Lecture 4: Analyzing Recursive Algorithms
General Plan for Analysis of Recursive algorithms

✧ Decide on parameter $n$ indicating input size
✧ Identify algorithm’s basic operation
✧ Determine worst, average, and best cases for input of size $n$
✧ Set up a recurrence relation, with initial condition, for the number of times the basic operation is executed
✧ Solve the recurrence, or at least ascertain the order of growth of the solution (see Levitin Appendix B)
Ex 2.4, Problem 1(a)

Use a piece of paper and do this now, individually.

- Solve this recurrence relation:

\[ x(n) = x(n - 1) + 5 \quad \text{for } n > 1 \]
\[ x(1) = 0 \]
Individual Problem (Q1):

Solve the recurrence

\[ x(n) = x(n - 1) + 5 \quad \text{for } n > 1 \]
\[ x(1) = 0 \]

What’s the answer?
Ex 2.4, Problem 1(c)

✧ Use a piece of paper and do this now, individually.

- Solve this recurrence relation:

\[
x(n) = x(n - 1) + n \quad \text{for } n > 0
\]
\[
x(0) = 0
\]
Ex 2.4, Problem 1(d)

✧ Use a piece of paper and do this now, individually.

- Solve this recurrence relation for $n = 2^k$:

\[
x(n) = x(n/2) + n \quad \text{for } n > 1
\]
\[
x(1) = 1
\]
Ex 2.4, Problem 1(d)

Use a piece of paper and do this now, individually.

- Solve this recurrence relation for $n = 2^k$:

\[
    x(n) = x(n/2) + n \quad \text{for } n > 1
\]

\[
    x(1) = 1
\]

Answer?

A. $x(n) = 2^{n+1}$
B. $x(n) = 2n - 1$
C. $x(n) = n(n+1)$
D. None of the above
What does that mean?

“Solving a Recurrence relation” means:

- find an explicit (non-recursive) formula that satisfies the relation and the initial condition.

- For example, for the relation
  \[ x(n) = 3x(n - 1) \text{ for } n > 1, \quad x(1) = 4 \]
  the solution is
  \[ x(n) = 4 \times 3^{n-1} \]

- Check:
What does that mean?

“Solving a Recurrence relation” means:

- find an explicit (non-recursive) formula that satisfies the relation and the initial condition.
- For example, for the relation
  \[ x(n) = 3x(n - 1) \quad \text{for } n > 1, \quad x(1) = 4 \]

the solution is
  \[ x(n) = 4 \times 3^{n-1} \]

- Check:
  \[ x(1) = 4 \times 3^0 = 4 \times 1 = 4 \]
  \[ x(n) = 3x(n - 1) \quad \text{definition of recurrence} \]
  \[ = 3 \times [4 \times 3^{(n-1)-1}] \quad \text{substitute solution} \]
  \[ = 4 \times 3^{n-1} = x(n) \]
2. Set up and solve a recurrence relation for the number of calls made by $F(n)$, the recursive algorithm for computing $n!$.

$$F(n) = \begin{cases} \text{if } n = 0 & \text{then return } 1 \\ \text{else return } F(n - 1) \times n \end{cases}$$
Ex 2.4, Problem 3

3. Consider the following recursive algorithm for computing the sum of the first \( n \) cubes: \( S(n) = 1^3 + 2^3 + \ldots + n^3 \).

**Algorithm** \( S(n) \)

//Input: A positive integer \( n \)
//Output: The sum of the first \( n \) cubes

**if** \( n = 1 \) **return** 1

**else** **return** \( S(n - 1) + n \times n \times n \)

a. Set up and solve a recurrence relation for the number of times the algorithm’s basic operation is executed.
Ex 2.4, Problem 3

3. Consider the following recursive algorithm for computing the sum of the first $n$ cubes: $S(n) = 1^3 + 2^3 + \ldots + n^3$.

