CS218 Design and analysis of algorithms **Jan-Apr 2024**

Exercise sheet 1

Lecture 8, 9 (Jan 23, 25) Fibonacci, Integer Multiplication

1. Let us try to apply the divide and conquer approach on the integer multiplication problem. Suppose we want to multiply two n-bit integers a and b. Write them as

$$a = a_1 2^{n/2} + a_0$$

$$b = b_1 2^{n/2} + b_0$$

The product of the two integers can be written as

$$ab = a_1 b_1 2^n + (a_1 b_0 + a_0 b_1) 2^{n/2} + a_0 b_0.$$

Can you compute these three terms $a_1 b_1$, $a_1 b_0 + a_0 b_1$, $a_0 b_0$, using only **three multiplications of** $n/2$ **bit integers** and a few additions/subtractions? If yes, then we will get an $O(n^{1.58})$ time algorithm.

2. Can you find square of an n-bit integer a, using square subroutine on **five** $n/3$ bit integers and a few additions/subtractions? You need to compute the following five terms using the square operation only 5 times.

$$P^2, PQ, 2PR + Q^2, QR, R^2.$$

3. Can you find multiplication of two n-bit integers, using the multiplication subroutine on **six** pairs of $n/3$ bit integers and a few additions/subtractions? You need to compute the following five terms using the multiplication operation only six times.

$$a_0 b_0, a_1 b_0 + a_0 b_1, a_0 b_2 + a_1 b_1 + a_2 b_0, a_1 b_2 + a_2 b_1, a_2 b_2.$$

Now, do this with only five multiplications.

4. Solve the recurrences

- $T(n) = 6T(n/3) + O(n)$.
- $T(n) = 5T(n/3) + O(n)$.
- $T(n) = 8T(n/4) + O(n)$.
- $T(n) = 7T(n/4) + O(n)$.

Arrange the items in increasing order of complexity.

5. Can you find square of an n-bit integer a, using square subroutine on **2k-1** integers with n/k bits and a some additions/subtractions? What's the running time you get? What if you take k as something like $n/2$? Does that give you a really fast algorithm?

6. Show that $n2^{\sqrt{logn}}$ is asymptotically smaller than $n^{1.01}$. That is, show that there exists a number N, such that for all $n > N$,

$$n2^{\sqrt{logn}} < n^{1.01}.$$

Do you think it works if we replace 1.01 with any constant greater than 1.

7. Write a program to compare different multiplication algorithms for multiplying 1024 bit integers. You can try the school method, Karatsuba, Toom-cook, or any combination of these.

8. **Matrix Multiplication.** Let us say we have 8 numbers $a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4$ and we consider these **seven** expressions.

$$p_1 = (a_1 + a_4)(b_1 + b_4), \quad p_2 = (a_3 + a_4)b_1, \quad p_3 = a_1(b_2 - b_4), \quad p_4 = a_4(b_3 - b_1)$$
$$p_5 = (a_1 + a_2)b_4, \qquad p_6 = (a_3 - a_1)(b_1 + b_2), \qquad p_7 = (a_2 - a_4)(b_3 + b_4)$$

(a) Compute the following four sums. This will be helpful later.

$$p_1 + p_4 - p_5 + p_7, \quad p_3 + p_5, \quad p_2 + p_4, \quad p_1 - p_2 + p_3 + p_6$$

Now, we want to apply divide and conquer technique to matrix multiplication. Let A and B be two $n \times n$ matrices, and we want to compute their product $C = A \times B$. The naive algorithm for this will take $O(n^3)$ arithmetic operations. We want to significantly improve this using divide and conquer.

A natural way to split any matrix can be this:

$$A = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix},$$

where each A_i is an $n/2 \times n/2$ matrix.

(b) Can you express the product matrix C, in terms of A_1, A_2, A_3, A_4 and B_1, B_2, B_3, B_4.

(c) Design an algorithm for matrix multiplication using divide and conquer which takes $O(\mathbf{7}^{\log_2 n}) = O(n^{\log_2 7}) = O(n^{2.81})$ time.

9. Can you apply Karatsuba's trick on polynomial multiplication and get a runtime bound better than $O(d^2)$ for degree d polynomials.