1 DIVIDE-AND-CONQUER [Chapter 5]

When the output corresponding to input of size \( n \) can be obtained from the outputs corresponding to inputs of size less than \( n \) then we can design an algorithm by following the steps:

Given input of size \( n \):

- Compute the necessary smaller-size inputs
- Recursively compute the corresponding outputs
- Combine these outputs.

1.1 Mergesort [Section 5.1]

The sorting problem:

**Input** An array of natural number \( A[1], \ldots, A[n] \)

**Output** A permutation of the elements in \( A \) so that they are in non-decreasing order:


For a sorting algorithm, we are interested in the total number of comparisons between elements of the arrays, i.e., comparisons of the form

\[ A[i] \leq A[j] \quad \text{or} \quad A[i] < A[j] \]

The Bubble Sort algorithm requires \( \Theta(n^2) \) comparisons.

The Merge Sort algorithm performs only \( O(n \log(n)) \) comparisons. The idea is to (i) divide the sequence into 2 halves, then (ii) sort each half separately, and then (iii) merge two sorted halves.

**Mergesort**\((A, p, r)\): % sort the subarray \( A[p \ldots r] \)

1. If \( r \leq p \) return
2. Else If \( r = p + 1 \)
4. Else
5. \( q \leftarrow \left\lceil \frac{p+r}{2} \right\rceil \) % midpoint
6. Mergesort \((A, p, q)\)
7. Mergesort \((A, q + 1, r)\)
8. Merge\((A, p, q, r)\)

Here Merge\((A, p, q, r)\) merges the two sorted subarrays \( A[p \ldots q] \) and \( A[(q+1) \ldots r] \) into a sorted array \( A[p \ldots r] \).
**Merge** \((A, p, q, r)\) \% Merge sorted \(A[p \ldots q]\) and \(A[(q + 1) \ldots r]\)

- Copy \(A[p \ldots q]\) to \(B[1 \ldots (q - p + 1)]\)
  1. \(i \leftarrow p; j \leftarrow 1\)
  2. while \(i \leq q\) do
  3. \(B[j] \leftarrow A[i]\)
  4. \(i \leftarrow i + 1; j \leftarrow j + 1\)
  5. end while

- Merge
  6. \(k \leftarrow p; i \leftarrow q + 1; j \leftarrow 1\)
  7. while \(j \leq (q - p + 1)\) do
  8. if \(i > r\) then
  9. \(A[k] \leftarrow B[j]; j \leftarrow j + 1\)
  10. else do
  11. if \(A[i] < B[j]\) then do
  12. \(A[k] \leftarrow A[i]; i \leftarrow i + 1\)
  13. else do
  14. \(A[k] \leftarrow B[j]; j \leftarrow j + 1\)
  15. end if
  16. end if
  17. \(k \leftarrow k + 1\)
  18. end while

**Number of comparisons** Merge two sorted subarray of total length \(n\) requires \(\Theta(n)\) comparisons. So the number of total comparison \(T(n)\) for Mergesort satisfies:

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

By the Master Theorem below, \(T(n) = \Theta(n \log(n))\).

**1.2 The Master Theorem** [see also Section 5.2]

If for some \(a, b, d > 0\):

\[
T(n) = aT(n/b) + \Theta(n^d)
\]

then

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

**1.3 Integer Multiplication Problem** [Section 5.5]

**Input** Two numbers in binary: \(A = a_{n-1} \ldots a_0, \ B = b_{n-1} \ldots b_0\)

**Output** The product \(A \cdot B\).
Note that

\[ A = a_{n-1} \ldots a_0 = \sum_{i=0}^{n-1} a_i 2^i, \quad B = b_{n-1} \ldots b_0 = \sum_{i=0}^{n-1} b_i 2^i \]  

(1)

The “School” Algorithm

1. Compute the “table” with \( n \) rows, where row \( i \) is \( R_i = A \cdot b_i 2^i \) (when \( b_i = 0 \), \( R_i = 0 \), and when \( b_i = 1 \), \( R_i \) is obtained from \( A \) by shifting left \( i \) bits), for \( 0 \leq i \leq n - 1 \).
2. Compute the sum \( R_0 + R_1 + \ldots + R_{n-1} \) by performing \( n - 1 \) additions.

Note that in general the rows have \( \Theta(n) \) bits, so adding two rows takes time \( \Theta(n) \). Therefore adding \( n \) rows takes time \( \Theta(n^2) \).

