Examples of 
Divide and Conquer 
and the Master theorem

Divide and Conquer

Reduce to any number of smaller instances:
1. Divide the given problem instance into subproblems
2. Conquer the subproblems by solving them recursively
3. Combine the solutions for the subproblems to a solution for the original problem
Search Problem

- **Input:** A set of numbers $a_1, a_2, \ldots, a_n$ and a number $x$
- **Question:** Is $x$ one of the numbers $a_i$ in the given set?

If sorted array $A$ of numbers $a_1, a_2, \ldots, a_n$ then **Binary Search**

Binary Search

- Compare $x$ to the middle element of the array $A[\lceil n/2 \rceil]$

  \[
  \begin{array}{c}
  \text{if } x = A[\lceil n/2 \rceil] \text{ then done} \\
  \text{if } x < A[\lceil n/2 \rceil] \text{ then} \\
  \quad \text{recursively Search } A[1, \ldots, \lceil n/2 \rceil - 1] \\
  \text{if } x > A[\lceil n/2 \rceil] \text{ then} \\
  \quad \text{recursively Search } A[\lceil n/2 \rceil + 1, \ldots, n]
  \end{array}
  \]
Binary Search Analysis

\[ T(n) = \begin{cases} 
T(n/2) + \Theta(1) & \text{for } n > 1 \\
\Theta(1) & \text{for } n = 1 
\end{cases} \]

\[ a = 1, \ b = 2, \ f(n) = \Theta(1), \ n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \]

Case 2 \( \Rightarrow T(n) = \Theta(\log n) \)

---

Merge Sort

1. **Divide:** Divide the given n-element sequence to be sorted into two sequences of length n/2
2. **Conquer:** Sort recursively the two subsequences using Merge Sort
3. **Combine:** Merge the two sorted subsequences to produce the sorted answer
Merge

• **Input:** Sorted arrays $K[1..n_1]$, $L[1..n_2]$
• **Output:** Merged sorted array $M[1..n_1+n_2]$

\[
i = 1, \quad j = 1
\]
\[
\text{for } t = 1 \text{ to } n_1 + n_2
\]
\[
\begin{cases}
\text{if } (i \leq n_1 \text{ and } (j > n_2 \text{ or } K[i] < L[j])) \\
\quad \text{then } M[t] = K[i], \quad i = i + 1
\end{cases}
\]
\[
\begin{cases}
\text{else } M[t] = L[j], \quad j = j + 1
\end{cases}
\]

Linear Time Complexity: $\Theta(n_1 + n_2)$

What if inputs, output in same array?

• **Input:** Sorted array segments $A[1..n_1]$, $A[n_1+1..n_1+n_2]$
• **Output:** Merged sorted array $A[1..n_1+n_2]$

Copy $A[1..n_1]$ into new array $K[1..n_1]$
Copy $A[n_1+1…n_1+n_2]$ into $L[1..n_2]$
Merge $K[1..n_1]$ and $L[1..n_2]$ into $A[1..n_1+n_2]$

Linear Time Complexity: $\Theta(n_1 + n_2)$
Merge-Sort

Merge-Sort \(A[1 \ldots n]\)

If \(n > 1\) then
1. Recursively merge-sort \(A[1 \ldots \lfloor n/2 \rfloor]\) and \(A[\lceil n/2 \rceil + 1 \ldots n]\)
2. Merge the two sorted subsequences

Analysis of Merge-Sort

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

Assume for simplicity that \(n\) is a power of 2

\[T(n) = 2T(n/2) + cn\]

\(a = 2, \ b = 2, \ f(n) = \Theta(n), \ n^{\log_a b} = n^{\log_2 2} = n\)

Case 2 \(\Rightarrow T(n) = \Theta(n \log n)\)
Maximum Sum Subarray Problem

- **Input:** Array $A[1...n]$ of integers (positive and negative)
- **Problem:** Compute a subarray $A[i^*...j^*]$ with maximum sum
  i.e., if $s(i,j)$ denotes the sum of the elements of a subarray $A[i...j]$,
  $$s(i, j) = \sum_{k=i}^{j} A[k]$$

  We want to compute indices $i^*\leq j^*$ such that
  $$s(i^*, j^*) = \max \{s(i, j) \mid 1 \leq i \leq j \leq n\}$$

