Programming and Algorithms

Bob Gross, Bio 39/139

Computer Programs

- provide instructions to the computer on what to do
- can be written at different levels
  - low level languages are more efficient but are more work for the programmer
  - high level languages are easier for programmer and often are intelligible by humans - usually need to be compiled
  - Compiling turns source code into an executable binary file that can be run by the computer

Some Computer Languages

- HTML
- BASIC
- Perl
- C++
- Java
- Javascript
- Machine language

Algorithm

is a list of well-defined instructions for completing a task

Opening a Door: Example 1

```
open the door
DONE
```

Opening a Door: Example 2

```
search until door found
  walk to door
  turn doorknob
  open door
  DONE
```

Very Low Level Language

- rotate head, examine what you are seeing
- identify door (need door definition)
- walk to door (need walk definition)
- adjust vision to examine parts of door (need adjust definition)
- identify doorknob (need doorknob definition)
- extend arm, open hand, grasp doorknob
- unlatch door bye twisting, squeezing or some other motion (need to determine which is appropriate action)
- either push or pull door (need to make decision)

Bob Gross, Bio 39/139

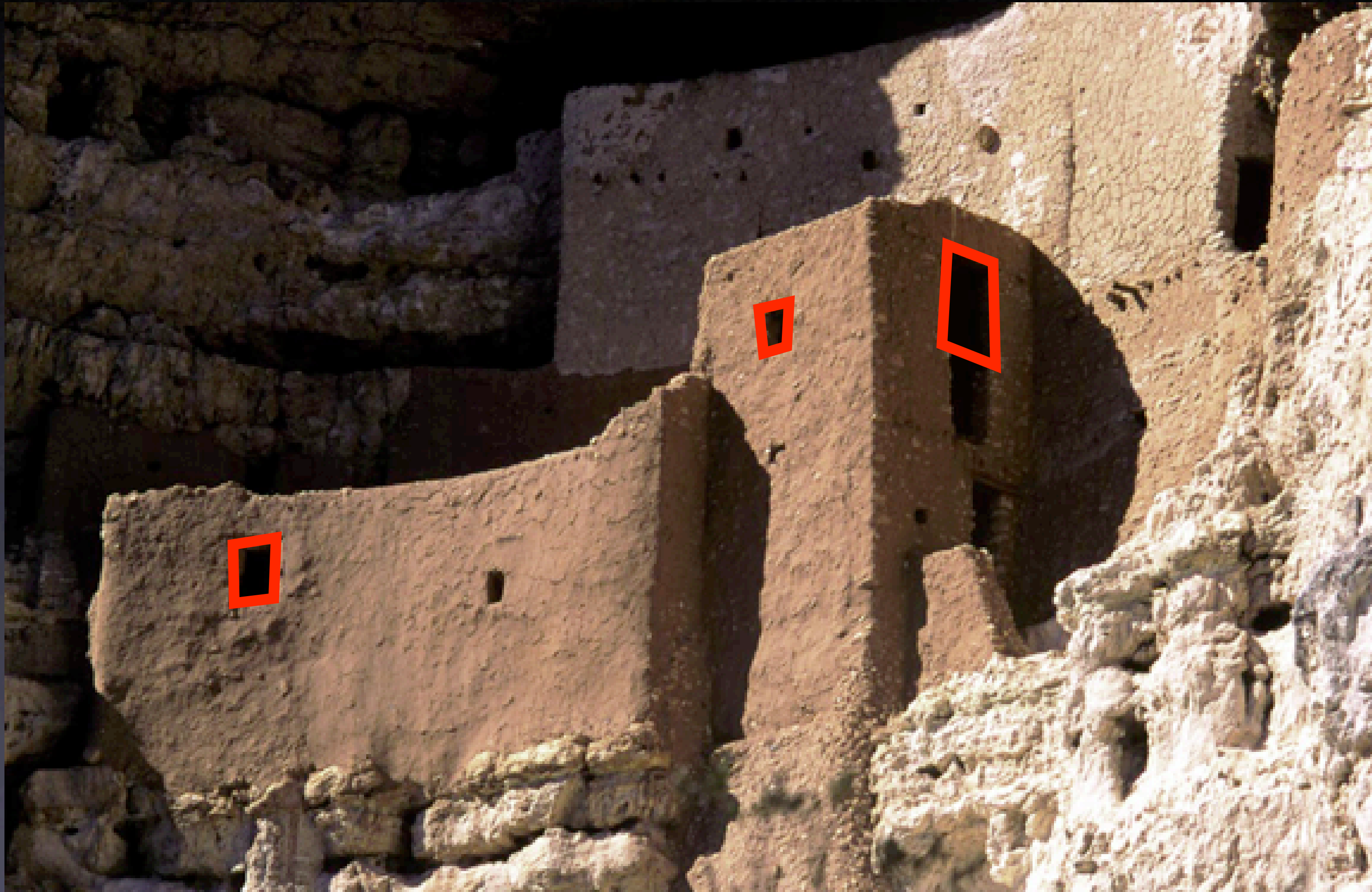
Opening a Door: Example 3

```
look straight ahead for a door
     if you see a door
        walk to door
        tilt your head to look at top of door
        look for doorknob
            if see a doorknob
               extend arm towards doorknob
               open hand
               grasp doorknob
               turn hand
               push door
                   if door opens
                      DONE
                   if door does not open
                      pull door
                      DONE
            if don't see doorknob
               look down 10°
     if you don't see a door
        rotate 10° to right
        start over at top
```

