Programming Abstractions

CS106B

Cynthia Bailey Lee
Julie Zelenski

Today's Topics

Recursion Week continues!

- Today, two applications of recursion:
  - Binary Search (one of the fundamental algorithms of CS)
    - We saw the idea of this on Wed, but today we'll code it up
  - Fractals (will help us visualize the order of operations in recursion)

Next time:

- More recursion! It's Recursion Week!
- Like Shark Week, but more nerdy

Binary Search Refresher

(RECALL FROM WEDNESDAY'S LECTURE)

Binary search (*refresher*)

The question we're trying to answer is, given a list of numbers, does this list contain some particular value, or not? For convenience, we have kept our list **sorted**.

0	1	2	3	4	5	6	7	8	9	10
2	7	8	13	25	29	33	51	89	90	95

If instead we **jump right to the middle**, one of three things can happen:

1. The middle one happens to be the number we were looking for, yay!
2. We realize we went too far
3. We realize we didn't go far enough

Binary search (*refresher*)

The question we're trying to answer is, given a list of numbers, does this list contain some particular value, or not? For convenience, we have kept our list **sorted**.

0	1	2	3	4	5	6	7	8	9	10
2	7	8	13	25	29	33	51	89	90	95

Let's say the answer was case 3, "we didn't go far enough"

- We ruled out the entire first half, and now only have the second half to search
- We could start at the front of the second half and proceed forward checking each item one at a time… but why do that when we know we have a **better way**?

Jump right to the middle of the region to search

Binary search (*refresher*)

The question we're trying to answer is, given a list of numbers, does this list contain some particular value, or not? For convenience, we have kept our list **sorted**.

0	1	2	3	4	5	6	7	8	9	10
2	7	8	13	25	29	33	51	89	90	95

Let's say the an[illegible]er wa[illegible] we[illegible]dn't go far enough"

- We ruled out [illegible]ow only have the second h[illegible]
- We co[illegible]cond half and proceed forward [illegible]e at a time… but why do that when w[illegible] we have a **better way**?

Jump right to the middle of the region to search

Binary Search Implementation

NOW WE UNDERSTAND THE APPROACH.

WHAT DOES THE CODE LOOK LIKE?

```
bool binarySearch(Vector<int>& data, int key) {
  // want to keep passing same data by reference for efficiency,
  // but then how do we cut in half?
  return binarySearch(data, key, 0, data.size() - 1); // 2 new params
}

bool binarySearch(Vector<int>& data, int key, int start, int end) {









}
```

Recursive Function Design Tip: Wrapper function

- When we want to write a recursive function that needs more book-keeping data passed around than an outsider user would want to worry about, do this:

1. Write the function as you need to for correctness, using any extra book-keeping parameters you like, in whatever way you like.
2. Make a second function that the outside world sees, using only the minimum number of parameters, and have it do nothing but call the recursive one.
   - Called a "wrapper" function because it's like pretty outer packaging.

```
bool binarySearch(Vector<int>& data, int key) {
  // want to keep passing same data by reference for efficiency,
  // but then how do we cut in half?
  return binarySearch(data, key, 0, data.size() - 1); // 2 new params
}

bool binarySearch(Vector<int>& data, int key, int start, int end) {
    if (start > end) {
        return false;
    }
    int mid = (start + end) / 2;
    if (key == data[mid]) {
        return true;
    } else if (key < data[mid]) {
        return binarySearch(data, key, _________, _________);
    } else {
        return binarySearch(data, key, _________, _________);
    }
}
```

