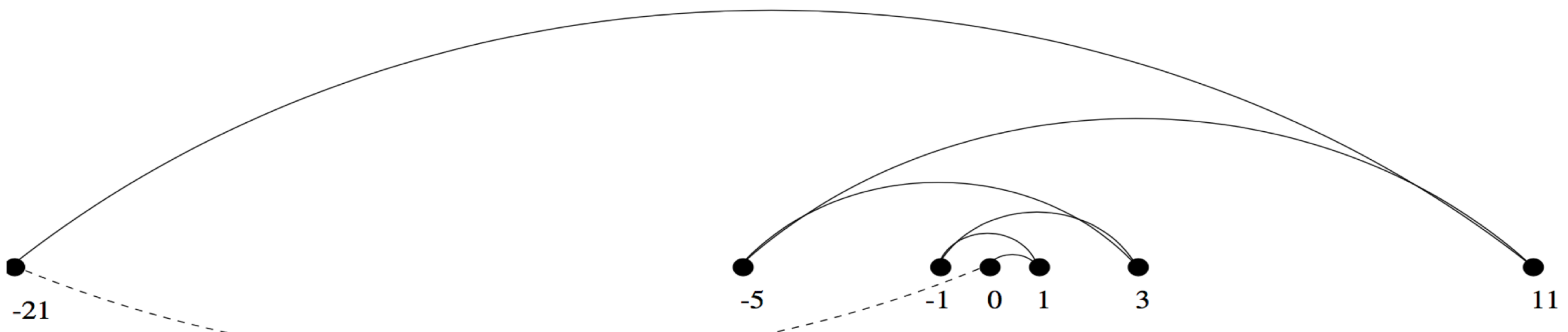
A popular solution starts at some point p_0 and then walks to its nearest neighbor p_1 first, then repeats from p_1, etc. until done.

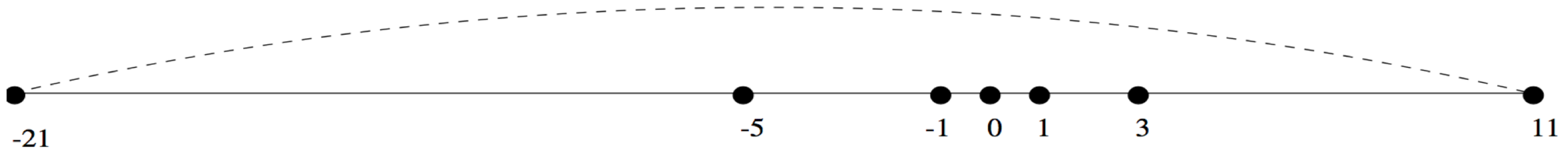
- Pick and visit an initial point p_0

```
p = $p_0$
i = 0
While there are still unvisited points
        i = i+1
        Let $p_i$ be the closest unvisited point to pi−1 Visit pi
```

- Return to p_0 from p_i

Nearest Neighbor Tour is Wrong!





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)
  Thinkaboutexampleswithextremesofbigandsmall...

Note: Failure to find a counterexample to a given algorithm does not mean "it is obvious" that the algorithm is correct.

A Correct Algorithm for TSP: Exhaustive Search

Try all possible orderings of the points, then select the one which minimizes the total length:

```
• d=∞
  For each of the n! permutations $\Pi_i$ of the n $\text{Type equation here.}$points
          If (cost($\Pi_i$) ≤ d) then
                  d = cost(Πi) and Pmin = Πi
Return
```

$$P_{min}$$

All orderings considered, guaranteed optimal tour.

Exhaustive Search is Very Very Slow!

- Because it tries all n! permutations, it is much too slow to use when there are more than 10-20 points.
  How long would this take on a 1 THz processor if it takes a clock cycle to evaluate a permutation?
- BTW, are there really n! distinct tours?
  No efficient, correct algorithm has been discovered for the traveling salesperson problem!