

CS502 Spring 2004
Homework 1 Solutions

Problem 1. Consider the 7 log identities at the bottom page 53 in CLR. Assuming that the first four identities are true (and using whatever basic facts about logarithms you like) derive the last 3 identities. For any $a, b, n > 0$

$$\begin{aligned}\log_b(1/a) &= -\log_b a \\ \log_b a &= 1/\log_a b \\ a^{\log_b n} &= n^{\log_b a}\end{aligned}$$

Problem 2. Prove the following formula by induction on n . For all $n \geq 1$:

$$\sum_{i=0}^{n-1} i(i-1)(i-2) = \frac{n(n-1)(n-2)(n-3)}{4}$$

Problem 3. You are given two multiple precision positive integers expressed as two arrays, $A[0..n]$ and $B[0..n]$. Each entry of each array contains a single decimal digit from 0 to 9, such that the least significant digit is stored in the 0th entry of the arrays. For example, for $n = 5$, the numbers $A = 9,725$ and $B = 153,496$ would be stored as $A[0..5] = \langle 5, 2, 7, 9, 0, 0 \rangle$ and $B[0..5] = \langle 6, 9, 4, 3, 5, 1 \rangle$.

- (a) Give pseudocode for a procedure which, given A , B , and n , computes the *sum* of A and B and stores the result in array $C[0..n+1]$. Briefly explain how your procedure works. Remember, that when writing pseudocode, keep the description simple and to the point. You do not need to give a complete program with full declarations. Let your explanation handle any missing details.
- (b) Give pseudocode for a procedure which, given A , B , and n , computes the *product* of A and B and stores the result in an array $D[0..2n+1]$. You may invoke the procedure from part (a) in solving this part. Briefly explain how your procedure works.

Solution 1:

(a) $\log_b(1/a) = \log_b a^{-1} = -1 \cdot \log_b a = -\log_b a$.

(b) $\log_b a = (\log_a a)/(\log_a b) = 1/(\log_a b)$.

(c) We make use of the fact that the log function is 1-1, so that to show that $X = Y$ for positive X and Y , it suffices to show that $\log_b X = \log_b Y$.

$$\log_b(a^{\log_b n}) = (\log_b n)(\log_b a) = (\log_b a)(\log_b n) = \log_b(n^{\log_b a}).$$

Solution 2: The proof is by induction on n . We will use strong induction, which means that we

will assume that the induction hypothesis holds for all values strictly smaller than n and then prove the hypothesis for n .

Basis: ($n = 1$) The value of the summation is

$$\sum_{i=0}^0 i(i-1)(i-2) = 0.$$

The formula gives $(1 \cdot 0 \cdot -1 \cdot -2)/4 = 0$. Since these are equal, we have proved the basis case.

Induction step: For the induction step, we have $n \geq 2$, and we assume (by the induction hypothesis) that the formula holds whenever we plug in a value n' that is strictly smaller than n . In particular, it holds when $n' = n - 1$. We first write down the summation in the form that we want to prove, and then strip off the last term.

$$\begin{aligned} \sum_{i=0}^{n-1} i(i-1)(i-2) &= \left(\sum_{i=0}^{n-2} i(i-1)(i-2) \right) + (n-1)(n-2)(n-3) \\ &= \left(\sum_{i=0}^{(n-1)-1} i(i-1)(i-2) \right) + (n-1)(n-2)(n-3) \end{aligned}$$

Observe that the summation that remains is of exactly the same form as the original, but it stops at $n - 1$ rather than n . By the induction hypothesis, we may replace this summation with the formula, but using the value $n - 1$ in place of n . When we do this we get the following:

$$\begin{aligned} \sum_{i=0}^{n-1} i(i-1)(i-2) &= \frac{(n-1)(n-2)(n-3)(n-4)}{4} + (n-1)(n-2)(n-3) \\ &= \frac{(n-1)(n-2)(n-3)((n-4)+4)}{4} = \frac{n(n-1)(n-2)(n-3)}{4} \end{aligned}$$

This is exactly equal to the desired formula, and so this completes the proof.

Solution 3:

(a) This procedure works by the usual additional rule which you learned in elementary school: add corresponding digits along with the carry from the least to most significant digits. If s is this value and it is 10 or larger, then we store $s \bmod 10$ (the remainder of $s/10$) in the answer, and carry $s/10$. This is a bit of overkill (since the carry can only be 0 or 1), but we do it for symmetry with the multiplication procedure that follows. The carry is stored in variable `carry` (initially 0) and the sum is stored in `sum`.

```
Add(A, B, n)
{
  C[0..n+1] = 0; // zero out C
  carry = 0; // initial carry is 0
```

```

for i = 0 to n
{
    sum = A[i] + B[i] + carry;
    C[i] = sum % 10; // the result
    carry = sum / 10; // the new carry
}
C[n+1] = carry; // handle any left-over carry
return C;
}

```

(b) This procedure also works by the standard elementary school algorithm. For i running from 0 to n , we compute the product of the entire number represented in A times the single digit $B[i]$. To compute this product we multiply each digit $A[j]$ times each digit $B[i]$, and need to accumulate the results. Rather than using separate storage for the intermediate results, we will accumulate everything in D . When multiplying $A[j]$ by $B[i]$, the result should be placed in digit $D[i + j]$. As before, we need to add in the previous carry, and keep track of any new carry. We use the same rule as above to generate the result and new carry.

```

Mult(A, B, n)
{
    D[0..2n+1] = 0; // zero out D
    for i = 0 to n // multiply A by the single digit B[i]
    {
        carry = 0; // initial carry
        for j = 0 to n
        {
            sum = D[i+j] + A[j]*B[i] + carry;
            D[i+j] = sum % 10; // accumulate the result in D
            carry = sum / 10; // new carry
        }
        D[i+n+1] = carry; // handle any left-over carry
    }
    return D;
}

```

Note that the pseudo code is rather free form. However, it does clearly outline the steps involved. For example, the zeroing out of an entire array would require a loop; the algorithm indicates this as simply $C[0..n + 1] = 0$.

Analysis: Although you were not asked to do so, the “zeroing out” operation would take $\Theta(n)$ time in the analysis. The running time of part (a) is $\Theta(n)$, and the time for part (b) is $\Theta(n^2)$.