A. The kernel `[[-1, -2, 0], [-2, 0, 2], [0, 2, 1]]` works. (This is not the only way to answer this question. In general, we probably want zeroes along the diagonal, positive numbers below the diagonal, and negative numbers above the diagonal.)

B.

a. `f`: function

b. `width`: integer

c. `row`: integer

d. `pixel`: instance (or object) (of class `cImage.Pixel`)

e. `rgb`: list (of three integers)

f. `f(rgb)`: list (of three integers)

C. In the `for` loop, the variable `i` counts from 1 to $n-1$; thus this loop fires $n-1$ times. In the `while` loop, the variable `hole` counts from `i` down to 1; because `i` is at most $n-1$, this loop fires at most $n-1$ times on each iteration of the `for` loop. The other lines of code represent constant-time operations. The total number of operations is something like

$$\begin{aligned}(n-1)(1+1+(n-1)2+1)+1 &= (n-1)(2n+1)+1\\ &= 2n^2-2n+n-1+1\\ &= 2n^2-n\\ &\leq 2n^2,\end{aligned}$$

which is $\mathcal{O}(n^2)$.

D. The encryption step computes $t^e\%n$, where t is the plaintext message, e is the encryption exponent, and n is the encryption modulus. The naive algorithm computes t^e using e multiplication operations, but the repeated squaring algorithm computes t^e using only $\log_2 e$ multiplications. Assuming that we use the more sophisticated algorithm (which isn't difficult to implement anyway), the encryption step is

$$\mathcal{O}(\log_2 e) = \mathcal{O}(\log e) = \mathcal{O}(E),$$

where E is the length of the binary representation of the number e.

E.

```
def listSum(a, b):
    if len(a) == 0:
        return []
    else:
        return [a[0] + b[0]] + listSum(a[1:], b[1:])
```

F.

```
def bits(n):
    if n < 2:
        return [n]
    else:
        return bits(n / 2) + [n % 2]
```

G. (This is not the only way to answer the question. In general, a good answer demonstrates that you know a couple of concepts. The `__init__` method need to initialize the object by setting all of the object's variables. These variables may be used subsequently in other methods of the object. The particular choice of variables is largely up to you, but there are some choices that are reasonable and others that are unreasonable. Notice that I use comments liberally, to explain my design choices.)

```
def __init__(self, text, action):
    # The button's width and height are determined by the size of the text.
    self.setText(text)
    # The action is a function that takes no arguments.
    # This function will be called whenever the button is pressed.
    self.action = action

# Assumes that all characters are rendered using 20x20 glyphs (pictures).
def setText(self, text):
    # The button should be 10 pixels longer than the text it holds, say.
    self.width = len(text) * 20 + 10
    # The button should be 10 pixels taller than the text it holds, say.
    self.height = 30;
    # When it's time to draw the button, this text string will be drawn.
    self.text = text
```