Divide-and-Conquer
Divide-and-Conquer technique

- **Divide**: Divide (break) the problem (instance) into a number of subproblems that are smaller instances of the same problem.
- **Conquer**: Solve the subproblems recursively. If the subproblem sizes are small enough just solve them in a straightforward manner.
- **Combine**: Combine the subproblem solutions into the solution for the original problem.
Divide and Conquer technique: Proving correctness

- Prove base cases are correct
- Prove the correctness of the part combining the result
- Proof by induction the whole algorithm is correct
  - Assume that the subproblems are solved correctly and use that to prove the correctness of the problem
Divide and Conquer technique: Runtime

- Develop a recurrence relation representing the time complexity of the algorithm
- Solve the relation using one of the following method
  - guess a solution (using substitution method) and prove its correctness by induction
  - Use recurrence tree to find a solution and then prove its correctness using induction
  - Master theorem
Example: Mergesort

- Divide: Split the array into two half
- Conquer: Sort each half using mergesort
- Combine: Merge the two half

**Base case:** An empty or single-element array is sorted
Merging two arrays

def merge(A, B):
    result = []
    i = 0
    while i < len(A) and j < len(B):
        if A[i] <= A[j]:
            result.append(A[i])
            i += 1
        else:
            result.append(B[j])
            j += 1

    while i < len(A):
        result.append(A[i])
        i += 1

    while j < len(B):
        result.append(B[j])
        j += 1

    return result
Mergesort algorithm

def mergesort(A):
    if len(A) <= 1:
        return A
    //Divide the list into two halves L and R
    mid = (ceil(n/2))
    L = A[1 … mid]
    R = A[mid+1 … n]
    mergesort(L)
    mergesort(R)
    return merge(L, R)

Runtime:

\[
T(1) = \Theta(1)
\]
\[
T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + \Theta(n)
\]
Question

Why not split the array into fourth? Or eights?
Runtime of recursive algorithms

- **Recurrence relation:**
  a. describes the runtime of a problem of size $n$ in terms of the runtime on smaller inputs.
  b. Used in algorithms that use recursion like divide-and-conquer technique.

- **General approach for solving a recurrence**
  a. finding an explicit expression
  b. Finding an asymptotic bound on its growth rate

- **Techniques to solve recurrence relation:**
  a. Guessing method
     i. How to guess: Substitution method (educated guess)
  b. Recurrence tree method

- Both of the above techniques solve a recurrence intuitively. Then we can use induction to prove their correctness formally.
Solving Recurrence Relation for MergeSort

\[ T(n) = \begin{cases} 
\Theta(1) & n = 1 \\
T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + \Theta(n) & n > 1 
\end{cases} \]

\[ T(n) \leq \begin{cases} 
c & n = 1 \\
2T(\frac{n}{2}) + cn & n > 1 
\end{cases} \]

1) We assume \( n \) is a power of 2 → floor and ceiling are not needed
2) \( \Theta(1) \) is a constant and \( \Theta(n) \) is multiple of a constant.
   a) We use the same constant
Technique 1: Substitution method

Finding a guess assuming $n$ is a power of 2

$n = 2^k$

$n / 2^k = 1$

$\log_2 n = k$

$T(n) \leq 2T\left(\frac{n}{2}\right) + cn$

$= 2 \left(2T\left(\frac{n}{4}\right) + \left(\frac{cn}{2}\right)\right) + cn$

$= 4T\left(\frac{n}{4}\right) + cn + cn$

$= 4T\left(\frac{n}{4}\right) + 2cn$

$= 4 \left(2T\left(\frac{n}{8}\right) + \left(\frac{cn}{4}\right)\right) + 2cn$

$= 8T\left(\frac{n}{8}\right) + cn + 2cn$

$= 8T\left(\frac{n}{8}\right) + 3cn$

\[ \vdots \]

$= 2^kT\left(\frac{n}{2^k}\right) + kcn$

$= 2^{\log_2 n}T(1) + cn\log_2 n$

$= nT(1) + cn\log_2 n$

$= cn + cn\log_2 n$

$= O(n\log n)$
Technique 1: Substitution method: proving using induction

Proving the correctness of guess using induction. In other words, prove

\[ T(n) \leq cn + cn \log n \]

**Proof:** base case, if \( n = 1 \), then \( T(n) = T(1) \leq cn \log_2 n + cn = c \)

**Inductive step:**

- **hypothesis:** assume the claim holds for all \( m < n \) that are powers of two
- **Induction:**

\[
\begin{align*}
T(n) & \leq 2T \left( \frac{n}{2} \right) + cn \\
& \leq 2 \left( \left( \frac{cn}{2} \right) \log \frac{n}{2} + \left( \frac{cn}{2} \right) \right) + cn \\
& = cn \log \frac{n}{2} + cn + cn \\
& = cn \log n - cn + cn + cn \\
& \leq cn \log n + cn
\end{align*}
\]
What if \( n \) is not a power of 2?

The previous inequality is true if \( n \) is a power of two.

Since the function \( T(n) \) is increasing, then
\[
T(n) \leq T(n')
\]
where \( n' \) is the first power of two larger than \( n \):
\[
n \leq n' < 2n
\]

So, if we prove a bound for \( T(n') \) we can use that to get a bound for \( T(n) \)

\[
T(n) \leq \begin{align*}
& cn' \log n' + cn' \\
& (2n)c \log(2n) + 2nc \\
& (2n)c \log(2n) + 2nc \\
& 2nc \log n + 2nc + 2nc \\
& 2nc \log n + 4nc \\
& 4nc \log n + 4nc \\
& nc' \log n + nc' \\
& \Theta(n \log n)
\end{align*}
\]