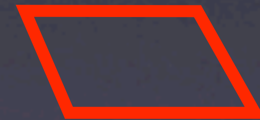
How Many Rectangles?

Are These Rectangles?

Are These Rectangles?

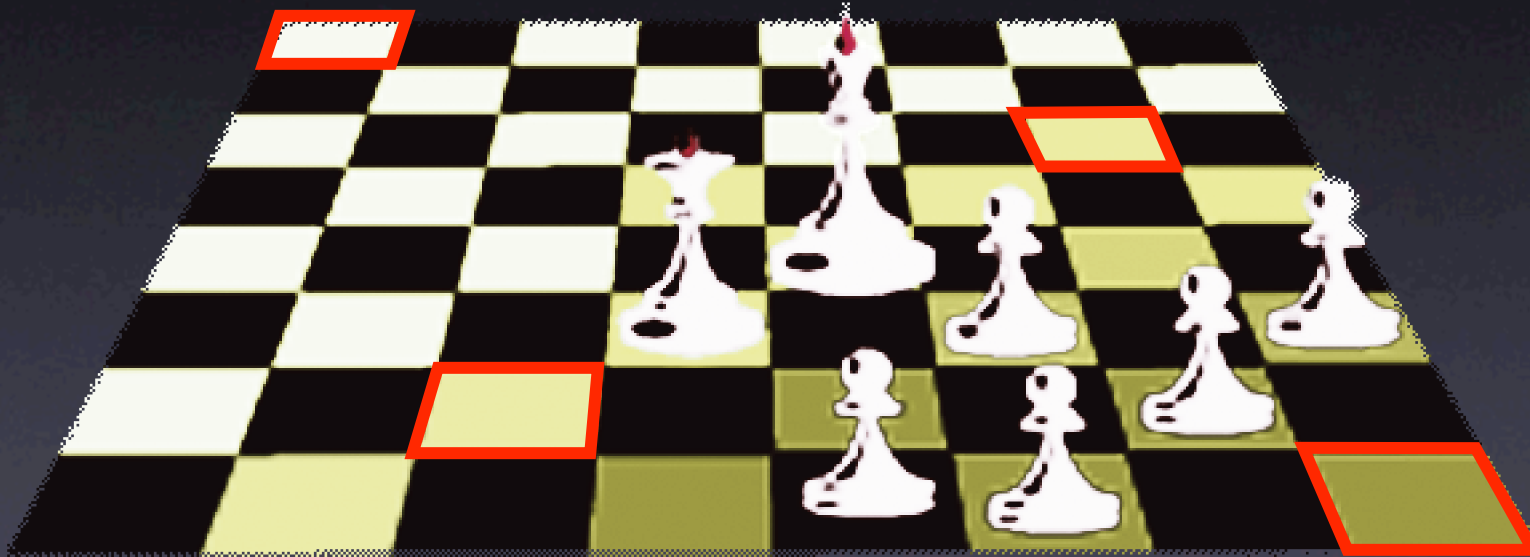


Are These "Rectangles" The Same Size?



Bob Gross, Bio 39/139

Are These "Rectangles" The Same Size?