Divide-and-Conquer Algorithms We use the following observation. Assume that \( n \) is even. First, let \( A_1, A_0, B_1, B_0 \) be the \( (n/2) \)-bits numbers:

\[ A_1 = a_{n-1} \ldots a_{n/2} \quad A_0 = a_{n/2-1} \ldots a_0 \]
\[ B_1 = b_{n-1} \ldots b_{n/2} \quad B_0 = b_{n/2-1} \ldots b_0 \]

then

\[ A = A_1 2^{n/2} + A_0 \quad B = B_1 2^{n/2} + B_0 \]

Now

\[ A \cdot B = (A_1 \cdot B_1) 2^n + (A_1 \cdot B_0 + A_0 \cdot B_1) 2^{n/2} + A_0 \cdot B_0 \]  

(2)

The First Attempt Assuming the length \( n \) is a power of 2. Using (2), we will compute \( A \cdot B \) recursively as follows:

\textbf{Mult1}(A, B)

1. If \( n = 1 \) Return \( a_0 \cdot b_0 \)
2. Else
3. Compute \( A_1, A_0, B_1, B_0 \)
4. \( C \leftarrow \text{Mult1}(A_1, B_1) \)
5. \( D_1 \leftarrow \text{Mult1}(A_1, B_0) \)
6. \( D_2 \leftarrow \text{Mult1}(A_0, B_1) \)
7. \( E \leftarrow \text{Mult1}(A_0, B_0) \)
8. Return \((C 2^n + (D_1 + D_2) 2^{n/2} + E)\).
9. End If

Running time Addition and multiplying by \( 2^n \) or \( 2^{n/2} \) (on line 8) take time \( \Theta(n) \). So the running time \( T(n) \) satisfies

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

Therefore, by the Master Theorem, \( T(n) = \Theta(n^2) \). Asymptotically this does not improve on the “school algorithm” above.

The Second Attempt The improvement comes from the observation that

\[ A_1 \cdot B_0 + A_0 \cdot B_1 = (A_1 + A_0) \cdot (B_1 + B_0) - (A_1 \cdot B_1 + A_0 \cdot B_0) \]

There are only 3 multiplications of numbers of length \( n/2 \). Hence the number of recursive calls is 3.
**Mult2**(*A, B*)

1. If *n* = 1  
   Return *a₀ * *b₀*
2. Else
3. Compute *A₁, A₀, B₁, B₀*
4. *C* ← **Mult2**(*A₁, B₁*)
5. *E* ← **Mult2**(*A₀, B₀*)
6. *F₁* ← *A₁ + A₀, F₂* ← *B₁ + B₀*
7. *G* ← **Mult2**(*F₁, F₂*)
8. *D* ← *G − (C + E)*
9. Return \((C2^n + D2^n/2 + E)\)
10. End If

**Running time** As before, addition, subtraction and multiplying by \(2^n\) or \(2^{n/2}\) on lines 6, 8, 9 take time \(\Theta(n)\). So the running time \(T(n)\) of **Mult2** satisfies

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

The Master Theorem gives \(T(n) = \Theta(n^{\log_2 3})\).

### 1.4 The Matrix Multiplication Problem [CLRS 28.2]

**Input**  \(A, B\): \(n \times n\) matrices.

**Output**  \(A \times B\).

We are interested in the total number of operations (integer addition, integer multiplication) performed by an algorithm that computes \(A \times B\). Note that if we compute the product \(A \times B\) using the formula

\[
c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}
\]

then each \(c_{i,j}\) requires \(\Theta(n)\) operations (\(n\) integer multiplications and \(n\) integer additions). So the total number of operations is \(\Theta(n^3)\).

For the divide-and-conquer approach, we write

\[
A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = A \times B = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}
\]

**Computing** \(C₁₁, C₁₂, C₂₁, C₂₂\):

\[
C_{11} = A_{11}B_{11} + A_{12}B_{21}, \quad C_{12} = A_{11}B_{12} + A_{12}B_{22} \\
C_{21} = A_{21}B_{11} + A_{22}B_{21}, \quad C_{22} = A_{21}B_{12} + A_{22}B_{22}
\]

**The First Attempt** Using the formulas above, we have the following algorithm **MMult** (assuming that \(n\) is a power of 2):
\textbf{MMult}(A, B)

1. If $n = 1$  Output $A \times B$
2. Else
3. \hspace{1em} Compute $A_{11}, B_{11}, \ldots, A_{22}, B_{22}$ \% by computing $m = n/2$
4. \hspace{1em} $C_{11} \leftarrow \text{MMult}(A_{11}, B_{11}) + \text{MMult}(A_{12}, B_{21})$
5. \hspace{1em} $C_{12} \leftarrow \text{MMult}(A_{11}, B_{12}) + \text{MMult}(A_{12}, B_{22})$
6. \hspace{1em} $C_{21} \leftarrow \text{MMult}(A_{21}, B_{11}) + \text{MMult}(A_{22}, B_{21})$
7. \hspace{1em} $C_{22} \leftarrow \text{MMult}(A_{21}, B_{12}) + \text{MMult}(A_{22}, B_{22})$
8. \hspace{1em} Output $C$
9. End If

Matrix addition (online 4–7) needs $\mathcal{O}(n^2)$ operations (integer addition). So the total number of operations $T(n)$ of MMult when $A$ and $B$ have size $n \times n$ satisfies

$$T(n) = 8T(n) + \Theta(n^2)$$

By the Master Theorem, $T(n) = \Theta(n^3)$. This is NOT an improvement over the straightforward $\Theta(n^3)$ algorithm. Next week we will learn Strassen’s algorithm which needs only $\Theta(n \log_2(7))$ operations.