Problem 1 (Divide and Conquer the Peak, 30 pts)

Suppose we have an array A of n distinct integers and moreover are guaranteed that the array has the following property: up to some index $1 \leq i \leq n$, A is increasing, i.e., $A[1] < A[2] < \ldots < A[i]$, and then after index i, A is decreasing, i.e., $A[i] > A[i+1] > \ldots > A[n]$. In this array, we call $A[i]$ the *peak* of A. For example, consider the array $[1, 4, 7, 8, 6, 2]$, which has peak 8.

Create an algorithm that finds the peak of an input array A in sublinear time (i.e. $o(n)$). Justify the correctness and time complexity of your proposed algorithm.

Solution

Problem Statement:
Given an array A of n distinct integers that increases up to some index i (i.e.,

$$A[1] < A[2] < \cdots < A[i]$$

) and then decreases (i.e.,

$$A[i] > A[i+1] > \cdots > A[n]$$

), the element $A[i]$ is the *peak* of A. For example, the array

$$[1, 4, 7, 8, 6, 2]$$

has peak 8. Design an algorithm to find the peak in sublinear time (i.e., $o(n)$).

Algorithm Description:
The key observation is that since the array is first increasing and then decreasing, a binary search can be employed to efficiently find the peak. Let low and high denote the current bounds of the search interval. At each step, compute

$$\text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor .$$

Then, compare $A[\text{mid}]$ with $A[\text{mid} + 1]$:

- If $A[\text{mid}] < A[\text{mid} + 1]$, then the array is still increasing at mid, so the peak must lie in the right half. Set $\text{low} = \text{mid} + 1$.
- Otherwise, if $A[\text{mid}] > A[\text{mid} + 1]$, then the peak is either at mid or to its left, so set $\text{high} = \text{mid}$.

The loop terminates when $\text{low} = \text{high}$, and at that point, $A[\text{low}]$ is the peak.

Pseudocode:
FindPeak$A[1 \ldots n]$ **initialize** low $\leftarrow$ 1, high $\leftarrow$ n low $<$ high mid $\leftarrow \left\lfloor \frac{\text{low}+\text{high}}{2} \right\rfloor$ $A[\text{mid}] < A[\text{mid} + 1]$ low $\leftarrow$ mid + 1 high $\leftarrow$ mid **return** $A[\text{low}]$
[1]

Correctness:
The algorithm maintains the invariant that the peak is always within the interval [low, high]. At each iteration:

- If $A[\text{mid}] < A[\text{mid}+1]$, the increasing property guarantees that the peak lies in the interval $[\text{mid}+1, \text{high}]$.
- If $A[\text{mid}] \geq A[\text{mid}+1]$, the peak is in $[\text{low}, \text{mid}]$ since either $A[\text{mid}]$ is the peak or the peak lies to its left.

When the loop terminates with low = high, only one element remains, and by our invariant, it must be the peak.

Time Complexity:
At each iteration, the search space is reduced by roughly half. Therefore, the number of iterations is $O(\log n)$, which is sublinear with respect to n.

Conclusion:
The proposed binary search algorithm correctly identifies the peak element in a unimodal array in $O(\log n)$ time. x

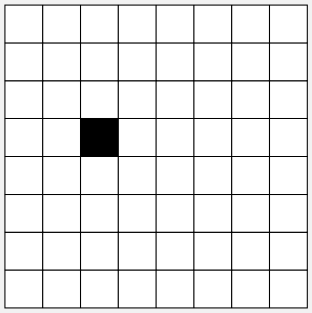
Problem 2 (Two-Server Search, 35 pts)

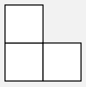
You have (limited) access to two databases, each of which contains n numbers. For simplicity, you may assume each value across the $2n$ entries is unique. You may only query the databases in the following way: You give one of the databases an integer $1 \leq k \leq n$, and it responds with the k-th smallest number in its database.

Give a divide-and-conquer algorithm that finds the median value across all $2n$ entries using $O(\log n)$ queries to the servers. Justify the correctness and time complexity of your proposed algorithm.

Problem 3 (Grid Tiling, 35 pts)

For a positive integer n, consider a $2^n \times 2^n$ grid, where one of the unit squares is black and all others are white. Show that regardless of the position of the black unit square, the white area can be fully covered, without any overlapping, by L-shaped tiles consisting of 3 unit squares (rotations are allowed).





Left: an example grid with $n = 3$; Right: the L-shaped tile

(*Hint: Use a proof by construction, i.e. design a divide-and-conquer algorithm that takes as input n and (i, j) with $1 \leq i, j \leq 2^n$, and outputs a way to tile a $2^n \times 2^n$ grid with the black square at position (i, j).*)