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 \(f : \mathbb{R} \rightarrow \mathbb{R}\) 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 \(\text{Random}(k)\), 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 \textit{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 \( A[1 \ldots n-1] \), can you use it to find the LIS \( S_n \) in the array \( A[1 \ldots n] \)? 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 \( A[1 \ldots n] \) 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? Maybe 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 \( A[1 \ldots n] \) and find \( T_n \) that is the LIS in \( A[1 \ldots n] \) 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 \( 2 \times 2 \) 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?

\[
\begin{align*}
T(n) &\leq 2T(n/2) + 5n^2 \\
T(n) &\leq 3T(n/2) + 7n \\
T(n) &\leq 7T(n/2) + 3n^2
\end{align*}
\]