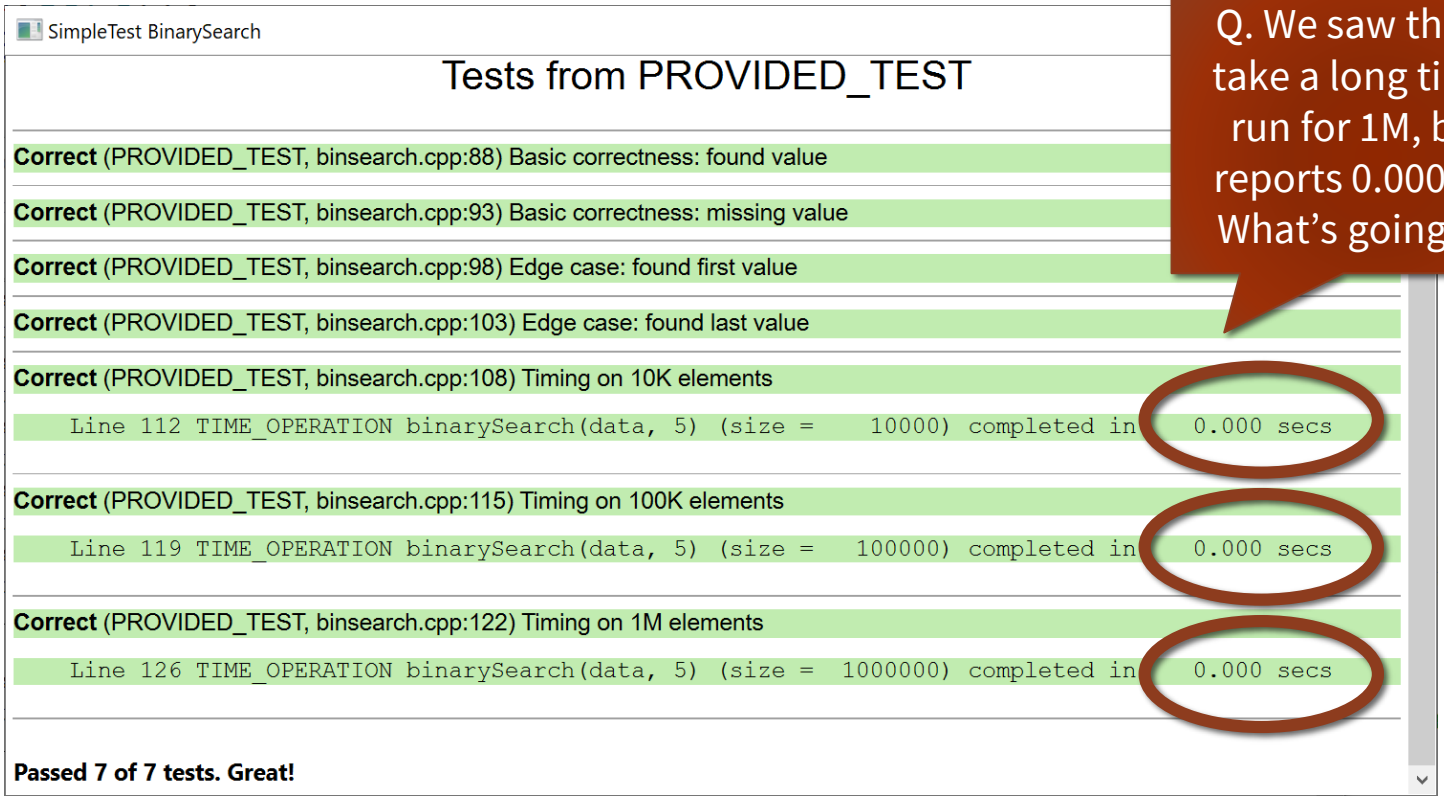
```
bool binarySearch(Vector<int>& data, int key) {
  // want to keep passing same data by reference for efficiency,
  // but then how do we cut in half?
  return binarySearch(data, key, 0, data.size() - 1); // 2 new params
}

bool binarySearch(Vector<int>& data, int key, int start, int end) {
    if (start > end) {
        return false;
    }
    int mid = (start + end) / 2;
    if (key == data[mid]) {
        return true;
    } else if (key < data[mid]) {
        return binarySearch(data, key, _________, _________);
    } else {
        return binarySearch(data, key, _________, _________);
    }
}
```

Your Turn:

What goes on the blanks below, to divide the remaining searchable region of our vector in half?

