Recursive algorithms -> solve large problem by applying themselves to a part of the problem
   then use solution to solve larger problem
   “part of the problem” may only have 1 less item than full problem
   called reduce and conquer algos
   “part of the problem” may be half the elements in full problem
   called divide and conquer
   recursive algorithms are simple to understand, small, and straightforward

   saw in first chapter -> analyzing iterative algorithm = determine
   how much work is done within each loop
   how many times loop executes

   analyzing a recursive algo entails determining:
   amount of work done to produce smaller pieces
   amount of work to put solutions together to get larger solution
   combine this info w/ # of smaller pieces and sizes => produce recurrence relation
   recurrence can be converted into closed form to compare to others

   start with describing process for creating recurrence relation from recursive algo
   then look at how to solve or approx recur relation to closed, non-recursive form

   Recurrence Relations
   can be directly derived from a recursive algo,
   but form does not let us identify the efficiency
   must convert RR to closed form by removing recursion
   done by substitutions until we see pattern develop
   examples...

   EXAMPLES:
   sequence 5,8,11,14,17, ...
   A0 = 5
   An = An-1 + 3
   = (An-2 + 3) + 3
   = (An-3 + 3) + 3 + 3
   = ...
   = A0 + 3n
   = 5 + 3n for n >= 0
   sequence 3, 6, 12, 24, 48...
   A0 = 3
   An = 2*An-1
   = 2 ( 2 * An-2) = 4*An-2
   = 4 ( 2 * An-3) = 8*An-3
   = 8 ( 2 * An-4) = 16*An-4
   = ...
   = 2^n * A0
   = 2^n * 3 for n >= 0

   Analyzing recursive algorithms
   recur algos -> small and powerful way to solve problem
   how to analyze one?
   when we count comparisons in loops, only need to determine
   how many comparisons are done inside the loop
   how many times loop executes
   recursive algos- not clear how many times a task will be done
   algo does not determine how deeply it will recur

   factorial (recursive)

   factorial(N)
   if n==1
      return 1
   else
      n = N - 1
      ans = factorial( n )
      return N * ans

   direct solution (base case)
   return 1
   divide input in general solution
   one subtraction
   recur
   combine

   analysis of recursive algorithms is straightforward if you can map it into 4 generic steps:
1) direct solution
2) division of input
3) number of recursive calls
4) combination of solutions
   - can use formula to compute complexity of divide and conquer algos.

\[ \text{REC}(N) = \left\{ \begin{array}{ll}
\text{DIR}(N) & \text{for } N = 1 \\
\text{DIV}(N) + \sum_{i=1}^{\text{numberSmaller}} \text{REC}({\text{smallerSizes}[i]}) + \text{COM}(N) & \text{for } N > 1
\end{array} \right. \]

- REC(N) - complexity of recursive algorithm
- DIR(N) complexity of direction solution
- DIV(N) - complexity of divide
- COM(N) - complexity of combine

- factorial example:
  - direction solution (return 1) is no calculations, so DIR(N) = 0, for N = 1
  - divide = 1 subtraction DIV(N) = 1
  - combine = 1 multiplication COM(N) = 1
  - recursive calls are 1 smaller than original
    - RR for factorial’s number of calculations:

\[ \text{calc}(N) = \left\{ \begin{array}{ll}
0 & \text{for } N = 1 \\
1 + \text{calc}(N - 1) + 1 & \text{for } N > 1
\end{array} \right. \]

- complexity of recursive factorial?
  - T(1) = 0
  - T(n) = T(n-1) + 2
    - based on previous formula
  - substitution method:
    - T(n) = T(n-1) + 2
    - replace n with n+1:
      - T(n) = T((n+1) - 1) + 2
    - substitute n+1 is original equation:
      - T(n) = T(n-2) + 2
      - replace n with n+2:
        - T(n+2) = T((n+2) - 1) + 2
        - T(n+2) = T(n) + 2
        - sub n+2 in previous equation:
          - T(n) = T(n+2) + 2
          - = T(n+3) + 2 + 2
          - = T(n+3) + 4
          - look for pattern:
            - T(n) = T(n-k) + 2k
        - use base case to solve for k:
          - T(1) = 0
          - n - k = 1
          - k = n - 1
          - T(n) = T(n - (n - 1)) + 2(n-1)
          - = T(1) + 2n - 2
          - = O(n)

- 2.1.1 exercises
  - #1 in class: Fibonacci
    - how many RR for # of additions?
      - which is DIRECT?
      - Division of input?
      - Combination?

1. The direct solution is the return of the value 1 and does no additions. The division of the input is the two subtractions in the calculation of N – 1 and N – 2. The combination of the solutions is the addition in the return statement. This gives the recurrence relation:

\[ A(N) = \left\{ \begin{array}{ll}
0 & \text{if } N = 0 \text{ or } N = 2 \\
2 + A(N-1) + A(N-2) & \text{if } N > 2
\end{array} \right. \]

- #2 in class
  - power(x,y) = x * power(x-1,y-1)
write recursive version
give RR for recursive function

2.

```
Power(x, y)
if y == 0 then
    return 1
else
    return x * Power(x, y-1)
end if
```

\[
M(N) = \begin{cases} 
0 & \text{if } N = 0 \\
1 + M(N-1) & \text{if } N > 0 
\end{cases}
\]

- complexity?
- \( T(n) = T(n-1) + 1 \)
- \( = O(n) \) substitute, replace

- Approximating the order of RRs
  - when considering divide and conquer algs
    - problem of size \( N \)
    - divided into \( a \) parts of size \( N / b \)
    - plus work to divide and combine (e.g., \( f(N) \) is some function)
    - \( T(N) = a \cdot T(N / b) + f(N) \)
  - e.g., divide into four parts

- Master Method
  
  \[
  T(N) = \begin{cases} 
\Theta(N^d) & \text{if } a < b^d \\
\Theta(N^d \log N) & \text{if } a = b^d \\
\Theta(N^{\log_b a}) & \text{if } a > b^d 
\end{cases}
\]

  - \( a \) is number of recursive calls
  - \( b \) is factor of division
  - \( d \) is amount of work done outside of recursion.

- exercises 2.2.2
  - #5

  5. \( a = 5, b = 4, \text{ and } d = 1 \). So, \( a > b^d \) and \( T(N) = \Theta(N^{1.16}) \)

- Closest pair 1D
  - draw line of points
    - 0,5,10,11,15,29
  - brute force. complexity?
    - \( 2 \) for loops

  - divide and conquer
    - sort.
      - \( \log n \)
    - divide?
      - cut in half
    - conquer?
    - find closest pair in \( L, R \)
    - combine?
      - find closest b/t \( L, R \)
      - check in between