Problem 5-1 (Running Median 2) 10 points

In homework 4, we designed a data structure supporting the following operations:

- **BUILD**(A[1...n]): Initializes the data structure with the elements of the array A in time O(n).
- **INSERT**(x): Inserts the element x into the data structure in time O(log n), where n is the number of elements stored in the data structure.
- **MEDIAN**: Returns the median\(^1\) in time O(1) of the currently stored elements.

In this problem we assume that all elements are integer numbers from 1 to 10^6, and we will design a data structure that supports the same set of operations but inserts an element in constant time.

(a) (3 points) Describe a data structure such that you can perform the operations BUILD, INSERT, and MEDIAN with running times as required below.

Describe in pseudo-code implementations of

(b) (3 points) **BUILD** running in time O(n).

(c) (1 points) **INSERT** running in time O(1), and

(d) (3 points) **MEDIAN** running in time O(1).

Problem 5-2 (The kth Smallest Element) 14 points

The standard array-based implementation of **MinHeap** supports, in particular, the following operations\(^2\):

- **EXTRACTMIN**: Returns a minimum element and removes it from the heap in time O(log n).
- **INSERT**(x): Inserts the element x into the heap in time O(log n).
- Let A be the array that represents the heap. Recall that the root of the heap is A[1], and the indices of the children of i are 2i and 2i + 1.

Consider the task of, on input array-based **MinHeap**, outputting the kth smallest element in the heap.

---

\(^1\)The median of n elements is the \([\frac{n}{2}]\)-largest element.

\(^2\)Here n denotes the number of elements in the heap.
(a) (2 points) Develop an algorithm that outputs the $k$th smallest element in the heap in time $O(k \log n)$, justify its correctness. (You are allowed to change the original heap.)

(b) (2 points) Show how to find the 2nd smallest element in the heap in time $O(1)$, justify your answer.

(c) (2 points) Show how to find the 3rd smallest element in the heap in time $O(1)$, justify your answer.

(d) (8 points) Using intuition developed in parts (b) and (c), develop an algorithm that outputs the $k$th smallest element in the heap in time $O(k \log k)$. Write pseudo-code of your algorithm, justify its correctness.

(Hint: Introduce an auxiliary heap. Remember that the heap is array-based, thus you can find children of an element of the heap in constant time.)

Problem 5-3 (Nuts and Bolts) 15 points

Assume that we are given $n$ bolts and $n$ nuts of different sizes, where each bolt exactly matches one nut. Our goal is to find the matching nut for each bolt. The nuts and bolts are too similar to compare directly; however, we can test whether any nut is too big, too small, or the same size as any bolt.

(a) (4 points) Prove that in the worst case, $\Omega(n \log n)$ nut-bolt tests are required to correctly match up the nuts and bolts.

(b) (6 points) Prove that in the worst case, $\Omega(n + k \log n)$ nut-bolt tests are required to find $k$ arbitrary matching pairs.

(c) (5 points) Give a randomized algorithm that runs in expected time $O(n)$ and finds the $k$-th largest nut given any integer $k$. You may assume that it is possible to efficiently sample a random nut/bolt.

Problem 5-4 (Fun with Medians) 16 points

(a) (3 points) HALVING is the the operation that takes an array $A$ with $n$ distinct numbers and separates it into two half-sized$^3$ arrays $A_0$ and $A_1$, where all elements of $A_0$ are smaller than all elements of $A_1$. (Note that it is not required that $A_0$ and $A_1$ are sorted.)

Prove that HALVING can be done in linear time.


Give a linear-time algorithm that transforms any given array $A$ with $n$ distinct elements into a roller coaster array $B$. Namely, $B$ must contain exactly the same $n$ distinct elements as $A$, but must also be a roller coaster.

$^3$The size of $A_0$ is $\lceil n/2 \rceil$, the size of $A_1$ is $\lfloor n/2 \rfloor$. 
(c) (8 points) Describe a linear-time algorithm which, given an array $A$ with $n$ distinct elements and a number $k < n$, returns $k$ elements of $A$ which are closest to the median of $A$ (excluding the median itself).

For example, if $A = (10, 5, 11, 1, 6, 7, 25)$ and $k = 2$, the median of $A$ is 7, and 2 closest numbers to 7 are 6 and 5.