Divide-and-Conquer algorithms:

Examples:  Merge sort
            Quick sort
            Binary search

Format of divide-and-conquer algorithms:

- Divide: Split the array or list into smaller pieces
- Conquer: Solve the same problem recursively on smaller pieces
- Combine: Build the full solution from the recursive results
Merge sort:

- **Divide**: Split array of size $n$ into two subarrays of size $n/2$
- **Conquer**: Two recursive calls on subarrays of size $n/2$
- **Combine**: Merge two sorted subarrays to create sorted array of size $n$

Let $T(n)$ = running time of merge sort on array of size $n$

$$T(n) = T_{\text{divide}}(n) + T_{\text{conquer}}(n) + T_{\text{combine}}(n)$$

$$T(n) = \theta(1) + [T(n/2) + T(n/2)] + \theta(n)$$

$$T(n) = 2 \ T(n/2) + \theta(n)$$

- Number of recursive subproblems = 2
- Size of each subproblem = $n/2$
- Time for all the non-recursive steps = $\theta(n)$

Later we’ll see how to solve these kinds of recurrence equations to determine $T(n)$

The solution for $T(n) = 2 \ T(n/2) + \theta(n)$ is $T(n) = \theta(n \ lg \ n)$
Polynomial multiplication and Large integer multiplication:

Given two polynomials:

\[ P = 5x^3 + 7x^2 + 6x + 2 \implies [5, 7, 6, 2] = [p_3, p_2, p_1, p_0] \]
\[ Q = x^3 - 8x^2 + 9x - 1 \implies [1, -8, 9, -1] = [q_3, q_2, q_1, q_0] \]

\[ P \times Q = (5x^3 + 7x^2 + 6x + 2)(x^3 - 8x^2 + 9x - 1) \]

Alternatively, given two large integers:

\[ P = 9863 = 9x^3 + 8x^2 + 6x + 3 \text{ where } x = 10 \]
\[ Q = 7245 = 7x^3 + 2x^2 + 4x + 5 \text{ where } x = 10 \]

\[ P \times Q = (9863)(7245) = (9x^3 + 8x^2 + 6x + 3)(7x^3 + 2x^2 + 4x + 5) \text{ where } x = 10 \]

So large integer multiplication is a special case of polynomial multiplication, where \( x \) = the base in which the numbers are represented.
Divide-and-conquer Algorithm 1 for Polynomial multiplication:

\[ P = 5x^3 + 7x^2 + 6x + 2 = (5x + 7)x^2 + (6x + 2) = Ax^2 + B \]
\[ Q = x^3 - 8x^2 + 9x - 1 = (x - 8)x^2 + (9x - 1) = Cx^2 + D \]

\[ A = 5x + 7 \Rightarrow [5, 7] \]
\[ B = 6x + 2 \Rightarrow [6, 2] \]
\[ C = x - 8 \Rightarrow [1, -8] \]
\[ D = 9x - 1 \Rightarrow [9, -1] \]

Let \( n \) = number of terms in \( P \) and \( Q \)

\[ P = Ax^{n/2} + B \]
\[ Q = Cx^{n/2} + D \]

\( A, B, C, D \) are polynomials with \( n/2 \) terms

\[ P \ast Q = (Ax^{n/2} + B)(Cx^{n/2} + D) = (AC)x^n + (BC + AD)x^{n/2} + (BD) \]

Four recursive subproblems:

\[ A \ast C \]
\[ B \ast C \]
\[ A \ast D \]
\[ B \ast D \]

Stop the recursion when \( n = 1 \)
Let $T(n) =$ running time for the preceding Algorithm 1

\[ T(n) = 4 \ T(n/2) + \theta(n) \]

- Number of recursive subproblems = 4
- Size of each subproblem = $n/2$
- Time for all the non-recursive steps = $\theta(n)$:
  - adding $n$-term polynomials
  - $x^n$ and $x^{n/2}$ which are just shifts (not products)
  - also note the final product $P \times Q$ has $< 2n$ terms

Again, later we’ll see how to solve these kinds of recurrence equations to determine $T(n)$

The solution for $T(n) = 4 \ T(n/2) + \theta(n)$ is $T(n) = \theta(n^2)$
Divide-and-conquer Algorithm 2 for Polynomial multiplication (Karatsuba’s algorithm):

As before,
\[ P \times Q = (Ax^{n/2} + B)(Cx^{n/2} + D) = (AC)x^n + (BC + AD)x^{n/2} + (BD) \]

This time we write \( (BC + AD) = (A+B)(C+D) - (AC) - (BD) \)

So only three recursive calls:
- \( A \times C \)
- \( B \times D \)
- \( (A+B) \times (C+D) \)

Again stop the recursion when \( n = 1 \)

Let \( T(n) = \) running time for the preceding Algorithm 2

\[ T(n) = 3 \, T(n/2) + \theta(n) \]

- Number of recursive subproblems = 3
- Size of each subproblem = \( n/2 \)
- Time for all the non-recursive steps = \( \theta(n) \)

Again, later we will see how to solve these kinds of recurrence equations to determine \( T(n) \)

The solution for \( T(n) = 3 \, T(n/2) + \theta(n) \) is \( T(n) = \theta(n^{\lg 3}) \approx \theta(n^{1.58}) \)
Majority element problem:

Given array A of size n, does there exist any value M that appears more than n/2 times in array A?

Majority element algorithm:
- Phase 1: Use divide-and-conquer to find candidate value M
- Phase 2: Check if M really is a majority element, $\theta(n)$ time, simple loop

Phase 1 details:

- **Divide:**
  - Group the elements of A into n/2 pairs
  - If n is odd, there is one unpaired element, x
    - Check if this x is majority element of A
      - If so, then return x, but otherwise discard x
  - Compare each pair (y, z)
    - If (y==z) keep y and discard z
    - If (y!=z) discard both y and z
  - So we keep $\leq n/2$ elements
- **Conquer:** One recursive call on subarray of size $\leq n/2$
- **Combine:** Nothing remains to be done, so omit this step
Example:
A = [7, 7, 5, 2, 5, 5, 4, 5, 5, 5, 7]
    (7, 7) (5, 2) (5, 5) (4, 5) (5, 5) (7)
A = [7, 5, 5]
    (7, 5) (5) \Rightarrow \text{return 5 (candidate, also majority)}

Example:
A = [1, 2, 3, 1, 2, 3, 1, 2, 9, 9]
    (1, 2) (3, 1) (2, 3) (1, 2) (9, 9)
A = [9] \Rightarrow \text{return 9 (candidate, but not majority)}

Let $T(n) = \text{running time of Phase 1 on array of size } n$

$T(n) = T(n/2) + \theta(n)$

- Number of recursive subproblems = 1
- Size of each subproblem = $n/2$  [worst-case]
- Time for all the non-recursive steps = $\theta(n)$

Later we’ll see how to solve these kinds of recurrence equations to determine $T(n)$

The solution for $T(n) = T(n/2) + \theta(n)$ is $T(n) = \theta(n)$