Recursion & Image Analysis Cont'd

A note on the Fibonacci sequence

- The recursive Fibonacci generator is only intended to get you the n’th Fibonacci number, not the whole sequence:

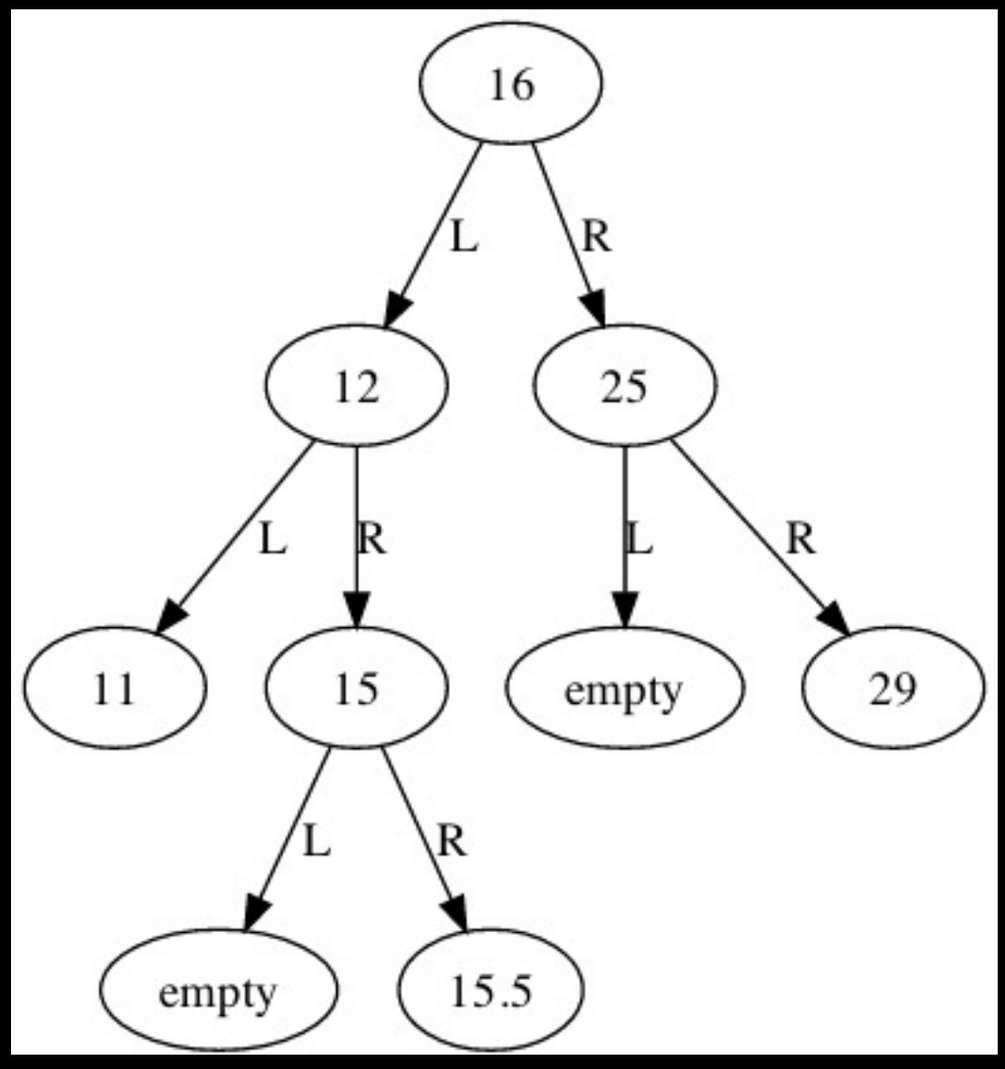
$$F_n = F_{n-1} + F_{n-2}$$

or

$$f(n) = f(n-1) + f(n-2)$$

Parsing a Binary Tree

- Sidenote: duplicates are OK



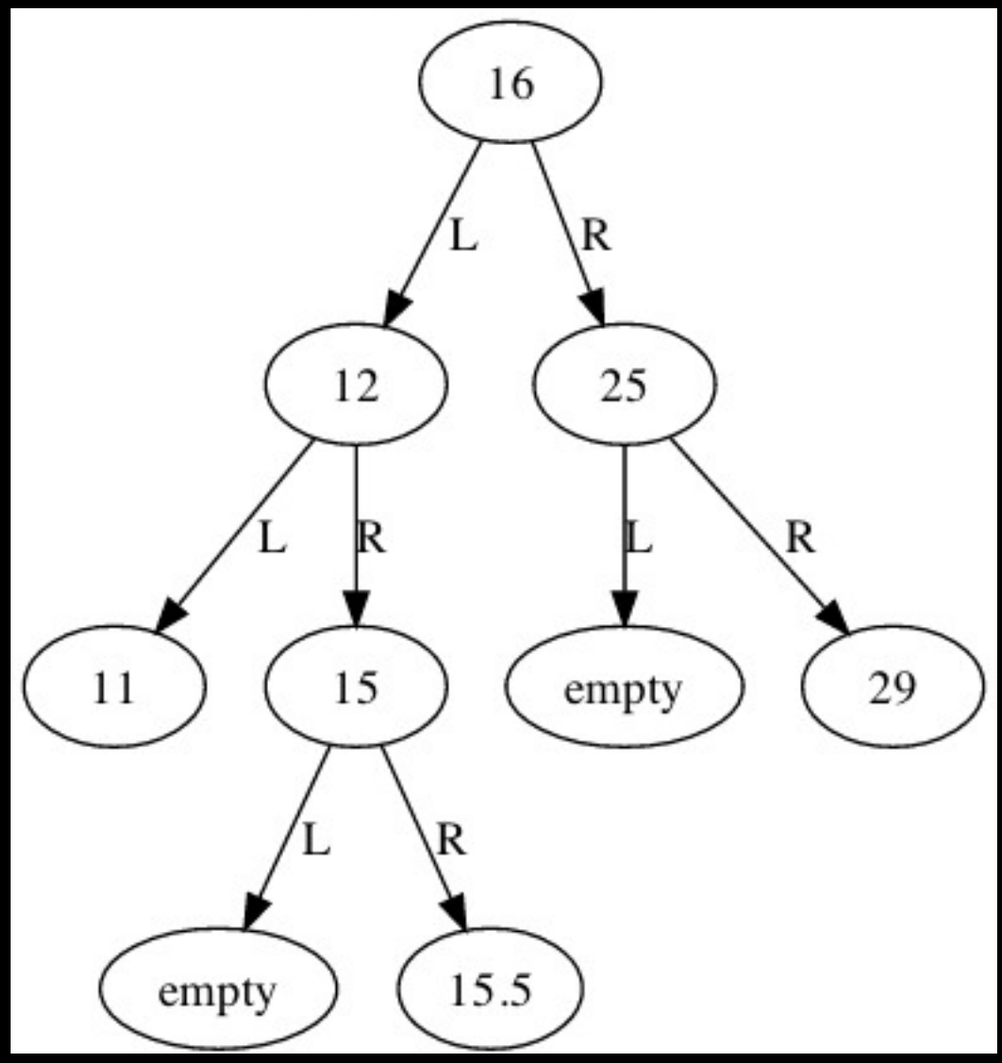
Binary Tree

- Just like a linked list:

```
; binary tree structure definition
pro binary_tree__define
    dummy = {binary_tree,$
        left: ptr_new(),$
        right: ptr_new(),$
        data:0.0}
end
```

Traversal

```
; Read a binary tree in sorted order into an array
function traverse_tree,binary_tree
    if binary_tree eq !null then return,[]
    return,[traverse_tree((*binary_tree).left,nsteps),$
            (*binary_tree).data,$
            traverse_tree((*binary_tree).right,nsteps)]
end
```