Looking up a word in a dictionary

Finding a Word: Method 1

```
define variables
    p = 1 [current page number being examined]
    w = 1 [current word on page being examined]

open dictionary to page p
    examine word w
        if word w matches query
            DONE
        if word w does not match query
            if not at last word on page
                w = w + 1 (increment w by 1|add one to w|w++)
            if at last word on page
                p = p + 1
                w = 1
```

Finding a Word: Method 1a

```
define variables
    p = 1 [current page number being examined]
    w = 1 [current word on page being examined]

repeat until end of dictionary
open dictionary to page p
    examine word w
        if word w matches query
            DONE
        if word w does not match query
            if not at last word on page
                w = w + 1 (increment w by 1, add one to w, w++)
            if at last word on page
                p = p + 1
                w = 1
```

Finding a Word: Method 2

```
define variables
    w = 1 [word number on page]
    first = 1 [first page number to examine]
    last = total number of pages in dictionary [last page to examine]
    p = a number between first and last

open dictionary to page p
    examine word w
      if word w matches query
        DONE
      if word w does not match query
        if word w is alphabetically AFTER query word
          last = p
          p = p - (p-first)/2 [go half way back to beginning]
          w = 1
        if word w is alphabetically BEFORE query word
          w = w + 1
          if new w is on next page
            first = p
            p = (last-p)/2 + p [go halfway to end]
            w = 1
```

Finding a Word: Method 2

```
define variables
    w = 1 [word number on page]
    first = 1 [first page number to examine]
    last = total number of pages in dictionary [last page to examine]
    p = a number between first and last

open dictionary to page p
    examine word w
      if word w matches query
        DONE
      if word w does not match query
        if word w is alphabetically AFTER query word
          last = p
          p = p - (p-first)/2 [go half way back to beginning]
          w = 1
        if word w is alphabetically BEFORE query word
          w = w + 1  (could check last word on page to speed things up)
          if new w is on next page
            first = p
            p = (last-p)/2 + p [go halfway to end]
            w = 1
```

Finding a Word: Method 3

```
define variables
    w = 1
    first = page number of first tab
    last = page number of tab following the first tab
    p = a number between first and last

select TAB in dictionary that has same first letter as
your query word

use method 2 with w, first, last, p
```

Base Composition: Method 1

```
define variables
     Atot=0,Ctot=0,Gtot=0,Ttot=0      'total occurrences for each base
     n = 1          'position of current nucleotide being examined
     len = length of sequence

look at nucleotide at position n
     if nucleotide at n is 'A'
        Atot = Atot + 1
     if nucleotide at n is 'C'
        Ctot = Ctot + 1
     if nucleotide at n is 'G'
        Gtot = Gtot + 1
     if nucleotide at n is 'T'
        Ttot = Ttot + 1
     n = n + 1
     if n > len
        DONE          'no more bases left to count

values in Atot, Ctot, Gtot, and Ttot represent the base composition
```

Base Composition: Method 2

```
define variables
Alist="",Clist="",Glist="",Tlist=""  'these are string variables, not numbers
Atot=0,Ctot=0,Gtot=0,Ttot=0
n = 1          'position of current nucleotide being examined
len = length of sequence
nuc = identification of the current nucleotide being examined [A, C, G, T]

nuc = nucleotide at position n
     if nuc is an 'A'
        Atot = Atot + 1
        APPEND n to Alist      'assume commas are placed in list as delimiters
     if nuc is a 'C'
        Ctot = Ctot + 1
        APPEND n to Clist
     if nuc is a 'G'
        Gtot = Gtot + 1
        APPEND n to Glist
     if nuc is a 'T'
        Ttot = Ttot + 1
        APPEND n to Tlist
     n = n + 1
     if n > len
        DONE
Atot = number of entries in Alist, Ctot = number of entries in Clist
Gtot = number of entries in Glist, Ttot = number of entries in Tlist
```

Object Oriented Programming

- modern way of writing code
- large complex programs can be put together by combining individual units of code called objects
- each object performs a specific function and operates independently of other objects
- objects can interact with each other through messages
- easy maintenance of code - can change one object without affecting rest of program
- objects can be reused

Base Composition Example

```
main program
    call input object
    call counting object
    call output object(count)
```

`input object` - interacts with user to obtain a query string [e.g. 'A'] and a target string [sequence]

`counting object` - counts the number of times a query occurs within a target (gets query and target from input object)

`output object` - displays the results of the analysis (gets results to display from counting object). This object can be told to display a count or a list of positions or both by passing a parameter (in parentheses).

Execution Speed

- depends on CPU speed
- depends on algorithm
- for large datasets, speed of hard disk is important
- can utilize parallel processing to improve speed for some applications