DUE: THURSDAY, FEBRUARY 16. SUBMISSIONS SHOULD BE TYPESET (LATEX
PREFERRED). EXPLAIN THE WORKING OF EACH ALGORITHM SIMPLY AND
PRECISELY. THE CORRECTNESS AND RUNNING TIME OF ALL ALGORITHMS MUST BE BACKED UP BY
FORMAL ARGUMENTS.

**Question 1 [30 points]** One of the most basic tasks in statistics is to summarize a set
of observations \( \{x_1, x_2, \ldots, x_n\} \subseteq \mathbb{R} \) by a single number. Two popular choices for this
summary statistic are: the median, which we’ll call \( \mu_1 \) and the mean (or average), which
we’ll call \( \mu_2 \).

(a) Show that the median is the value of \( \mu \) that minimizes the function

\[
\sum_i |x_i - \mu|
\]

You can assume for simplicity that \( n \) is odd.

(b) Show that the mean is the value of \( \mu \) that minimizes the function

\[
\sum_i (x_i - \mu)^2
\]

(c) Notice how the function for \( \mu_2 \) penalizes points that are far from \( \mu \) much more heavily
than the function for \( \mu_1 \). Thus \( \mu_2 \) tries much harder to be close to all the observations.
This might sound like a good thing at some level, but it is statistically undesirable because
just a few outliers can severely throw off the estimate of \( \mu_2 \). It is therefore sometimes said
that \( \mu_1 \) is a more robust estimator than \( \mu_2 \). Worse than either of them, however, is \( \mu_\infty \),
the value of \( \mu \) that minimizes the function

\[
\max_i |x_i - \mu|
\]

Show that \( \mu_\infty \) can be computed in \( O(n) \) time (assuming the numbers \( x_i \) are small enough
that basic arithmetic operations on them take unit time).

**Question 2 [30 points]** You are given a binary heap of size \( n \) (with the largest element
on top), represented as an array. Given a value \( x \) and a number \( k \), give an algorithm to
decide whether the value of the \( k \)th largest element of the heap is at most \( x \). The running
time should be \( O(k) \). Note that the algorithm *does not* need to explicitly find the value of
the \( k \)th largest element – all it needs is to decide if the value is at most \( x \).

The running time must depend only on \( k \) and *must* be independent of the size of the
heap \( n \), else you get no credit. The algorithm can use \( O(k) \) extra space. As an example, if
\( k = 1 \), the algorithm simply compares the first element of the array with \( x \), so the running
time is \( O(1) \).

**Question 3 [30 points]** (a) Starting from the definition of \( x \equiv y \mod N \) (namely that \( N \)
divides \( x - y \)) justify the substitution rules

\[
x \equiv x' \mod N, y \equiv y' \mod N \implies x + y \equiv x' + y' \mod N
\]
and
\[ x \equiv x' \mod N, \quad y \equiv y' \mod N \implies xy \equiv x'y' \mod N \]

(b) Show that if \( a \equiv b \mod N \) and if \( M \) divides \( N \) then \( a \equiv b \mod M \)