

Problem 1. Unimodal Array Maximum. You are given an array $A = [a_1, \dots, a_n]$ of n distinct numbers that is *unimodal*, that is, for some index $p \in \{1, 2, \dots, n\}$, the values in the array entries increase up to index p and then decrease from index $p + 1$ till index n . The problem is to find the peak entry p by reading as few array entries as possible. Show how to find the peak index p in time $O(\log n)$.

Note. 1. Use a divide and conquer approach, taking cue from the fact that the solution to the recurrence equation $T(n) = T(n/2) + c$ is $T(n) = O(\log n)$, where, c is the constant. (This solution holds even for the equation $T(n) = T(\lceil n/2 \rceil) + c$). So design an algorithm so that after a constant amount of work, you can discard one half of the current sub-array, as in binary search. 2. If you were to plot a unimodal array, with array indices j on the x -axis and array entries $A[j]$ on the y -axis, the plotted points will rise until x -value p where it attains a maximum, and then falls from thereon.

Problem 2. Counting Inversions. Given an array $A = [a_1, a_2, \dots, a_n]$ of n integers (with possible repetitions), we say that, for $1 \leq i < j \leq n$, a pair (i, j) is an inversion, if $a_i > a_j$. Give an $O(n \log n)$ algorithm to count the number of inversions.

Problem 3. Counting Significant Inversions. Given an array $A = [a_1, a_2, \dots, a_n]$ of n integers, we say that a pair (i, j) with $i < j$ is a *significant inversion* if $a_i > 2a_j$. Give an $O(n \log n)$ algorithm to compute the number of significant inversions in A .

Problem 4. Median of union of two sorted arrays. Given two arrays $A = [a_1, a_2, \dots, a_n]$ and $B = [b_1, b_2, \dots, b_m]$ that are each individually sorted in increasing order. Assume that the numbers in $A \cup B$ are all distinct. You have to find the median of $A \cup B$ in time $O(\log n)$.

Problem 5. Hadamard Matrices. Hadamard matrices H_n are square $2^n \times 2^n$ matrices and are defined as follows.

1. H_0 is the 1×1 matrix $[1]$.

2. For $k \geq 1$, H_k is the $2^k \times 2^k$ matrix $H_k = \frac{1}{\sqrt{2}} \begin{bmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{bmatrix}$

Show that if v is a column vector of length $n = 2^k$, then the matrix-vector product $H_k v$ can be calculated in $O(n \log n)$ operations. Assume that the numbers in v are small enough so that basic arithmetic operations like addition and multiplication take unit time. (*Note.* An interesting property of the Hadamard matrices is that it is (real) orthonormal, that is, $H_k^T H_k = I$, analogous to the DFT matrix F_n .)

Problem *6. Closest pair of points in 2-dimensions. Given n points in the plane, the problem is to find the pair that is the closest in terms of Euclidean distance. Design an $O(n \log n)$ algorithm for this problem.

Note. You may assume that each point is a pair (x, y) and has an id (between $1, 2, \dots, n$). Let P be an array of points. Assume also that the Euclidean distance between two points can be

calculated in constant time. Note that an $O(n^2)$ algorithm is trivial, since one checks the distance between all $\binom{n}{2}$ pairs of points and returns the pair that has the minimum distance. The $O(n \log n)$ time algorithm is a divide and conquer algorithm with a careful geometric analysis for the combine phase. A solution may be found in the text CLRS Section 33.4.

The overall approach is the following. Let P_x be a copy of P sorted on the x -coordinate and P_y be a copy of P sorted on the y coordinate. The two copies are mutually synced in the following sense (or an equivalent implementation of it). For every point p in P_x , there is a field of p that points to the copy of p in P_y and vice-versa. Partition P by the x coordinate into two equal halves Q (the left half) and R (the right half). Q contains the first $n/2$ (actually, $\lceil n/2 \rceil$) points (by x -coordinate) of P_x and R contains the remaining $n/2$ ($\lfloor n/2 \rfloor$) points. Now in $O(n)$ time, using the mutual syncing of P_x and P_y , in $O(n)$ time, we can create four lists Q_x and Q_y , which are points in Q sorted by x and y coordinates respectively and mutually synced, and lists R_x and R_y , which are points in R sorted by x and y coordinates respectively and mutually synced. Let L be a vertical line passing through the $\lceil n/2 \rceil$ th point, that is the point with the highest x -coordinate in Q .

Now recursively determine the closest pair of points in Q (using the lists Q_x and Q_y) and R respectively. Let q_0^*, q_1^* be returned as the closest pair in Q and r_0^*, r_1^* be returned as the closest pair in R . Let δ be the minimum of $d(q_0^*, q_1^*)$ and $d(r_0^*, r_1^*)$. This value of δ is the closest pair distance unless there is a closer pair (q, r) , $q \in Q$ and $r \in R$, where, $d(q, r) < \delta$. We only need to look for “cross-pairs” (q, r) .

Show that it suffices to look at only point pairs (q, r) that lie in a 2δ -band, where, $q \in Q$ is at a distance of at most δ from L (to its left) and $r \in R$ is within at most a distance of δ from L (to its right).

We can now delete from Q all points that are at a distance greater than δ to the left of the line L and similarly, drop all points of R that are at a distance greater than δ to the right of L . This can be done in time $O(n)$ and leaves the remaining points in Q and R sorted by y coordinate. Let $S = Q \cup R$, after the deletions have been done. Merge S into a single sorted list by y -coordinate in $O(n)$ time.

Now (the meticulous part), show that if $s, s' \in S$ satisfying $d(s, s') < \delta$, then, s and s' are within 15 positions of each other in the sorted list S_y . (Here, 15 is used to refer to an absolute constant, CLRS argues it down to 7). Hence, the merging can be done in time $O(n)$.