  **Example:** 3 -4 5 -2 -2 6 -3 5 -3 2

  max sum = 9

Brute force solution

- Compute the sum of every subarray and pick the maximum
  Try every pair of indices $i,j$ with $1 \leq i \leq n$ , and for each one compute
  $$s(i, j) = \sum_{k=i}^{j} A[k]$$

- Time complexity $\Theta(n^3)$

- With a little more care, can improve to $\Theta(n^2)$:
  can compute the sums of all the subarrays in time $\Theta(n^2)$. 
Brute force solution - improved

• With a little more care, can improve to $\Theta(n^2)$:
• Can compute the sums for all subarrays with same left end in $O(n)$ time $\Rightarrow$ compute the sums of all the subarrays (there are $n(n-1)/2 + n$ subarrays) in time $O(n^2)$

for $i = 1$ to $n$
  \{  
  $s(i, i) = A[i]$
  for $j = i + 1$ to $n$
    $s(i, j) = s(i, j-1) + A[i, j]$
  
  $s(i, j) = s(i, j-1) + A[i, j]$
  \}

Divide and Conquer

• A subarray $A[i^* \ldots j^*]$ with maximum sum is
  – Either contained entirely in the first half, i.e. $j^* \leq n/2$
  – Or contained entirely in the right half, i.e. $i^* \geq n/2$
  – Or overlaps both halves: $i^* \leq n/2 \leq j^*$

• We can compute the best subarray of the first two types with recursive calls on the left and right half.
• The best subarray of the third type consists of the best subarray that ends at $n/2$ and the best subarray that starts at $n/2$. We can compute these in $O(n)$ time.
Divide and Conquer analysis

- **Recurrence**: \( T(n) = 2T(n/2) + \Theta(n) \)
- **Solution**: \( T(n) = \Theta(n \log n) \)

- It is possible to do better: can compute the maximum sum subarray in \( \Theta(n) \) time.

*HW Exercise. Not divide and conquer*

For a nice paper on this problem see
J. Bentley, Programming Pearls, Addison-Wesley,
chapter 8 (Algorithm Design Techniques)
Also in Communications of the ACM, 27(9), 1984.

Multiplication of Big integers

- Given integers \( A, B \) with \( n \) bits each, can + , - in \( O(n) \) time.
- Ordinary multiplication: \( n^2 \) time (\( n \) additions)

- D&C: partition into high \( n/2 \) and low \( n/2 \) bits

\[
\begin{array}{c|c|c}
A & A_h & A_l \\
\hline
B & B_h & B_l \\
\end{array}
\]

\[
A = A_h \cdot 2^{n/2} + A_l
\]

\[
B = B_h \cdot 2^{n/2} + B_l
\]

\[
A \cdot B = (A_h \cdot 2^{n/2} + A_l) \cdot (B_h \cdot 2^{n/2} + B_l)
\]

\[
= A_hB_h \cdot 2^n + A_hB_l \cdot 2^{n/2} + A_lB_h 2^{n/2} + A_lB_l
\]
Multiplication of Big integers

\[ A \cdot B = (A_h \cdot 2^{n/2} + A_l) \cdot (B_h \cdot 2^{n/2} + B_l) \]
\[ = A_hB_h \cdot 2^n + A_hB_l \cdot 2^{n/2} + A_lB_h \cdot 2^{n/2} + A_lB_l \]

4 multiplications of \( n/2 \)-bit numbers: \( A_hB_h, A_lB_l, A_lB_h, A_lB_l \),
additions and shifts.

Note: multiplications by powers of 2 are just shifts

Recurrence: \( T(n) = 4T(n/2) + cn \)
(last term for additions and shifts)

Solution: \( T(n) = O(n^2) \)

---

Multiplication of Big integers – Karatsuba’60

\[ A \cdot B = (A_h \cdot 2^{n/2} + A_l) \cdot (B_h \cdot 2^{n/2} + B_l) \]
\[ = A_hB_h \cdot 2^n + A_hB_l \cdot 2^{n/2} + A_lB_h \cdot 2^{n/2} + A_lB_l \]

\[ (A_h + A_l)(B_l + B_h) = A_hB_l + A_hB_h + A_lB_l + A_lB_h \Rightarrow \]
\[ A \cdot B = A_hB_h \cdot 2^n + [(A_h + A_l)(B_h + B_l) - A_hB_h - A_lB_l] \cdot 2^{n/2} + A_lB_l \]

3 multiplications of \( n/2 \)-bit numbers:
\( A_hB_h, A_lB_l, (A_h+A_l)(B_h+B_l) \)
+ additions, subtractions and shifts.

