# CS240

Nathan Sprague

March 1, 2013

# Warm-Up

```python
def fun(n):
    result = fun2(n-2)
    return result + 4

def fun2(n):
    result = 0
    for i in range(n):
        result = result + 1
    return result
```

How many addition operations when
fun(5) is executed?
fun(n)?

# Recursive Warm-Up

```
def fun3(n)
    if n == 0:
        return 20
    else:
        result = 0
        for i in range(4):
            result += 1
        return result + fun3(n - 1)
```

Let's develop an equation describing how many additions will be performed:

$T(0) = ??$

$T(3) = ??$

$T(n) = ??$

# Recurrences

We can express this as a recurrence :

```
def fun3(n)
    if n == 0:
         return 20
    else:
        result = 0
        for i in range(4):
            result += 1
        return result + fun3(n-1)
```

Let's develop an equation describing how many additions will be performed:

$T(0) = 0$

$T(n) = 5 + T(n-1)$

# Recurrence Exercise

Develop a Recurrence:

```
def fun3(n)
    if n == 0:
        return 20
    else:
        result = 0
        return result - fun3(n - 1) + fun3(n-1)
```

# STAGE 1 - Develop a Recurrence

- Develop a recurrence relation that describes the number of times the basic operation occurs in the worst (best, average) case:
- Typically:

**Initial Conditions**

$T(size\_of\_base\_case) = \#operations\_required\_for\_base\_case$

**Recurrence Relation**

$T(n) =$
$\#operations\_in\_call + \#recursive\_calls \times T(size\_of\_recursive\_calls)$

# STAGE 2 - Solve the Recurrence

- One approach is the method of back substitution:
  1. Expand the recurrence by repeated substitution until a pattern emerges.
  2. Characterize the pattern by expressing the recurrence in terms of $n$ and $i$, (where $i$ is the number of substitutions).
  3. Substitute for $i$ an expression that will remove the recursive term.
  4. Manipulate the result to achieve a closed-form solution.

# STAGE 3 Check Your Answer

- Make sure that the recurrence and the closed form solution agree for several values of $n$.

# Example: Recursive Binary Search, Stage 1

Worst-case recurrence:
$W(1) = 1$
$W(n) = 1 + W(n/2)$

# Example: Recursive Binary Search, Stage 2

Backwards substitution:

$W(1) = 1$

$W(n) = 1 + W(n/2)$

$W(n) = 1 + 1 + W((n/2)/2)$ (substitute)

$W(n) = 1 + 1 + W(n/4)$ (simplify)

$W(n) = 1 + 1 + 1 + W(n/8)$

...

$W(n) = i + W(\frac{n}{2^i})$ (generalize)

Solve for $i$ that results in initial condition:

$\frac{n}{2^i} = 1$

$n = 2^i$

$i = \log_2 n$

Substitute $\log_2 n$ for $i$: $W(n) = \log_2 n + 1$

# Example: Recursive Binary Search, Stage 3

Applying recurrence:
$W(1) = 1$
$W(2) = 1 + W(1) = 1 + 1 = 2$
$W(4) = 1 + W(2) = 1 + 2 = 3$

Applying solution:
$W(1) = \log_2 1 + 1 = 0 + 1 = 1$
$W(2) = \log_2 2 + 1 = 1 + 1 = 2$
$W(4) = \log_4 4 + 1 = 2 + 1 = 3$

# Another Example

```
def fun(items):
    if len(items) <= 1:
        return basicOperation()
    else:
        basicOperation()
        return fun(items[2:])
```

# STAGE 1

```python
def fun(items):
    if len(items) <= 1:
        return basicOperation()
    else:
        basicOperation()
        return fun(items[2:])
```

Initial Conditions:

$$T(0) = 1$$

$$T(1) = 1$$

Recursive part:

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

# STAGE 2

Substitution:

$T(n) = 1 + T(n - 2)$
$T(n) = 1 + 1 + T((n - 2) - 2)$
$T(n) = 1 + 1 + T(n - 4)$
$T(n) = 1 + 1 + 1 + T(n - 6))$
...
$T(n) = i + T(n - 2i)$

# STAGE 2 (Continued)

Recursive term disappears when $n - 2i = 0$ or $n - 2i = 1$ (The first will apply for even $n$, the second will apply for odd $n$.)
$n - 2i = 0$
$i = n/2$

Substitute $n/2$ for $i$:
$T(n) = n/2 + 1$
For even n.

Similarly, $T(n) = \frac{n-1}{2} + 1$ for odd $n$.

# Recurrence Exercise

```python
def fun(items):
    n = len(items)
    if n <= 1:
        return 3
    for i in range(4):
        mid = n // 2
        fun(items[:mid])
    sum = 0
    for i in range(n):
        for j in range(n):
            sum = items[i] + items[j]
    return sum
```

# Recurrence

$$T(1) = 0$$

$$T(n) = n^2 + 4T(n/2)$$

# Solving With Backward Substitution

$T(1) = 0$
$T(n) = n^2 + 4T(\frac{n}{2})$
$T(n) = n^2 + 4((\frac{n}{2})^2 + 4T((\frac{n}{2})/2))$ (substitute)
$T(n) = n^2 + n^2 + 16T(\frac{n}{4})$ (simplify)
$T(n) = n^2 + n^2 + 16((\frac{n}{4})^2 + 4T(n/8))$ (substitute)
$T(n) = n^2 + n^2 + n^2 + 64T(n/8)$ (simplify)
...
$T(n) = i \times n^2 + 4^i T(\frac{n}{2^i})$ (generalize)
Solve for $i$ that results in initial condition:
$\frac{n}{2^i} = 1$
$i = \log_2 n$
Substitute $\log_2 n$ for $i$: $T(n) = n^2 \log_2 n$

# Checking The Answer

Applying recurrence:
$T(1) = 0$
$T(2) = 2^2 + 4T(1) = 4 + 4(0) = 4$
$T(4) = 4^2 + 4T(2) = 16 + 4(4) = 32$

Applying solution:
$T(1) = 1^2 \times \log_2 1 = 1 \times 0 = 0$
$T(2) = 2^2 \times \log_2 2 = 4 \times 1 = 4$
$T(4) = 4^2 \times \log_2 4 = 16 \times 2 = 32$