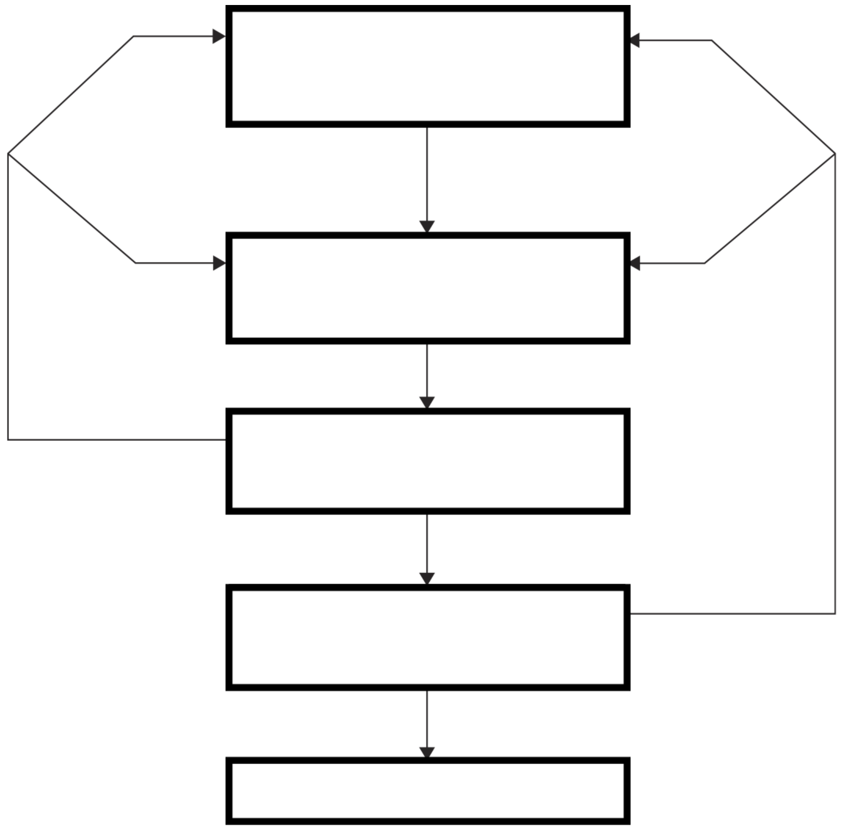
1 Algorithm Design & Analysis Process

Fill in the flowchart below with the appropriate process in each box.

Design an algorithm.
Analyze the running time.
Prove correctness.
Understand and clearly specify the problem.
Implement (code) the algorithm.



2 Writing Pseudocode

Consider the following cumulative variation of a popular children's song:

Old MacDonald had a farm. E-I-E-I-O.
And on that farm he had a **pig**.
With an **oink oink** here. And an **oink oink** there.
Old MacDonald had a farm. E-I-E-I-O.

Old MacDonald had a farm. E-I-E-I-O.
And on that farm he had a **duck**.
With an **quack quack** here. And an **quack quack** there.
With an oink oink here. And an oink oink there.
Old MacDonald had a farm. E-I-E-I-O.

Old MacDonald had a farm. E-I-E-I-O.
And on that farm he had a **horse**.
With an **neigh neigh** here. And an **neigh neigh** there.
With an quack quack here. And an quack quack there.
With an oink oink here. And an oink oink there.
Old MacDonald had a farm. E-I-E-I-O.

Old MacDonald had a farm. E-I-E-I-O.
And on that farm he had a **sheep**.
With an **baaa baaa** here. And an **baaa baaa** there.
With an neigh neigh here. And an neigh neigh there.
With an quack quack here. And an quack quack there.
With an oink oink here. And an oink oink there.
Old MacDonald had a farm. E-I-E-I-O.

⋮

Write pseudocode for an algorithm to sing this song, given lists of animals and sounds:

Algorithm 1 Farm Song Algorithm

```
procedure FarmSong($animals[1..n]$, $sounds[1..n]$)
    ▷ Fill this in
                                                        ◁
```

3 Understanding Algorithms

Let's figure out what the following algorithm does:

Algorithm 2 Unknown Algorithm

```
procedure SOMETHING($A[1..n]$)
    ▷ Input: Array $A[1..n]$ of numbers
      Output: ??? ◁
    $d \leftarrow \infty$
    for $i = 1, \ldots, n$ do
        for $j = 1, \ldots, n$ do
            if $i \neq j$ and $|A[i] - A[j]| < d$ then
                $d \leftarrow |A[i] - A[j]|$
    return $d$
```

1. Write down a small sample array, A.
2. Trace through the execution of the nested loops.
3. What is the **if** statement checking?
4. How many times is d updated for your array?
5. What does the algorithm produce as the overall result?

4 Reformulating Algorithms

Consider the following iterative[1] algorithm from the zeroth chapter of our textbook:

Algorithm 3 Peasant multiplication

```
procedure PeasantMult($x, y$)
    $prod \leftarrow 0$
    while $x > 0$ do
        if $x$ is odd then
            $prod \leftarrow prod + y$
        $x \leftarrow \lfloor x/2 \rfloor$
        $y \leftarrow y + y$
    return $prod$
```

1. Pick two 3-digit numbers and trace the execution of the algorithm on them.
2. Why does the algorithm work?
3. Rewrite the algorithm so that it is *recursive* instead of iterative.
4. How would you go about analyzing the running time of the algorithm?

[1] "Iterative" means it is described using a *loop* construct (`while` or `for`).

5 Problem Types

Let's talk about the most important/common types of problems encountered in CS.

Table 1: 7 types of problems

Geometric problems	String processing	Sorting	Numerical problems
Searching	Combinatorial problems	Graph problems	

Table 2: Examples of applications

Find the **best route** between two cities.
Find the **closest pair** among n points in the plane.
Search an entire gene sequence for a particular **subsequence of characters**.
Rank search results on the Internet.
Compute the **roots of a polynomial** function.
Quickly **retrieve information** from a large database.
Find a subset of numbers in a list that sum to zero.

Match up the seven problem types above, and the applications, with the definitions below:

Type	Description	Example
	Rearrange the items of a list in some order.	
	Finding a key in a set of data.	
	Handling a sequence of characters from an alphabet.	
	Modeling and analyzing a collection of vertices connected by edges.	
	Finding an object of a finite collection that satisfies certain constraints	
	Deal with objects such as points, lines, and polygons.	
	Problems that involve mathematical objects of continuous nature.	