In this task you will design a data structure supporting the following operations:

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

In the following, you are allowed to use the standard heap operations from CLRS Chapter 6 (incurring the corresponding running times).

(a) (4 points) Describe in English an idea for a data structure such that you can perform the operations **BUILD**, **INSERT**, and **MEDIAN** with running times as required below. Do not formally analyze the running time yet.

**Hint**: Use a min-heap and a max-heap.

**Solution:** INSERT YOUR SOLUTION HERE

(b) (3 points) **BUILD** running in time \(O(n)\), where \(n\) is the number of elements in the data structure.

**Hint**: Use the **FIND-MEDIAN** algorithm given in class to find the median of a list in time \(O(n)\).

**Solution:** INSERT YOUR SOLUTION HERE

Using three-way partitioning ensures that duplicates do not cause trouble. Do not deduct points if a student’s solution does not handle duplicates properly (point it out to them though).

(c) (3 points) **INSERT** running in time \(O(\log n)\), where \(n\) is the number of elements in the data structure.

**Solution:** INSERT YOUR SOLUTION HERE

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

**Solution:** INSERT YOUR SOLUTION HERE

\(^1\)The median of \(n\) elements is the \(\lceil \frac{n}{2} \rceil\)-smallest element.
In the previous problem, we designed a data structure supporting the following operations:

- **BUILD**($A[1 \ldots n]$): 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 $^2$ 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 in English an idea for a data structure such that you can perform the operations **BUILD**, **INSERT**, and **MEDIAN** with running times as required below. Do not formally analyze the running time.

**Solution:** INSERT YOUR SOLUTION HERE

In parts (b)-(d), describe in pseudo-code your implementations from part (a) and analyze the running time.

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

**Solution:** INSERT YOUR SOLUTION HERE

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

**Solution:** INSERT YOUR SOLUTION HERE

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

**Solution:** INSERT YOUR SOLUTION HERE

---

$^2$The median of $n$ elements is the $\lceil \frac{n}{2} \rceil$-largest element.
Although any comparison-based sorting algorithm must take time $\Omega(n \log n)$, in some cases it is possible to obtain faster algorithms if we assume more about the structure of the input list. For each of the following assumptions, give the best lower bound on the number of comparisons for sorting you can use a decision tree argument. Because of the assumptions on the structure of the list, you might get a better lower bound than $\Omega(n \log n)$. If this is the case, give an algorithm to sort the list that matches the lower bound.

(a) (5 points) The input list is $k$-sorted, that is, the first $k$ elements are each less than the next $k$ elements, which are each less than the next $k$ elements, and so on. More specifically,

$$a_{(i-1)k+j} \leq a_{ik+\ell} \text{ for all } 1 \leq i < n/k, 1 \leq j, \ell \leq k.$$ 

**Solution:** INSERT YOUR SOLUTION HERE

(b) (5 points) The input list is sorted if you look at only the even positions. That is, the second element is less than the fourth element, which is less than the sixth element, and in general $a_{2i} \leq a_{2j}$ if $i \leq j$. However, the odd positions can be anything. You can assume without proof that $\binom{n}{n/2} \approx \frac{2^n}{\Theta(\sqrt{n})}$.

**Solution:** INSERT YOUR SOLUTION HERE

(c) (5 points) The input list is sorted if you look at only the even positions. That is, the second element is less than the fourth element, which is less than the sixth element, and in general $a_{2i} \leq a_{2j}$ if $i \leq j$. The input list is also sorted if you look at only the odd positions. That is, the first element is less than the third element, which is less than the fifth element, and in general $a_{2i+1} \leq a_{2j+1}$ if $i \leq j$. There is no assumed relationship between an element in an odd position and an element in an even position. You can assume without proof that $\binom{n}{n/2} \approx \frac{2^n}{\Theta(\sqrt{n})}$.

**Solution:** INSERT YOUR SOLUTION HERE
(a) (3 points) HALVING is the operation that takes an array $A$ with $n$ distinct numbers and separates it into two half-sized 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.

**Solution:** INSERT YOUR SOLUTION HERE


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.

**Solution:** INSERT YOUR SOLUTION HERE

(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.

**Solution:** INSERT YOUR SOLUTION HERE

---

3The size of $A_0$ is $\lceil n/2 \rceil$, the size of $A_1$ is $\lfloor n/2 \rfloor$. 