Example: badpassword II: the recursening

Goal: Replace all e’s with 3’s

What is the base case?

What is the recursion case?

Trace how this program works with the input “feet”
Example: badpassword II: the recursening

`rbadpassword("feet")`

= “f” + `rbadpassword("eet")`
= “f” + [“3” + `rbadpassword("et")"]
= “f” + [“3” + [“3” + `rbadpassword("t")"]]
= “f” + [“3” + [“3” + [“t” + `rbadpassword(""")"]]]
= “f” + [“3” + [“3” + [“t” + “"]]]
= “f” + [“3” + [“3” + “t”]]
= “f” + [“3” + “3t”]
= “f” + “33t”
= “f33t”
Example: replace all letters with “X”

Trace how this program works with the input “cat”
Example: replace all letters with “X”

```
rreplaceX("cat")
```

```
= "X" + rreplaceX("at")
= "X" + ["X" + rreplaceX("t")]
= "X" + ["X" + ["X" + rreplaceX("")]]
= "X" + ["X" + ["X" + ""]]
= "X" + ["X" + "X"]
= "X" + ["XX"]
= "XXX"
```
Example: Check if all elements are even

Trace the input when \( L = [2, 4, 3] \) and when \( L = [2, 4, 0] \)
Example: Check if all elements are even

\[\text{rallEven}([2,4,3])\]

\[= (2 \mod 2 == 0) \text{ and } \text{rallEven}([4,3])\]
\[= (2 \mod 2 == 0) \text{ and } [(4 \mod 2 == 0) \text{ and } \text{rallEven}([3])]\]
\[= (2 \mod 2 == 0) \text{ and } [(4 \mod 2 == 0) \text{ and } [(3 \mod 2 == 0)]]\]
\[= True \text{ and } [True \text{ and } False]\]
\[= True \text{ and } [False]\]
\[= False\]
Example: Check if all elements are even

\[ \text{rallEven([2,4,0])} \]

\[ = (2 \mod 2 == 0) \text{ and } \text{rallEven([4,0])} \]
\[ = (2 \mod 2 == 0) \text{ and } [(4 \mod 2 == 0) \text{ and } \text{rallEven([0])}] \]
\[ = (2 \mod 2 == 0) \text{ and } [(4 \mod 2 == 0) \text{ and } [(0 \mod 2 == 0)] \]
\[ = \text{True and [True and True]} \]
\[ = \text{True and [True]} \]
\[ = \text{True} \]
Example: length of a string

Trace how this program works with the input “coffee”
Example: length of a string

`strlen(“coffee”)`

\[
\begin{align*}
&= 1 + strlen(“offee”) \\
&= 1 + 1 + strlen(“ffee”) \\
&= 1 + 1 + 1 + strlen(“fee”) \\
&= 1 + 1 + 1 + 1 + strlen(“ee”) \\
&= 1 + 1 + 1 + 1 + 1 + strlen(“e”) \\
&= 1 + 1 + 1 + 1 + 1 + 1 + strlen(“”) \\
&= 1 + 1 + 1 + 1 + 1 + 1 + 1 + 0 \\
&= 1 + 1 + 1 + 1 + 1 + 1 \\
&= 1 + 1 + 1 + 2 \\
&= 1 + 1 + 3 \\
&= 1 + 4 \\
&= 5 \\
&= 6
\end{align*}
\]
Recursive bullseye

```
from graphics import *

def bullseye(win, center, radius):
    while radius > 10:
        c = Circle(center, radius)
        c.draw(win)
        radius = radius - 10

def rbullseye(win, center, radius):
    if radius < 10:
        return
    c = Circle(center, radius)
    c.draw(win)
    rbullseye(win, center, radius - 10)

def main():
    win = GraphWin("Tower", 500, 500)
    rbullseye(win, Point(250,250), 200)
    win.getMouse()

main()
```
Recursive stained glass window

def rbullseyewin(win, color, center, radius):
    if radius < 10:
        return

    # jitter color
    jitter = [0, 0, 0]
    for i in range(len(color)):
        jitter[i] = color[i] + random.randint(-100, 100)
        jitter[i] = min(max(0, jitter[i]), 255)

    c = Circle(center, radius)
    c.setFill(color_rgb(jitter))
    c.draw(win)

    p1 = Point(center.getX() - radius/2, center.getY())
    rbullseyewin(win, color, p1, radius/2)

    p2 = Point(center.getX() + radius/2, center.getY())
    rbullseyewin(win, color, p2, radius/2)

    p3 = Point(center.getX(), center.getY() - radius/2)
    rbullseyewin(win, color, p3, radius/2)

    p4 = Point(center.getX(), center.getY() + radius/2)
    rbullseyewin(win, color, p4, radius/2)

def main():
    win = GraphWin("Circles", 500, 500)
    red = random.randint(0, 255)
    green = random.randint(0, 255)
    blue = random.randint(0, 255)
    rbullseyewin(win, [red, blue, green], Point(250, 250), 200)
    win.getMouse()

main()
Example: Recursive tower

```python
def rtower(win, center, size):
    if size < 10:
        return

    px = center.getX()
    py = center.getY()
    tl = Point(px - size, py - size)
    br = Point(px + size, py + size)
    rectangle = Rectangle(tl, br)
    rectangle.draw(win)
    py = py - 2*size
    size = size - 10
    rtower(win, Point(px, py), size)
```
Recursive binary search: rbinsearch(x, L)

Base cases:

- if len(L) is 0, return False
- if L[mid] == x, return True

Recursion case:

- if x < L[mid], look in the left sublist
- if x > L[mid], look in the right sublist
recursive binary search

What is the function stack for
L = [1,3,7,9,13, 15] and x = 13?

What is the function stack for
L = [1,3,7,9,13, 15] and x = 2?
L = [1,3,7,9,13, 15] and x = 13
L = [1,3,7,9,13, 15] and x = 2