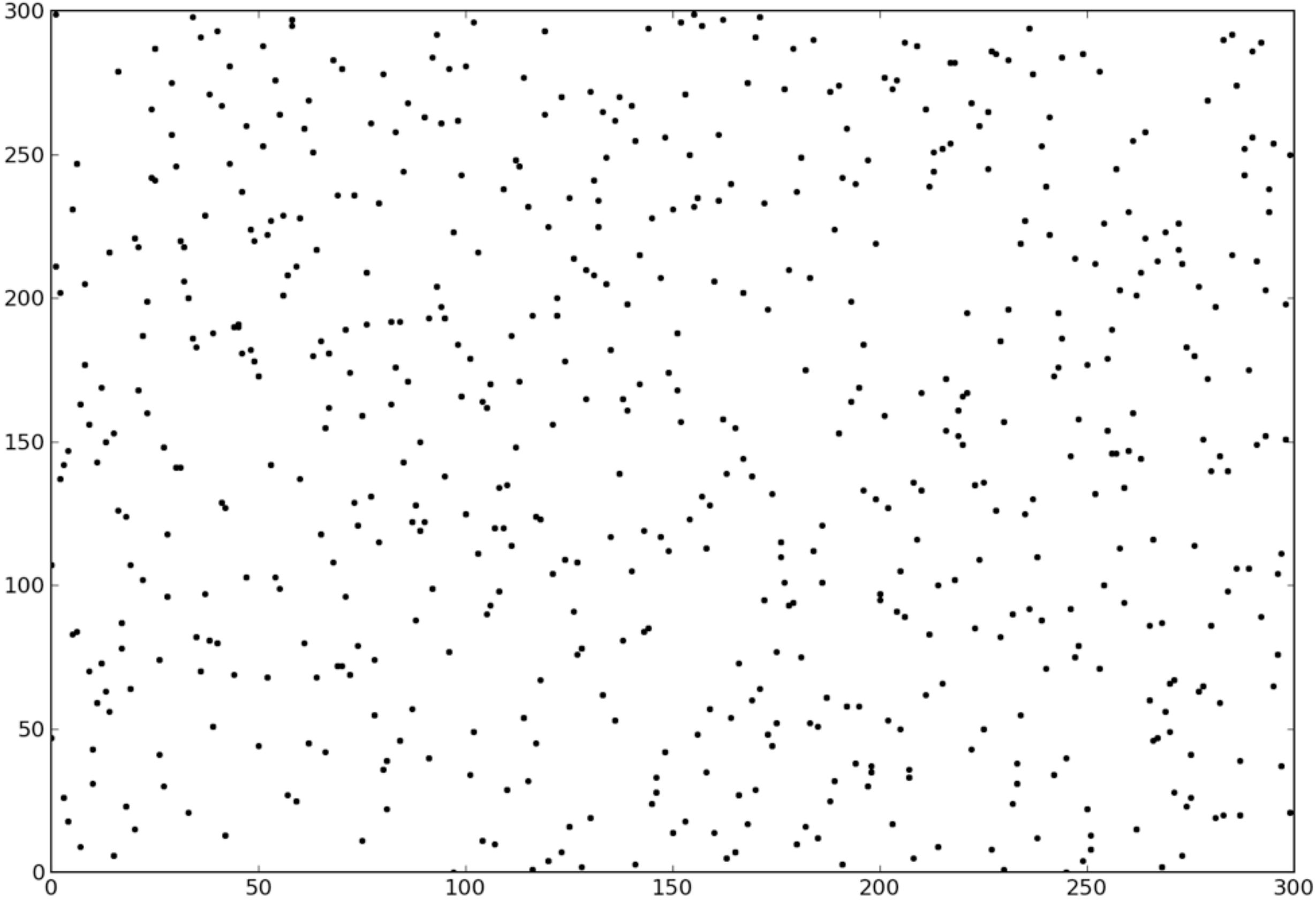


New Elements

```
; Add new data onto a binary tree in sorted order
pro add_to_tree,binary_tree,data
    if binary_tree eq !null then return
    if (*binary_tree).data gt data then begin
        if (*binary_tree).left eq !null then begin
            (*binary_tree).left = ptr_new({binary_tree})
            (*(*binary_tree).left).data = data
        endif else begin
            add_to_tree,(*binary_tree).left,data,nsteps
        endelse
    endif else begin
        if (*binary_tree).right eq !null then begin
            (*binary_tree).right = ptr_new({binary_tree})
            (*(*binary_tree).right).data = data
        endif else begin
            add_to_tree,(*binary_tree).right,data,nsteps
        endelse
    endelse
end
```