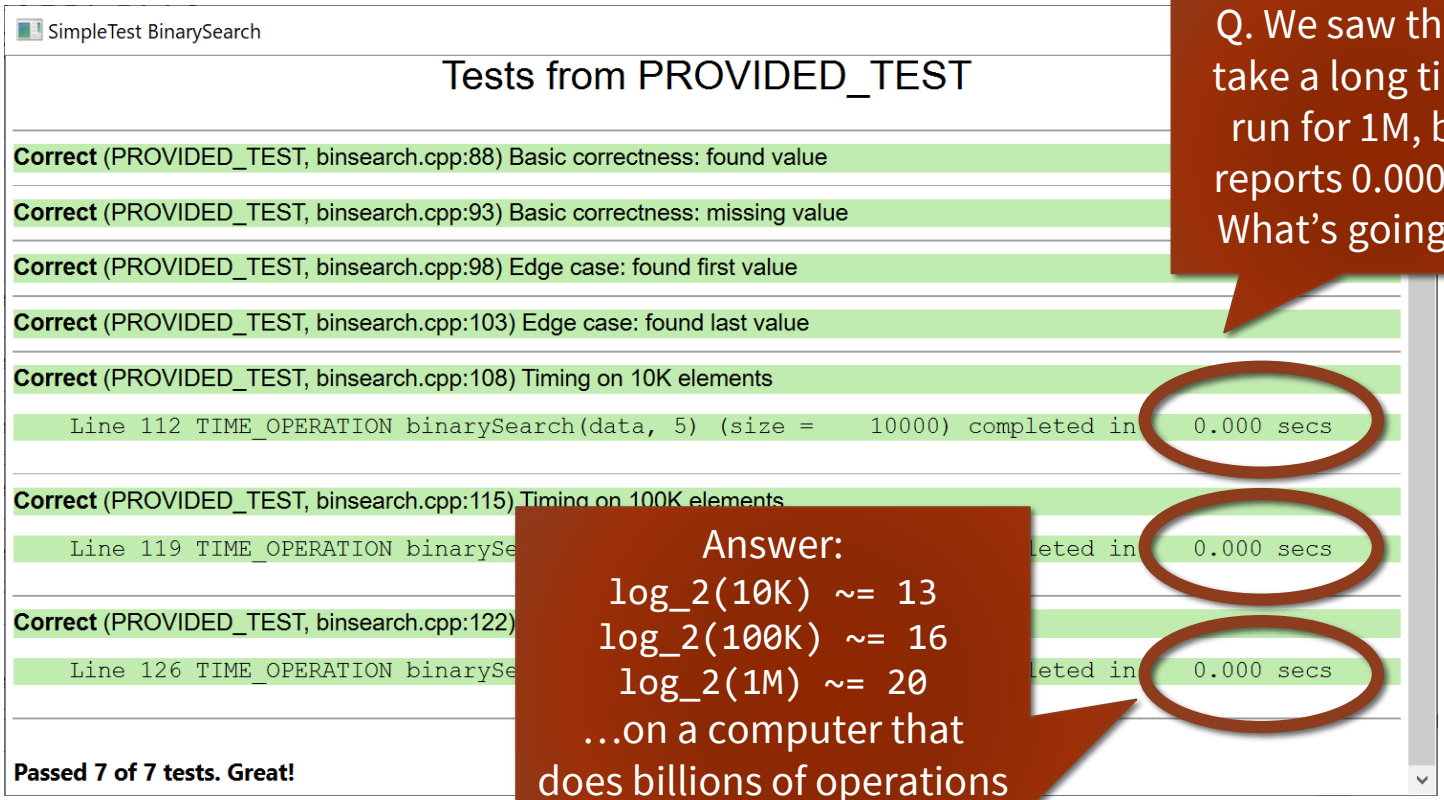
```
bool binarySearch(const Vector<int>& data, int key) {
  // want to keep passing same data by reference for efficiency,
  // but then how do we cut in half?
  return binarySearch(data, key, 0, data.size() - 1); // 2 new params
}

bool binarySearch(const Vector<int>& data, int key, int start, int end) {
    if (start > end) {
        return false;
    }
    int mid = (start + end) / 2;
    if (key == data[mid]) {
        return true;
    } else if (key < data[mid]) {
        return binarySearch(data, key, start, mid - 1);
    } else {
        return binarySearch(data, key, mid + 1, end);
    }
}
```

