CSE 373: Introduction

Chapter 1

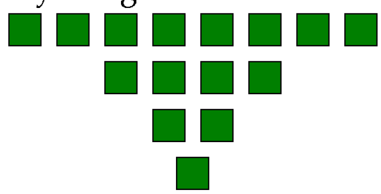
Basic Math

- Things to review on your own (§1.2.1–1.2.5)
  - exponents
  - logarithms
  - series
  - modular arithmetic
  - proof techniques (except inductive)

Notes on Logarithms

- Understanding $\log_b x$
  - usually defined: $\log_b x = y \Rightarrow b^y = x$

    ($\log_b x$ is the power to which b must be taken to get x)
  - more useful: $\log_b x$ is the number of times you must divide x by b to get 1



2^3: $\log_2 8 = 3$

2^2: $\log_2 4 = 2$

2^1: $\log_2 2 = 1$

2^0: $\log_2 1 = 0$

- b is almost always 2 and omitted by default

Recursion

Recursive function: A function that calls itself

- Analogous to recurrence relations in math:

  0! = 1 fact(0) = 1

  x! = x · (x-1)! fact(x) = x · fact(x-1)
- Recursively in C:

```
int fact(int x) {
  if (x == 0) {
    return 1;
  } else {
    return x * fact(x-1);
  }
}
```

Disadvantages of Recursion

- Function calls are expensive
  - take more time than standard operations
  - require memory proportional to the call depth
- Simple cases can be rewritten with loops:

```
int fact(int x) {
  if (x == 0) {
    return 1;
  } else {
    return x * fact(x-1);
  }
}
```

```
int fact(int x) {
  int product;
  product = 1;
  while (x > 0) {
    product *= x;
    x--;
  }
  return product;
}
```


Recursion II

Fibonnaci Numbers:

$fib_0 = 1$

$fib_1 = 1$

$fib_x = fib_{x-1} + fib_{x-2}$

- Recursively in C:

```
int fib(int x) {



}
```

Disadvantages of Recursion II

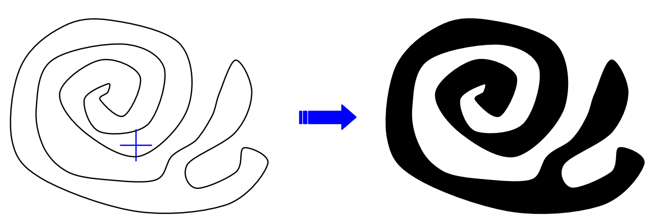
- Elegance disguises redundant computation
- What is the call chain like for `fib(5)`? `fib(10)`?

- Does `fib()` have a simple iterative rewrite?

Recursion III

```
void PaintFill(int pixel[][],int x,int y);
```

- pixels are either black (1) or white (0)
- starting at pixel (**x**,**y**) change white pixels to black, stopping at boundaries



Recursion III (continued)

```
void PaintFill(int pixel[][],int x,int y){




}
```

Does `PaintFill()` have an iterative rewrite?

Recursion Summary

- Recursive routines must:
  - have a base case
  - always make progress towards the base case
- Be sure to keep an eye out for:
  - recursive calls that have simple iterative rewrites
  - redundant computation

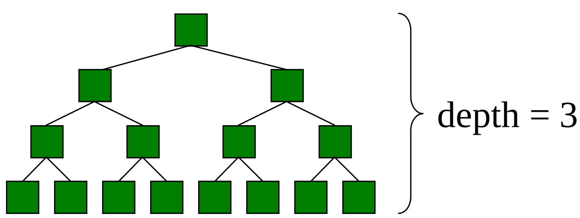
Inductive Proofs

Inductive proof – A way to prove a property true for an infinite number of (enumerable) cases

- prove property true for base case(s)
- assume it's true for the first k-1 instances, and use them to prove it's true for the k^{th} instance

Simple Inductive Proof

Prove: Every complete binary tree of depth d contains 2^{d+1} - 1 nodes



Simple Inductive Proof (cont'd)

Proof (by induction):

- Let P(i) = "A complete binary tree of depth i contains 2^{i+1} - 1 nodes"
- We must prove P(i) true for all $i \geq 0$
- *base case:* Prove P(0) is true

Simple Inductive Proof (cont'd)

Proof (continued):

- *inductive step:* Assuming P(0), P(1), …, P(k-1) are true, prove P(k) is true

- Therefore, for all $i \geq 0$, P(i) is true

Induction and Recursion

Induction and Recursion are analogous concepts

- both use base cases
- both solve "big" problems based on the assumption that "smaller" problems are solved in a similar way
- both require that you assume the recursive/ inductive step works without checking every case
- both have similar pitfalls
  - determining the number of base cases
  - handling the base case(s) correctly
  - getting the inductive step to work for all non-base cases

An Incorrect Inductive Proof

Prove: When h horses are within a fenced area, they are all the same color

Proof (by induction):

- Let P(i) = "when i horses are within a fenced area, they are all the same color"
- *base case:* when 1 horse is in a fenced in area, it is the same color as itself. Therefore, P(1) is true.

An Incorrect Inductive Proof (cont'd)

– *inductive step:* Assume P(1), P(2), …, P(k-1) are true.

- Consider k horses in a fenced-in area.
- Lead one of the horses, a, out of the area such that k-1 horses remain. Since P(k-1) is true, the remaining horses must all be the same color.
- Now lead a back in and lead a different horse, b, out, once again leaving k-1 horses within the fence. Since P(k-1) is true, these horses must also all be the same color.
- Since both subsets of k-1 horses were the same color, a and b must be the same color, and therefore all k horses must be the same color
- Therefore P(1), …, P(k-1) $\Rightarrow$ P(k) is true