Sorting

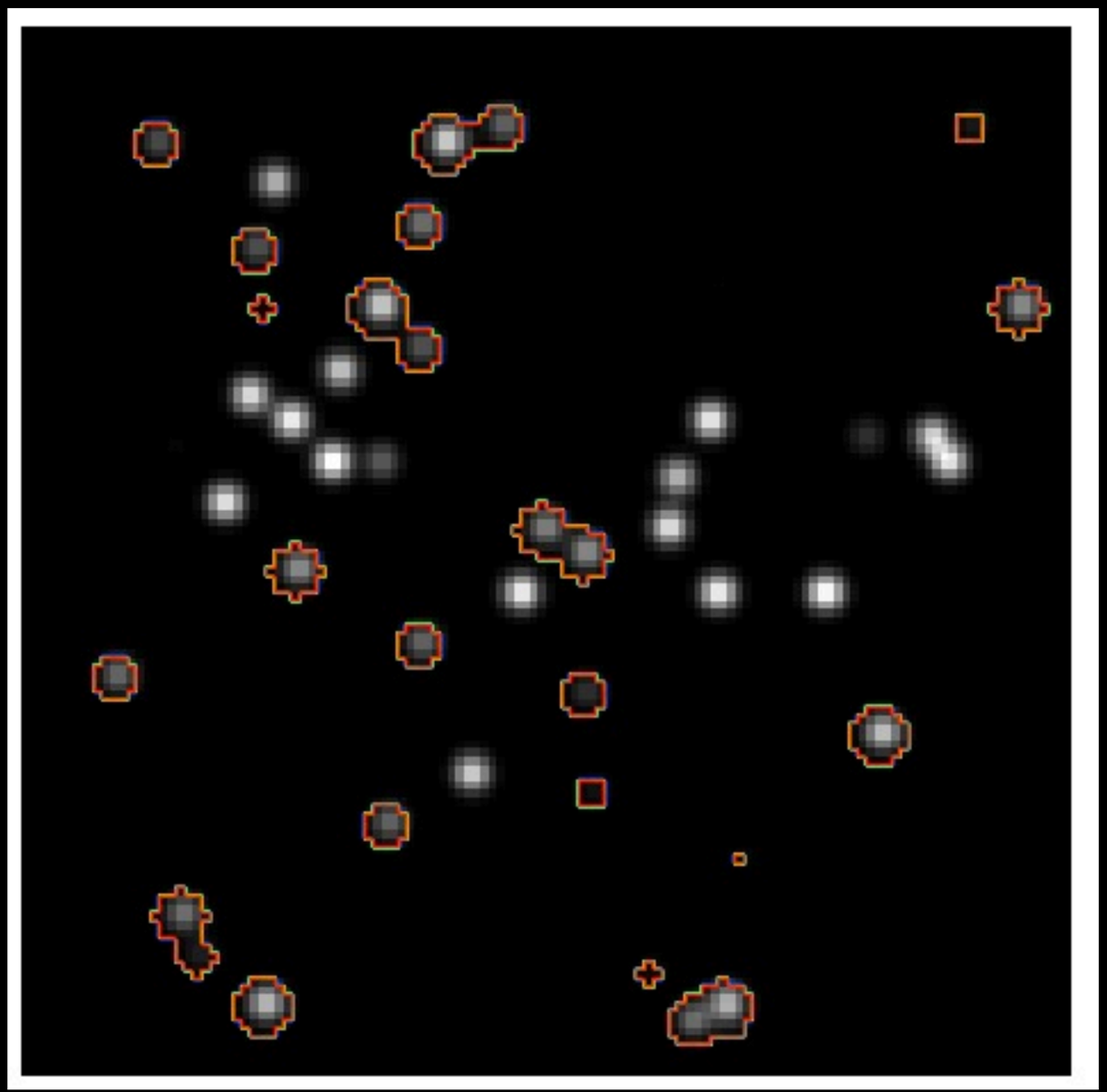
- If you'd taken a normal computer science class, you'd probably spend a lot of time working on sorting algorithms
- Sorting is something computers do well, but there are algorithms that are, in most cases, better than others
- Binary Sort is one of the fastest…
- Insertion Sort is slow but easy to understand