Binary Search performance



Q. We saw the test take a long time to run for 1M, but it reports 0.000 secs. What's going on??

Binary Search performance



Q. We saw the test take a long time to run for 1M, but it reports 0.000 secs. What's going on??

Answer:
`log_2(10K) ~= 13`
`log_2(100K) ~= 16`
`log_2(1M) ~= 20`
…on a computer that does billions of operations per second!

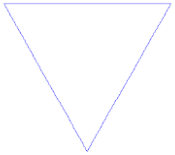
Fractals

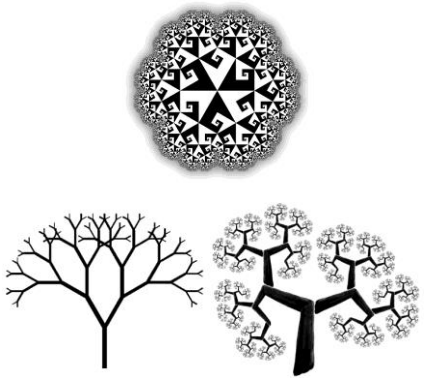
PRETTY!

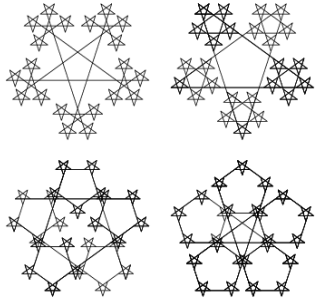
Fractals

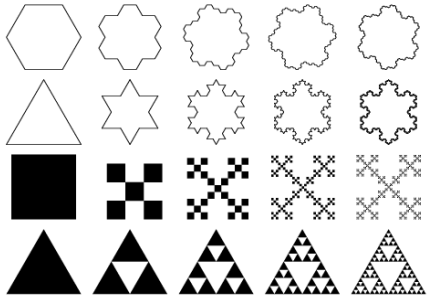
fractal: A self-similar mathematical set that can often be drawn as a recurring graphical pattern.

- Smaller instances of the same shape or pattern occur within the pattern itself.
- When displayed on a computer screen, it can be possible to infinitely zoom in/out of a fractal.



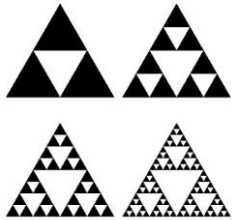




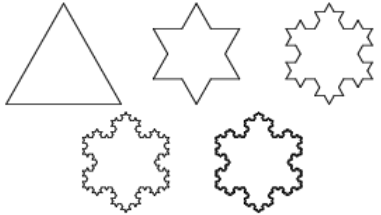


Example fractals

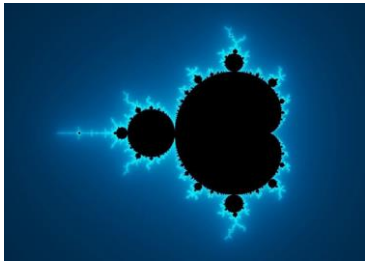
Sierpinski triangle: equilateral triangle contains smaller triangles inside it



Koch snowflake: a triangle with smaller triangles poking out of its sides



Mandelbrot set: circle with smaller circles on its edge



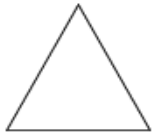
Coding a fractal

Many fractals are implemented as a function that accepts x/y coordinates, size, and a level parameter.

- The level is the number of recurrences of the pattern to draw.