Recurrence: \( T(n) = 3T(n/2) + cn \)
Solution: \( T(n) = n^{\log_2 3} = n^{1.585} \)
Multiplication of Big integers – Karatsuba’60

Recursive Algorithm MULT(A,B)
Write A = A_h 2^{n/2} + A_l and B = B_h 2^{n/2} + B_l
Compute a = A_h + A_l and b = B_h + B_l
C = MULT(a,b)
D_h = MULT(A_h, B_h)
D_l = MULT(A_l, B_l)
Return D_h \cdot 2^n + [C – D_h – D_l] \cdot 2^{n/2} + D_l

Time: T(n) = n^{\log_2 3} = n^{1.585}

FFT-based method: n \log n \log \log n

Matrix Multiplication

Input: Matrices A = [a_{ij}], B = [b_{ij}], i, j = 1,...,n
Output: C = [c_{ij}] = A \cdot B

\[ c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \]
Standard Matrix Multiplication algorithm

\[
\begin{align*}
\text{for } i = 1 & \text{ to } n \\
\quad \text{for } j = 1 & \text{ to } n \\
\quad \quad \{ & \quad c_{ij} = 0 \\
\quad \quad \quad \text{for } k = 1 & \text{ to } n \\
\quad \quad \quad \quad c_{ij} & = c_{ij} + a_{ik} b_{kj} \\
\quad \} \\
\end{align*}
\]

Time Complexity: $\Theta(n^3)$

---

Divide and Conquer

Partition matrices $A, B, C$ into $4 \frac{n}{2} \times \frac{n}{2}$ submatrices

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix} =
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

$C_{11} = A_{11}B_{11} + A_{12}B_{21}$

$C_{12} = A_{11}B_{12} + A_{12}B_{22}$

$C_{21} = A_{21}B_{11} + A_{22}B_{21}$

$C_{22} = A_{21}B_{12} + A_{22}B_{22}$

8 recursive multiplications of $\frac{n}{2} \times \frac{n}{2}$ matrices

4 additions (direct – no recursion)
\[ T(n) = 8T(n/2) + \Theta(n^2) \]

\[ a = 8, \ b = 2, \ f(n) = \Theta(n^2), \ n^{\log_b a} = n^3 \]

Case 1 \[ \Rightarrow T(n) = \Theta(n^3) \]

Same as standard MM algorithm

---

**Strassen’s algorithm**

- \[ P = (A_{11} + A_{22})(B_{11} + B_{22}) \]
- \[ Q = (A_{21} + A_{22})B_{11} \]
- \[ R = A_{11}(B_{12} - B_{22}) \]
- \[ S = A_{22}(B_{21} - B_{11}) \]
- \[ T = (A_{11} + A_{12})B_{22} \]
- \[ U = (A_{21} - A_{11})(B_{11} + B_{12}) \]
- \[ V = (A_{12} - A_{22})(B_{21} + B_{22}) \]

- \[ C_{11} = P + S - T + V \]
- \[ C_{12} = R + T \]
- \[ C_{21} = Q + S \]
- \[ C_{22} = P + R - Q + U \]
Strassen’s algorithm

- Can multiply 2x2 matrices with 7 multiplications, and 18 additions and subtractions. The method does not assume commutativity of multiplication.
- Method applies to multiplication of 2x2 block matrices.
- Can be used in divide and conquer scheme with 7 recursive multiplications of $\frac{n}{2} \times \frac{n}{2}$ submatrices.

\[ T(n) = 7 \ T(n / 2) + \Theta(n^2) \]

Strassen’s Algorithm

\[ T(n) = 7 \ (n / 2) + \Theta(n^2) \]

\[ a = 7, \ b = 2, \ f(n) = \Theta(n^2), \ n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \]

Case 1  \[ \Rightarrow \ T(n) = \Theta(n^{\log_2 7}) \]

Best current (theoretical) result: \[ \Theta(n^{2.373}) \]