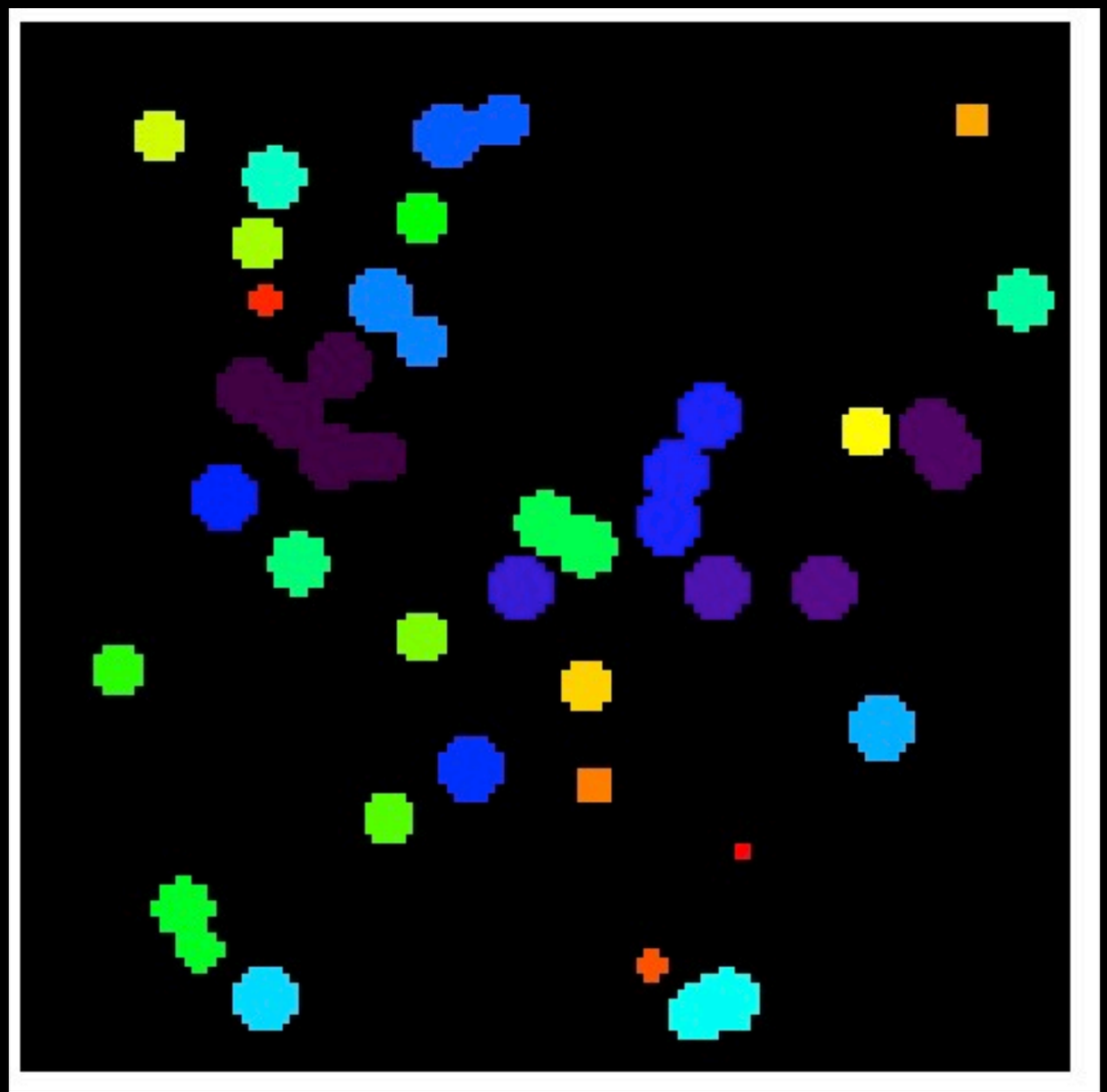

300
250
200
150
100
50
0
0
50
100
150
200
250
300

Star Identification

- Need to identify pixels associated with stars
  - Use a “mask” that is 1 where a pixel has starlight and 0 where it does not
- Then you look for continuous regions in the *mask* using recursion