Algorithm $S(n)$
//Input: A positive integer $n$
//Output: The sum of the first $n$ cubes
if $n = 1$ return 1
else return $S(n - 1) + n \times n \times n$

a. Set up and solve a recurrence relation for the number of times the algorithm’s basic operation is executed.

b. How does this algorithm compare with the straightforward nonrecursive algorithm for computing this function?

$S \leftarrow 1$
for $i \leftarrow 2$ to $n$ do
    $S \leftarrow S + i \times i \times i$
return $S$
Ex 2.4, Problem 4(a)

Consider the following recursive algorithm.

Algorithm $Q(n)$
//Input: A positive integer n
if $n = 1$ return 1
else return $Q(n - 1) + 2 \times n - 1$

a. Set up a recurrence relation for this function’s values and solve it to determine what this algorithm computes.
Ex 2.4, Problem 4(a)

Consider the following recursive algorithm.

Algorithm \( Q(n) \)

//Input: A positive integer \( n \)
if \( n = 1 \) return 1
else return \( Q(n - 1) + 2 \times n - 1 \)

a. Set up a recurrence relation for this function’s values and solve it to determine what this algorithm computes.

b. Set up a recurrence relation for the number of multiplications made by this algorithm and solve it.
Ex 2.4, Problem 4(a)

Consider the following recursive algorithm.

Algorithm \( Q(n) \)
//Input: A positive integer \( n \)
if \( n = 1 \) return 1
else return \( Q(n - 1) + 2 * n - 1 \)

a. Set up a recurrence relation for this function’s values and solve it to determine what this algorithm computes.

b. Set up a recurrence relation for the number of multiplications made by this algorithm and solve it.

c. Set up a recurrence relation for the number of additions/subtractions made by this algorithm and solve it.
Ex 2.4, Problem 8

a. Design a recursive algorithm for computing $2^n$ for any nonnegative integer $n$ that is based on the formula: $2^n = 2^{n-1} + 2^{n-1}$.

b. Set up a recurrence relation for the number of additions made by the algorithm and solve it.

c. Draw a tree of recursive calls for this algorithm and count the number of calls made by the algorithm.

d. Is it a good algorithm for solving this problem?
Ex 2.4, Problem 11

The determinant of an \( n \)-by-\( n \) matrix

\[
A = \begin{bmatrix}
a_{11} & a_{1n} \\
a_{21} & a_{2n} \\
a_{n1} & a_{nn}
\end{bmatrix},
\]

denoted \( \det A \), can be defined as \( a_{11} \) for \( n = 1 \) and, for \( n > 1 \), by the recursive formula

\[
\det A = \sum_{j=1}^{n} s_j a_{1j} \det A_j,
\]

where \( s_j \) is +1 if \( j \) is odd and -1 if \( j \) is even, \( a_{1j} \) is the element in row 1 and column \( j \), and \( A_j \) is the \( (n - 1) \)-by-\( (n - 1) \) matrix obtained from matrix \( A \) by deleting its row 1 and column \( j \).

a. Set up a recurrence relation for the number of multiplications made by the algorithm implementing this recursive definition. (Ignore multiplications by \( s_j \).)
Ex 2.4, Problem 11

The determinant of an $n$-by-$n$ matrix

$$A = \begin{bmatrix}
a_{11} & a_{1n} \\
a_{21} & a_{2n} \\
a_{n1} & a_{nn}
\end{bmatrix},$$

denoted $\det A$, can be defined as $a_{11}$ for $n = 1$ and, for $n > 1$, by the recursive formula

$$\det A = \sum_{j=1}^{n} s_j a_{1j} \det A_j,$$

where $s_j$ is +1 if $j$ is odd and -1 if $j$ is even, $a_{1j}$ is the element in row 1 and column $j$, and $A_j$ is the $(n - 1)$-by-$(n - 1)$ matrix obtained from matrix $A$ by deleting its row 1 and column $j$.

a. Set up a recurrence relation for the number of multiplications made by the algorithm implementing this recursive definition. (Ignore multiplications by $s_j$.)

b. Without solving the recurrence, what can you say about the solution’s order of growth as compared to $n!$?