Here are several problems that are easy to solve in $O(n)$ time, essentially by brute force. Your task is to design algorithms for these problems that are significantly faster using binary search related ideas.

1. Suppose we are given an array $A[1..n]$ of n *distinct integers*, which could be positive, negative, or zero, sorted in increasing order so that $A[1] < A[2] < \cdots < A[n]$.

   (a) Describe a fast algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists.

   (b) Suppose we know in advance that $A[1] > 0$. Describe an even faster algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists. *[Hint: This is **really** easy.]*

2. Suppose we are given an array $A[1..n]$ such that $A[1] \geq A[2]$ and $A[n-1] \leq A[n]$. We say that an element $A[x]$ is a ***local minimum*** if both $A[x-1] \geq A[x]$ and $A[x] \leq A[x+1]$. For example, there are exactly six local minima in the following array:

9	**7**	7	2	**1**	3	7	5	**4**	7	**3**	**3**	4	8	**6**	9
	▲			▲				▲		▲	▲			▲	

   Describe and analyze a fast algorithm that returns the index of one local minimum. For example, given the array above, your algorithm could return the integer 9, because $A[9]$ is a local minimum. *[Hint: With the given boundary conditions, any array **must** contain at least one local minimum. Why?]*

3. Suppose you are given two sorted arrays $A[1..n]$ and $B[1..n]$ containing distinct integers. Describe a fast algorithm to find the median (meaning the nth smallest element) of the union $A \cup B$. For example, given the input

   $$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \qquad B[1..8] = [2, 4, 5, 8, 17, 19, 21, 23]$$

   your algorithm should return the integer 9. *[Hint: What can you learn by comparing one element of A with one element of B?]*

4. Suppose you have an algorithm that given as input a directed graph $G = (V, E)$, nodes $s, t \in V$, and an integer k, outputs whether there is path from s to t in G with at most k edges. Thus the algorithm is solving a decision problem. Now you want to use this decision algorithm as a black box to find the length of the shortest path from s to t, which is an optimization problem. How do you reduce the optimization problem to the decision problem? What is an upper bound on the number of calls to the decision problem that your optimization algorithm makes? Assume n is the number of nodes in G. Now suppose the graphs has non-negative integer edge lengths with U being the largest edge length and $L \geq 1$ being the smallest edge length (all lengths are integers). Now the decision version is: given G, s, t and an integer T is there an s-t path of length at most T? How many calls to the decision version will your optimization algorithm make? Is it polynomial in the input length?

To think about later:

5. Now suppose you are given two sorted arrays $A[1..m]$ and $B[1..n]$ and an integer k. Describe a fast algorithm to find the kth smallest element in the array of size $m + n$ containing the numbers in both A and B (assume numbers are distinct for simplicity). Given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \qquad B[1..5] = [2, 5, 7, 17, 19] \qquad k = 6$$

   your algorithm should return the integer 7.

6. Suppose you have an algorithm that given as input a directed graph $G = (V, E)$, nodes $s, t \in V$, and an integer k, outputs whether the *number* of distinct shortest paths from s to t is at least k. Describe an algorithm that counts the number of distinct shortest s-t paths in G. Does your algorithm run in polynomial time?