Masking Images



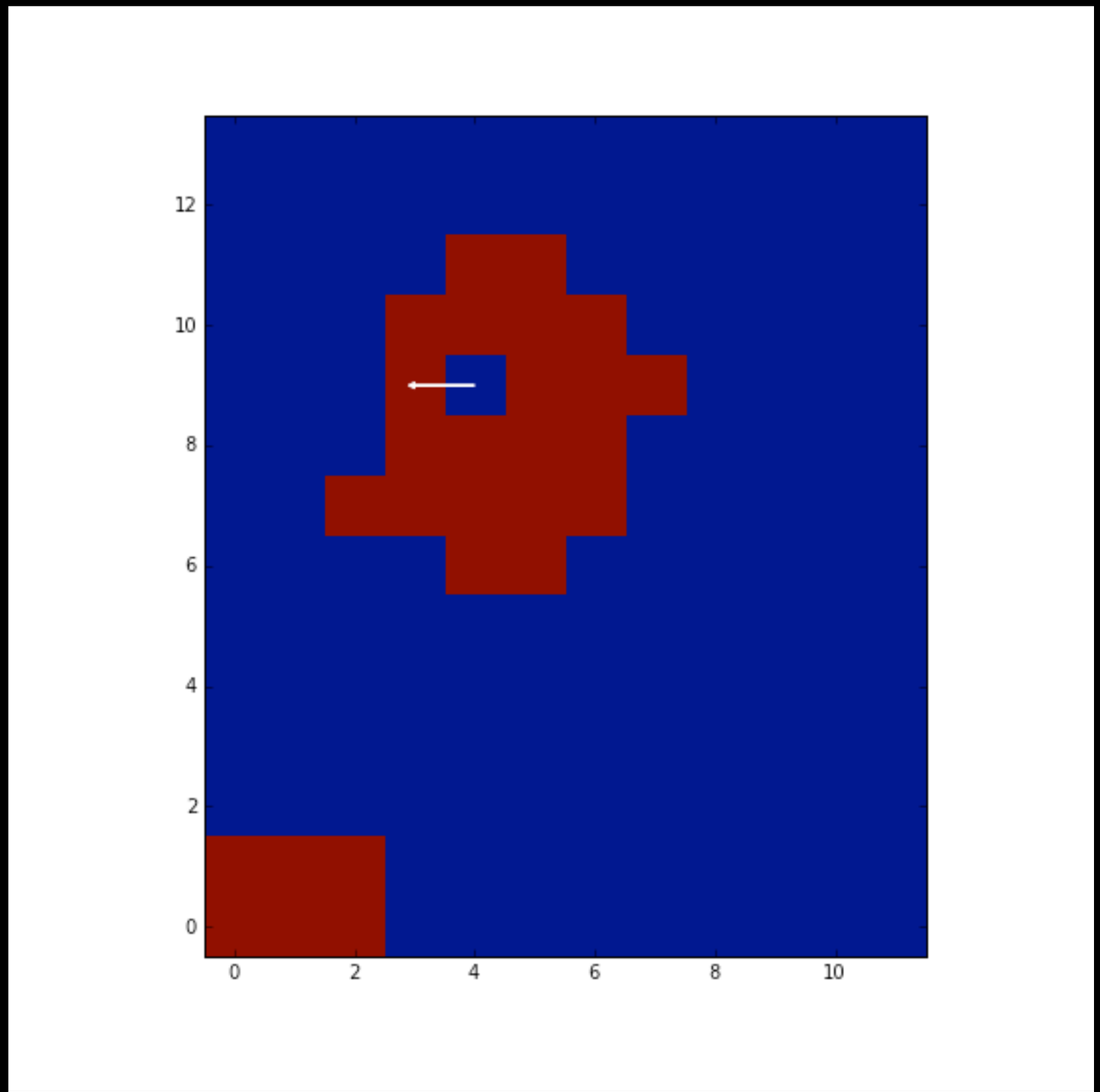


DEMO

Identifying Pixels

- Start with a pixel you know is in the star
- Let loose a recursive finder that examines each pixel with the appropriate mask value
- Finder needs to keep track of where it has been and explore all directions

Identifying Pixels



Zooming In

- To make an image look larger in IDL, you actually have to give it more pixels
- For 128x128 images, we can scale them to 512x512 like so:
  - `big = congrid(small,512,512)`

Centroid Trick

- Say you have an image with dimensions [10,10]
- Make its x and y axes:
- `x = findgen(10)`
- `y = findgen(10)`
- You can then use `total` to get the centroid like so:
- `xc = total(x*(total(img,2))) / total(img)`
- `yc = total(y*(total(img,1))) / total(img)`

Next Classes

- Will cover more useful features for your final projects
- Any specific requests?

Tutorial

- Both recursion & image analysis
- Need to install `astron` for the 2nd part