Example, Koch snowflake:

- `snowflake(window, x, y, size, 1);`



- `snowflake(window, x, y, size, 2);`

- `snowflake(window, x, y, size, 3);`

Cantor Set

The Cantor Set is a simple fractal that begins with a line segment.

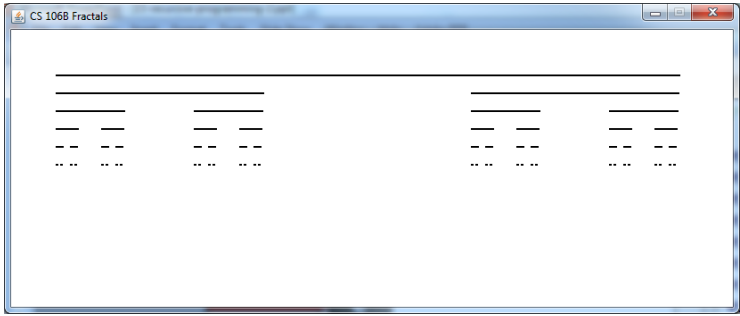
- At each level, the middle third of the segment is removed.
- In the next level, the middle third of each third is removed.

Write a function cantorSet that draws a Cantor Set with a given number of levels (lines) at a given position/size.

- Place 20 px of vertical space between levels.

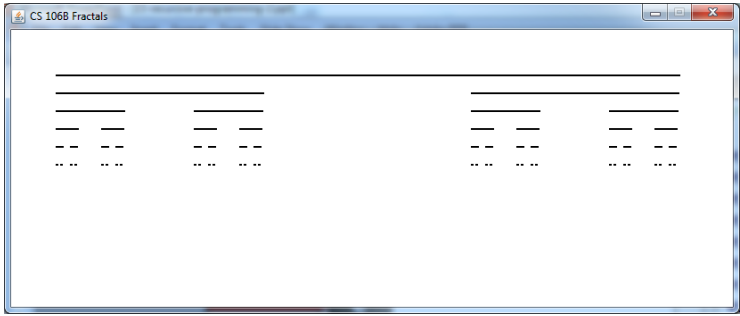
Cantor Set solution

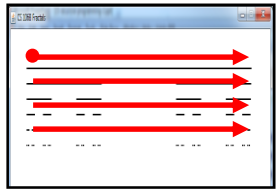
```
void cantorSet(GWindow& window, int x, int y, int length, int levels)
{
    if (levels > 0) {
        // draw our own line
        drawThickLine(window, x, y, length, levels);

        // recursively draw next lines
        int newY = y + LINE_SPACING;
        int newLength = length / 3;
        int newLevels = levels - 1;
        // left third
        cantorSet(window, x, newY, newLength, newLevels);
        // right third
        cantorSet(window, x + (2 * length / 3), newY, newLength, newLevels);
    }
}
```



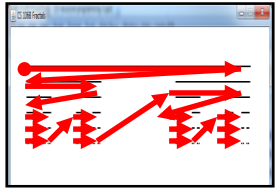
Your Turn: In what order does the recursion draw the lines?

```
void cantorSet(GWindow& window, int x, int y, int length, int levels)
{
    if (levels > 0) {
        // draw our own line
        drawThickLine(window, x, y, length, levels);

        // recursively draw next lines
        int newY = y + LINE_SPACING;
        int newLength = length / 3;
        int newLevels = levels - 1;
        // left third
        cantorSet(window, x, newY, newLength, newLevels);
        // right third
        cantorSet(window, x + (2 * length / 3), newY, newLength, newLevels);
    }
}
```

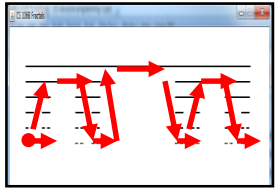




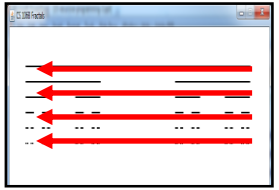
(A)



(B)



(C)



(D)

(E) other