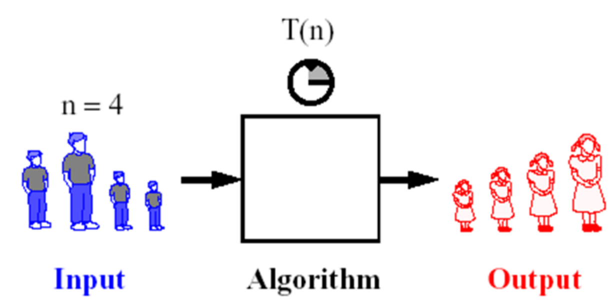
Mathematical Review

Many of my Color slides

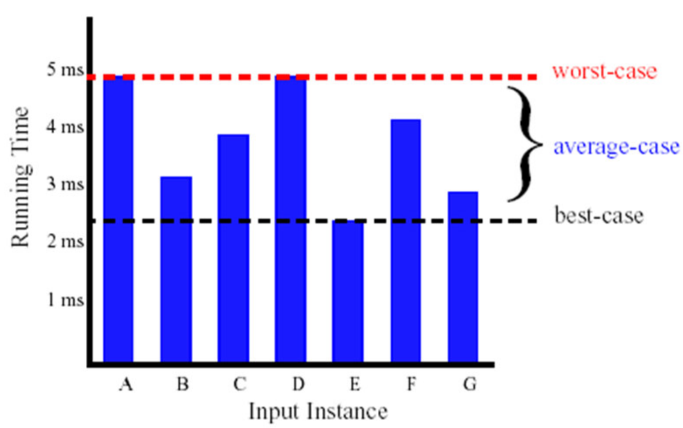
- were produced by Michael Goodrich and Roberto Tomassia, to go with their *Data Structures and Algorithms in JAVA* book, which is on the recommended reading list for the course.
- are mainly here for your reference. We will only dwell on one if you ask about it.



Running Times

- Algorithms may have different *time complexity* on different data sets
- What do we mean by "Worst Case" time complexity?
- What do we mean by "Average Case" time complexity?
- What are some application domains where knowing the Worst Case time complexity would be important?

Average Case and Worst Case



Quick Math Review 0

- Floor

$\lfloor x \rfloor$ = the largest integer $\leq$ x

- Ceiling

$\lceil x \rceil$ = the smallest integer $\geq$ x

•In `java.lang.Math`, there are static methods `floor()` and `ceil()`.

Math review 1

- Summations
  - general definition:

$$\sum_{i=s}^{t} f(i) = f(s) + f(s+1) + f(s+2) + \ldots + f(t)$$

  - where f is a function, s is the start index, and t is the end index

Math review 2

- Summations
  - general definition:

$$\sum_{i=s}^{t} f(i) = f(s) + f(s+1) + f(s+2) + \ldots + f(t)$$

  - where f is a function, s is the start index, and t is the end index

- Geometric progression: $f(i) = a^i$
  - given an integer $n \geq 0$ and a real number $0 < a \neq 1$

You will show this by induction later.

$$\sum_{i=0}^{n} a^i = 1 + a + a^2 + \ldots + a^n = \frac{1 - a^{n+1}}{1 - a}$$

  - geometric progressions exhibit exponential growth

Exercise: What is $\sum_{i=2}^{6} 3^i$?

(use the above formula)

Math Review 3

- You will probably use a geometric series sum in the analysis of the growable array algorithm, which you will do shortly.

Math Review 4

- Arithmetic progressions:
  - An example

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n = \frac{n^2 + n}{2}$$

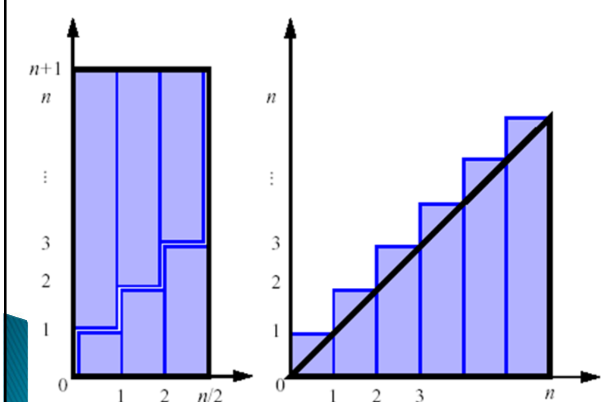
Exercise: $\sum_{i=21}^{40} i$

(Do it by the formula)

Math Review 5 – a visual proof

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n = \frac{n^2 + n}{2}$$

- two visual representations



An example where this sum is relevant

Selection sort

```
for (i=n-1; i>0; i--) {
  find the largest element among  a[o] ... a[i] ;
  exchange the largest element with a[i] ;
}
```

- **How many comparisons of array elements are done?**
- **How many times are array elements copied?**

(When you think you have the answers, compare with a partner)

More Math Review 1

- properties of exponentials:

$$a^{(b+c)} = a^b a^c$$

$$a^{bc} = (a^b)^c$$

$$a^b/a^c = a^{(b-c)}$$

$$b = a^{\log_a b}$$

$$b^c = a^{c*\log_a b}$$

More Math Review 2

- Logarithms and Exponents
  - properties of logarithms:

$$\log_b(xy) = \log_b x + \log_b y$$

$$\log_b(x/y) = \log_b x - \log_b y$$

$$\log_b x^{\alpha} = \alpha \log_b x$$

$$\log_b x = \frac{\log_a x}{\log_a b}$$

Practice with Exponentials and logs

(Do these with a friend after class, not to turn in)

Simplify: Note that log x (without a specified) base means $\log_2 x$. Also, log n is an abbreviation for log(n).

1. $\log(2\, n \log n)$
2. $\log(n/2)$
3. $\log(\text{sqrt}(n))$
4. $\log(\log(\text{sqrt}(n)))$
5. $\log_4 n$
6. $2^{2 \log n}$
7. $n^2 2^{3 \log n}$
8. if $N=2^{3k} - 1$, solve for k.

Where do logs come from in algorithm analysis?

Growable array analysis

Growable array exercise

- From pages 41-43 of Weiss DS.
- Read Strings from a text file (one per line) and place them into an array.
- We don't know in advance how many strings there will be.
- Start with an array of size 5 and grow it as needed (*via* calls to resize()).
  - Can we just add elements onto the end of an existing array?
- We want to measure the overhead involved.
  - If we insert N Strings altogether, how many times do we have to copy an array element during all of the calls to resize()?

```java
public class ReadStrings {  // Edited for brevity
    public static void main( String [ ] args ) {
        String [ ] array = getStrings( );
        for( int i = 0; i < array.length; i++ )
            System.out.println( array[ i ] );
    }

    // Read an unlimited number of String; return a String [ ]
    public static String [ ] getStrings( ) {
        BufferedReader in = new BufferedReader(
            new InputStreamReader(System.in));
        String [ ] array = new String[5];
        int itemsRead = 0;
        String oneLine;

        try {
          while( ( oneLine = in.readLine( ) ) != null)  {
             if( itemsRead == array.length )
               array = resize( array, array.length * 2 );
             array[ itemsRead++ ] = oneLine;
          }
        } catch( IOException e ) { /* details omitted */ }
        return resize( array, itemsRead );
    }
```

Magic incantation for line-at-a-time-reading

Original array size = 5

Read however many input lines there are.

Grow when necessary

How does **`resize()`** work?

What is the main "overhead cost" of resizing?