Lecture 4
Analysis of Recursive Algorithms

Reading: Sections 2.4, 2.5 and Appendix B

Exercises:
2.4: 1, 3, 4, 7, 8, 9
2.5: 1, 2, 6, 9

Next time: Brute force algorithms (Chapter 3)

Example Recursive evaluation of $n!$

- **Definition:** $n! = 1*2*...*(n-1)*n$

- **Recursive definition of $n!$:**

- **Algorithm:**
  if $n=0$ then $F(n) := 1$
  else $F(n) := F(n-1) * n$
  return $F(n)$

- **Recurrence for number of multiplications to compute $n!$:**
Time efficiency of recursive algorithms

Steps in mathematical analysis of recursive algorithms:

1. Decide on parameter $n$ indicating *input size*
2. Identify algorithm’s *basic operation*
3. Determine *worst*, *average*, and *best* case for input of size $n$
4. Set up a recurrence relation and initial condition(s) for $C(n)$—the number of times the basic operation will be executed for an input of size $n$ (alternatively count recursive calls).
5. Solve the recurrence to obtain a closed form or estimate the order of magnitude of the solution (see Appendix B)

Important recurrence types:

1. One (constant) operation reduces problem size by one.
   \[ T(n) = T(n-1) + c \quad T(1) = d \]
   Solution: \[ T(n) = (n-1)c + d \quad \text{linear} \]

2. A pass through input reduces problem size by one.
   \[ T(n) = T(n/2) + cn \quad T(1) = d \]
   Solution: \[ T(n) = \left\lfloor \frac{n(n+1)}{2} \right\rfloor c + d \quad \text{quadratic} \]

3. One (constant) operation reduces problem size by half.
   \[ T(n) = T(n/2) + c \quad T(1) = d \]
   Solution: \[ T(n) = c \log n + d \quad \text{logarithmic} \]

4. A pass through input reduces problem size by half.
   \[ T(n) = 2T(n/2) + cn \quad T(1) = d \]
   Solution: \[ T(n) = cn \log n + d n \quad \text{n log n} \]
A general divide-and-conquer recurrence

\[ T(n) = aT(n/b) + f(n) \quad \text{where } f(n) \in \Theta(n^k) \]

1. \( a < b^k \) \( T(n) \in \Theta(n^k) \)
2. \( a = b^k \) \( T(n) \in \Theta(n^k \log n) \)
3. \( a > b^k \) \( T(n) \in \Theta(n^{\log_b a}) \)

Note: the same results hold with \( O \) instead of \( \Theta \).

Fibonacci numbers

- The Fibonacci sequence:
  
  0, 1, 1, 2, 3, 5, 8, 13, 21, ...

- Fibonacci recurrence:
  
  \[ F(n) = F(n-1) + F(n-2) \]
  \[ F(0) = 0 \]
  \[ F(1) = 1 \]

- Another example:
  
  \[ \Lambda(n) = 3\Lambda(n-1) - 2\Lambda(n-2) \]
  \[ \Lambda(0) = 1 \quad \Lambda(1) = 3 \]
Solving linear homogeneous recurrence relations with constant coefficients

- Easy first: 1st order LHRRCCs:
  \[ C(n) = a \cdot C(n-1) \quad C(0) = t \quad \text{... Solution: } C(n) = t \cdot a^n \]

- Extrapolate to 2nd order:
  \[ L(n) = a \cdot L(n-1) + b \cdot L(n-2) \quad \text{... A solution?: } L(n) = r^n \]

- Characteristic equation (quadratic)
  \[ \text{Solve to obtain roots } r_1 \text{ and } r_2 \text{—e.g.: } A(n) = 3A(n-1) - 2A(n-2) \]

- General solution to RR: linear combination of \( r_1^n \) and \( r_2^n \)

- Particular solution: use initial conditions—e.g.: \( A(0) = 1 \quad A(1) = 3 \)

Computing Fibonacci numbers

1. Definition based recursive algorithm
2. Nonrecursive brute-force algorithm
3. Explicit formula algorithm
4. Logarithmic algorithm based on formula:
   \[
   \begin{pmatrix}
   F(n-1) \\ F(n)
   \end{pmatrix}
   =
   \begin{pmatrix}
   0 & 1 \\ 1 & 1
   \end{pmatrix}
   \begin{pmatrix}
   F(n) \\ F(n+1)
   \end{pmatrix}
   ^n
   \]

   * for \( n \geq 1 \), assuming an efficient way of computing matrix powers.