In [81]:
```
def basic_recursion(n):
    """What do I do?? Remember what we acted out during recitation?"""
    if n == 0:
        return 0

    return 1 + basic_recursion(n-1)
```

In [50]:
```
def recursive_fib(n):
    """Returns the nth number of the fibonacci sequence using vanilla recur
sion"""
    if n == 0:
        return 0
    elif n == 1:
        return 1

    return recursive_fib(n-2) + recursive_fib(n-1)

print("%d <-- that took way too long!" % recursive_fib(35))
```

```
9227465 <-- that took way too long!
```

In [56]:
```
fib_nums = [None]*1000 # This is an arbitrary length for this example. Do
n't do this in real life!
fib_nums[0] = 0
fib_nums[1] = 1

def memoized_fib(n):
    """Returns the nth number of the fibonacci sequence using top-down memo
ization"""
    if fib_nums[n] != None:
        return fib_nums[n]

    new_num = memoized_fib(n-2) + memoized_fib(n-1)
    fib_nums[n] = new_num
    return new_num

print("%d <-- that was way faster!" % memoized_fib(35))
```

```
9227465 <-- that was way faster!
```

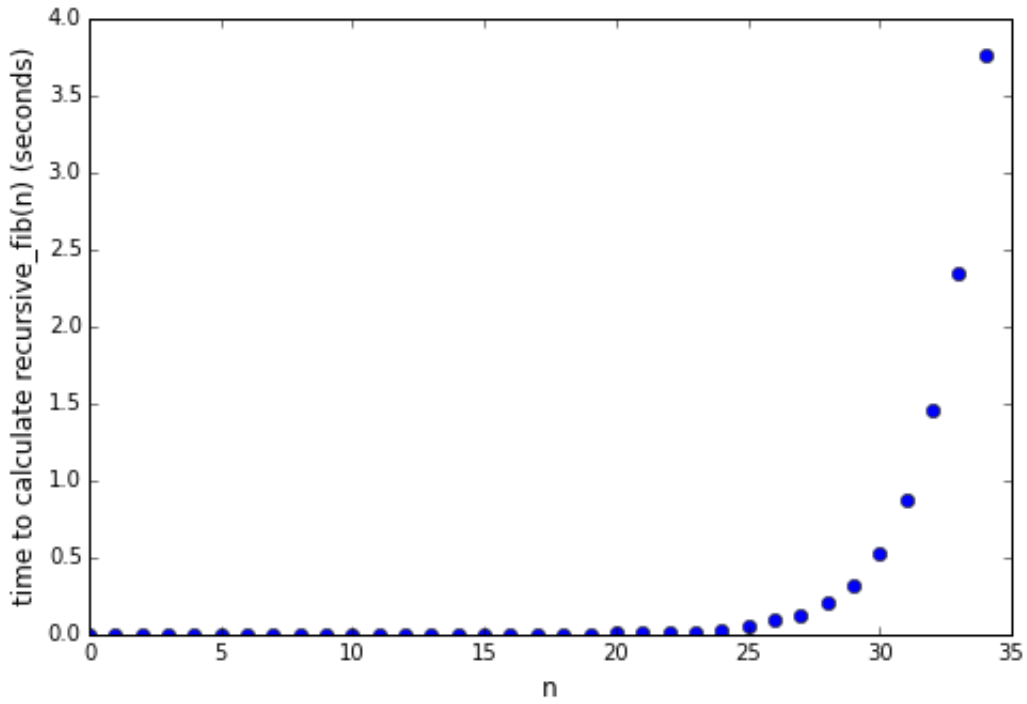
In [65]:
```
def dp_fib(n):
    """Returns the nth number of the fibonacci sequence using bottom-up dyn
amic programming"""
    fib_nums = [0,1]

    for i in range(n-1):
        fib_nums.append(fib_nums[-2] + fib_nums[-1])

    return fib_nums[n]

print("%d <-- and this is even better! (remember what I said about the cost
of function calls?)" % memoized_fib(35))
```

```
9227465 <-- and this is even better! (remember what I said about the cost o
f function calls?)
```

In [66]:
```
import time

def time_me(f, a):
    """I return the time it takes to run the function f with input a"""
    start = time.clock()
    f(a)
    end = time.clock()
    return end - start
```

In [67]:
```
%matplotlib inline
import matplotlib.pyplot as plt

recursive_inputs = list(range(35))
recursive_times = []

for n in recursive_inputs:
    recursive_times.append(time_me(recursive_fib, n))

fig = plt.figure()
axes = fig.add_axes([0, 0, 1, 1])
axes.set_xlabel('n', size="large")
axes.set_ylabel('time to calculate recursive_fib(n) (seconds)', size="larg
e")

axes.plot(recursive_inputs, recursive_times, 'o')

print("This is growing at roughly 2^n. Yikes.")
```

```
This is growing at roughly 2^n. Yikes.
```



In [80]:
```
dp_inputs = list(range(0, 3500, 100))
dp_times = []

for n in dp_inputs:
    dp_times.append(time_me(dp_fib, n))

fig = plt.figure()
axes = fig.add_axes([0, 0, 1, 1])
axes.set_xlabel('n', size="large")
axes.set_ylabel('time to calculate dp_fib(n) (seconds)', size="large")

axes.plot(dp_inputs, dp_times, 'o')

print("This is growing roughly linearly. Awesome!")
```

```
This is growing roughly linearly. Awesome!
```