CS601 Algorithms and Complexity **Jul-Nov 2021**

Homework (No submission)

Lecture 1 (July 29)

- Given two sorted integer arrays (with all distinct numbers in them) of size n, we want to find the median of the union of the two arrays. Can you find it by accessing only $O(\log n)$ entries in the two arrays.
- Given an array of n integers and an integer S, find a pair of integers in the array whose sum is S. Can you do it in $O(n \log n)$ time?
- Let $f : \mathbb{R} \to \mathbb{R}$ be a convex function that is promised to have a minimizing point. The function $f(x)$ and its derivative $f'(x)$ are not given explicitly, but via oracle access. That is, you can give any point $x \in \mathbb{R}$ and the oracle will give the values of $f(x)$ and $f'(x)$. How many queries do you need you find the point minimizing $f(x)$?

Lecture 3 (Aug 3)

- Suppose you are given a function *Random(k)*, that on an integer input k, gives you a uniform random integer from the range $\{1, 2, \ldots, k\}$. Using this function, can you generate a uniform random permutation of $\{1, 2, \ldots, n\}$, for any given n. That is, each permutation should be the outcome with probability $1/(n!)$.
- Design an algorithm to find the minimum and the second-minimum element in an array such that the total number of comparisons at most $n + \log n$, where n is the array size. Note that we don't care if the overall running time is much larger. The goal is to have as small a number of comparisons as possible. To get a hint, you can check the *Repechage* rule in Olympics wrestling.

Lecture 4 (Aug 5)

- Write a proof for linearity of expectation.
- Design a randomized algorithm for finding three smallest elements in an array. In each iteration, there should be a good probability of having only one comparison.

Lecture 5 (Aug 9)

- Prove that for any integer $n \geq 2$,
$$\sum_{i=2}^{n} \frac{1}{i} \leq \log_e n.$$
- Let X be a random variable which takes values between n and $2n$. If its expectation $\mathbb{E}[X] \leq n + \alpha$, then using Markov's inequality show that
$$\Pr[X \geq n + k\alpha] \leq 1/k.$$

- Implement the randomized algorithm discussed in the class for finding minimum and second-minimum elements in an array. To get an estimate of the expected number of comparisons, repeat the algorithm a few times and take the average of the number of comparisons. See how the expected number of comparisons is growing with n.
- Can you design a randomized algorithm to find the third-minimum element in an array such that the expected number of comparisons is $n + O(\log n)$.

Lecture 6 (Aug 10)

- In the longest increasing subsequence (LIS) problem, you are given an array of integers and you need to find a subsequence (a subset, not necessarily contiguous) that is increasing and has the longest length. Design an $O(n^2)$ time algorithm for this.

  Here is one way to think about it. If someone already gives you the LIS S_{n-1} in the subarray $A[1 \dots n-1]$, can you use it to find the LIS S_n in the array $A[1 \dots n]$? If it so happens that $A[n]$ is larger than the last element in S_{n-1}, then we can simply append $A[n]$ to S_{n-1} and that will give us S_n. However, what if it's not true? We can divide all possible increasing subsequences of $A[1 \dots n]$ into two groups, one with those which end with $A[n]$, and the second with those which don't end with $A[n]$. From the second group, we already know the longest sequence – it's S_{n-1}? What about the first group? May be we can strengthen the subproblem to give us the longest sequence from the first group as well.

  Here is the more general problem we can consider – find S_n that is the LIS in $A[1 \dots n]$ and find T_n that is the LIS in $A[1 \dots n]$ which ends with $A[n]$. Now, if S_i and T_i are already known for each $i \leq n-1$, can you compute S_n and T_n?
- We want to find the n-th Fibonacci number F_n in only $O(\log n)$ arithmetic operations. The idea is to reduce it to matrix exponentiation. If you prove the following, the algorithm will be immediately clear.

  There is a 2×2 matrix M such that

  $$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = M^{n-1} \begin{bmatrix} F_1 \\ F_0 \end{bmatrix}.$$

  Can you say what should be M? Can you compute M^{n-1} in $O(\log n)$ arithmetic operations?

Lecture 7 (Aug 12)

We saw an iterative approach for the hidden/visible lines problem. We asked if given a solution for $n-1$ lines, can we build a solution for n lines. Here are some questions to think about.

- Prove that the total number of intersections between the n-th line and the current set of visible segments is at most 2.
- Can you do a binary search for these two intersection points, and thus, needing only $O(\log n)$ time?
- Once you find the two intersection points, can you update the set of visible segments in $O(\log n)$ time? Do you need a different data structure? Is there a data structure where you can update in $O(\log n)$ time and also do the binary search from the previous step?
- Alternatively, can you design a divide and conquer algorithm for the problem that takes $O(n \log n)$ time overall.

Here are some typical recurrences which are seen in divide and conquer approach. Can you solve them?

- $T(n) \leq 2T(n/2) + 5n^2$
- $T(n) \leq 3T(n/2) + 7n$
- $T(n) \leq 7T(n/2) + 3n^2$

Lectures 8, 9 (Aug 16,17)

- Write a code for solving the hidden/visible lines problem discussed in the class using the Divide and Conquer approach.
- We had seen an $O(n^{1.58})$ time algorithm for squaring an integer. Does this easily give us an algorithm for multiplying two integers with the same running time?
- Let us try to directly 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_1b_1, $a_1b_0 + a_0b_1$, a_0b_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.

  Hint: Start with multiplying $(a_0 + a_1)(b_0 + b_1)$. Which do two other multiplications will you do?
- Going back to the squaring problem, what if divided the integer a into three parts instead of two? Can we get a better running time?

  – Can you find square of an n-bit integer a, using squaring subroutine on **six** $n/3$ bit integers and a few additions/subractions? What's the running time you get?

  – Can you find square of an n-bit integer a, using squaring subroutine on **five** $n/3$ bit integers and a few additions/subtractions? What's the running time you get?

Divide and conquer exercises

- **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.

- **Dominating points.** You are given a list of 2D points. A point (α, β) is said to be dominated by another point (γ, δ) if $\alpha \leq \gamma$ and $\beta \leq \delta$. Give an $O(n \log n)$ time algorithm to find the number points that are not dominated by any other point.
- **Majority Fingerprints.** You are given a collection of n fingerprints. You are also told that more than $(n/2)$ of these are identical to each other. You are only given access to an equality test, which takes two fingerprints and tells whether they are identical or not. Using this equality test, can you find out the one with more than $n/2$ copies in $O(n \log n)$ time?
- **Significant inversions.** In an array A of integers, a pair is called a significant inversion if $i < j$ and $A[i] > 2A[j]$. Design an $O(n \log n)$ time algorithm to find the number of significant inversions.

Lecture 10 (Aug 26)

We saw how divide and conquer approach gives better time complexity for integer squaring/multiplication. In particular, we saw that when we divide the integer into 3 parts instead of 2, we get better time complexity. Suppose we divide the integer into k parts. Then what is the best recurrence relation you can get?

$$T(n) = \ldots\ldots T(n/k) + O(n).$$

Do you think the multiplicative factor in the above relation can be something linear like k or $2k$, or will be quadratic like k^2?

Suppose it is possible possible to get a linear multiplicative factor, say something like $2k$. Then let's take $k = n/2$. Does that give $T(n) = O(n)$? There must be something wrong here.