CSC 373 H 1 Y — Summer 2007

University of Toronto — St. George Campus

Lecture Summary for Week 1

This summary is not a replacement for the lecture. If you miss a class, please arrange with a friend to take note for you.

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:

$$A[1] \leq A[2] \leq \ldots \leq A[n]$$

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

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

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

The Merge Sort algorithm performs only $\mathcal{O}(n \log(n))$ 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 $r = p + 1$
3. If $A[p+1] < A[p]$ swap $A[p]$ and $A[p+1]$ End If
4. Else
5. $q \leftarrow \lceil \frac{p+r}{2} \rceil$ % 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 $A[p \dots q]$ and $A[(q+1) \dots r]$

- Copy $A[p \dots q]$ to $B[1 \dots (q-p+1)]$

```
i ← p; j ← 1
while i ≤ q do
    B[j] ← A[i]
    i ← i + 1; j ← j + 1
end while
```

- Merge

```
k ← p; i ← q + 1; j ← 1
while j ≤ (q − p + 1) do
    if i > r then
        A[k] ← B[j]; j ← j + 1
    else do
        if A[i] < B[j] then do
            A[k] ← A[i]; i ← i + 1
        else do
            A[k] ← B[j]; j ← j + 1
        end if
    end if
    k ← k + 1
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: $A = a_{n-1} \ \dots \ a_0, \quad B = b_{n-1} \ \dots \ b_0$
Output The product $A \cdot B$.

Note that

$$A = a_{n-1} \ \ldots \ a_0 = \sum_{i=0}^{n-1} a_i 2^i, \qquad B = b_{n-1} \ \ldots \ b_0 = \sum_{i=0}^{n-1} b_i 2^i \tag{1}$$

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:

$$\begin{aligned} A_1 &= a_{n-1} \ \ldots \ a_{n/2} \qquad & A_0 &= a_{n/2-1} \ \ldots \ a_0 \\ B_1 &= b_{n-1} \ \ldots \ b_{n/2} \qquad & B_0 &= b_{n/2-1} \ \ldots \ b_0 \end{aligned}$$

then

$$A = A_1 2^{n/2} + A_0 \qquad B = B_1 2^{n/2} + B_0$$

Now

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

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

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. $C \leftarrow$ Mult1(A_1, B_1)
5. $D_1 \leftarrow$ Mult1(A_1, B_0)
6. $D_2 \leftarrow$ Mult1(A_0, B_1)
7. $E \leftarrow$ Mult1(A_0, B_0)
8. Return $(C2^n + (D_1 + D_2)2^{n/2} + E)$.
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, $T(n) = \Theta(n^2)$. Assymptotically 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 $a_0 \cdot b_0$
2. Else
3. Compute A_1, A_0, B_1, B_0
4. $C \leftarrow$ Mult2(A_1, B_1)
5. $E \leftarrow$ Mult2(A_0, B_0)
6. $F_1 \leftarrow A_1 + A_0$, $F_2 \leftarrow B_1 + B_0$
7. $G \leftarrow$ Mult2(F_1, F_2)
8. $D \leftarrow G - (C + E)$
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 $C_{11}, C_{12}, C_{21}, C_{22}$:

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \qquad & C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \qquad & C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

The First Attempt Using the formulas above, we have the following algorithm MMult (assuming that n is a power of 2):

MMult(A, B)

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