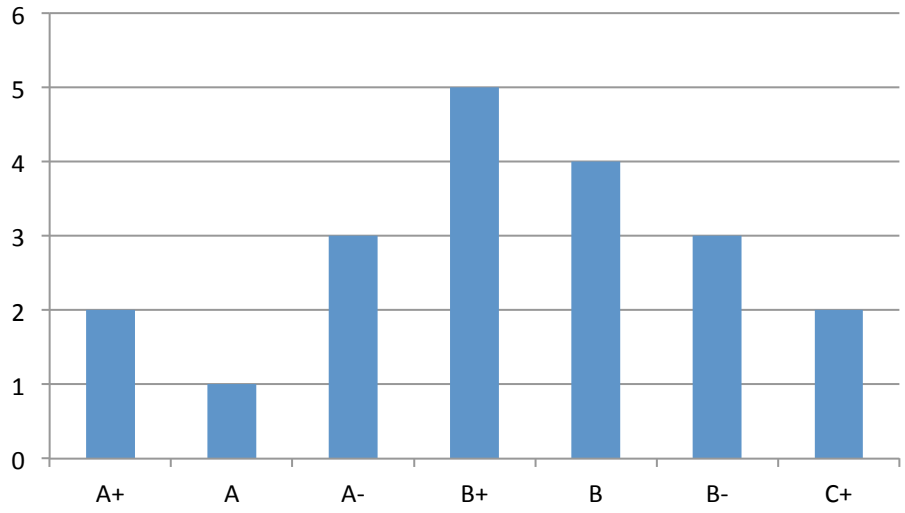
Chapter 12

Recursion

- Mean: 41
- Median: 37, B+



- Camera example

Recursive Methods

Complexity (often) controlled by an int parameter

- Recursive invocations simpler
- Simpler= smaller, non-negative parameter

Recursive methods

- Typically contain a conditional statement
- At least one case is a recursive invocation
- At least one case is a base case --i.e., no recursive invocation of the method

Writing recursive methods

- Write the base case
  - No recursive call
- Write the recursive case
  - All recursive calls should go to simpler cases
  - Simpler cases must eventually reach base case

numexponent= ?

```
public double simplePower(double num, int exponent)
{
}
```

Applying rules to simplePower

- Base case: exponent == 0
  - Returns 1
  - Correct answer for raising base to the 0th power
  - No recursive invocation
- Recursive case: uses else clause
  - Recursive call involves smaller value for exponent
  - Recursive calls eventually reach base case of 0: exponent greater than 0 to start and always goes down by 1

An example: Exponentiation

- Inspiration: Fast algorithms for exponentiation important to RSA algorithm for public key cryptography
- A simple (not fast!) recursive method:

```
// returns base raised to exponent as long as exponent >=0
public double simplePower(double base, intexponent) {
if (exponent == 0)
  {
    return 1;
  } else
  {
    return base * simplePower(base, exponent-1);
  }
}
```

Rules of Exponents

- Simple algorithm took advantage of these rules:
  - base0= 1
  - baseexp+1= base * baseexp
- New algorithm will make use of this rule:
  - basem*n= (basem)n
  - Let m = 2, n = exp/2

```
public double fastPower(double base, intexponent)
{
}
```

Faster exponentiation

```
public double fastPower(double base, int exponent)
{
    if (exponent == 0)
    {
        return 1;
    } else if ( exponent%2 == 1 )
    {
        return base * fastPower(base, exponent-1);
    } else
    {
        return fastPower(base* base, exponent/2);
    }
}
```

Tracing fastPower

fastPower(3, 16)

= fastPower(9, 8)

= fastPower(81, 4)

= fastPower(6561, 2)

= fastPower(43046721, 1)

= 43046721 * fastPower(43046721, 0)

= 43046721 * 1

= 43046721

Only 5 multiplications! Division by 2 is fast and easy for computers