1. Prove the correctness of the Binary Search algorithm.

   Hint: As the algorithm is recursive, try giving a proof by induction, assuming that the algorithm works correctly for all input arrays of size at most k.

2. Give the Merge Sort algorithm and prove its correctness (including its subtasks). Do not forget to divide the algorithm into subtasks, where applicable.

3. Consider the following variant of Merge Sort, where $f(n) \in [2, n] \cap \mathbb{N}$ is a function (it can be a constant function) .

   $kMergeSort(A)$
   Input. An array A containing $n \geq 1$ elements and a $k \in [1, n]$.
   Output. The k^{th} smallest element in A; equivalently, the k^{th} element in $sorted(A)$.

   if $n \leq 10$ **then**
       return $InsertionSort(A)$
   end if
   Divide A into $k = f(n)$ sub-arrays $\{A_1, \cdots, A_k\}$, each having n/k elements
   for $i = 1$ to k **do**
       return $kMergeSort(A_i)$
   end for
   return $kWayMerge(A, k)$

   (a) Assuming that the sorting is done in-place, give the input-output specification for the *kWayMerge* function.

   (b) Give recurrence relations for the time complexity of *kMergeSort*, for $k = 3$ and $\sqrt{n}$.

   (c) For $k = \sqrt{n}$, use appropriate data structures to run *kMergeSort* in $\Theta(n \log n)$ time.

4. Suppose that $T(n) = T\left(\frac{c_2}{c_1} n\right) + T\left(\frac{c_3}{c_1} n\right) + \Theta(n)$, where c_1, c_2, c_3 are constants.

   (a) If $c_1 = c_2 + c_3$, show that $T(n) = \Theta(n \log n)$,

   (b) If $c_1 > c_2 + c_3$, show that $T(n) = \Theta(n)$.

5. Consider the following algorithm. (We use $A[p..q]$ to denote the sub-array of A containing all the elements from $A[p]$ to $A[q]$, for $1 \leq p \leq q \leq n$.)

   $QuickSelect(A, k)$
   Input. An array A containing $n \geq 1$ elements and an integer $k \in [1, n]$.
   Output. The k^{th} smallest element in A; equivalently, the k^{th} element in $sorted(A)$.

   Pick an arbitrary integer $p \in [1, n]$
   $x = Partition(A, p)$ ▷ x contains the position of $A[p]$ after partitioning
   if $x == k$ **then**
       return $A[x]$
   else if $x > k$ **then**
       return $QuickSelect(A[1..x-1], k)$
   else
       return $QuickSelect(A[x+1..n], k - x)$
   end if

   (a) Give the input-output specification as well as the algorithm for implementing the *Partition* function and prove its correctness. Note that you need not give the most sophisticated algorithm; a simple algorithm running in $\Theta(n)$ time would suffice.

   (b) Prove the correctness of *QuickSelect*.

   (c) Compute the best and worst case complexities of *QuickSelect*.

6. In both QuickSelect and QuickSort, the algorithms' complexity is highly dependent on the pivot chosen. Consider the *Median* problem specified below.

   $\underline{Median(A)}$
   Input. An array A containing $n \geq 1$ elements.
   Output. The median of A; equivalently, the $\left\lceil \frac{n}{2} \right\rceil^{th}$ element in $sorted(A)$.

   Suppose algorithm $Median(A)$ finds the median in $\Theta(n)$ time. Use this algorithm to devise a *simple* recursive algorithm to solve *Select* in $\Theta(n)$ time. The recurrence relation for your algorithm's complexity must match that of Binary Search.

   Note: Similarly, if a linear time algorithm for *Select* is given, we can trivially solve *Median* in linear time as well, by simply using its definition. This implies that the Select and Median problems are effectively equivalent.

7. The *kSelect* algorithm (seen in class; also known as the *median-of-medians* method) improves on QuickSelect by ensuring that the pivots picked are always *good* so that even in the worst case, it takes only $\Theta(n)$ time. We say that an element is a good pivot if its the i^{th} element in $sorted(A)$, for some $i \in \left[\frac{n}{c}, \frac{c-1}{c}n\right]$ and constant c. To find a good pivot, the *kSelect* algorithm first divided the array into $l > 1$ sub-arrays, each having n/l elements.

   (a) We say that an assumption is made *without loss of generality* when the assumption made does not restrict any of the inputs to the original problem. Here, we assumed that n is divisible by l. Was this assumption made without loss of generality? Justify your answer.

   (b) We also assumed that the elements are distinct. Was this assumption made without loss of generality?

   (c) By picking $l = 5$, we saw that the pivot found must be the i^{th} element of $sorted(A)$, for some $i \in \left[\frac{3}{10}, \frac{7}{10}n\right]$.

      i. Let p be the pivot found and assume that p is an element of the first sub-array of the original array. Clearly justify why this does not contradict our proof that there are at least $3n/10$ elements smaller, and $3n/10$ elements larger, than p.

      ii. For each $l \in [2, 7] \setminus \{5\}$, give the best bounds you can on the range of values in which i, the index of the pivot in the sorted array, must lie.

      iii. Recall the term in the recurrence relation corresponding to the recursive call made after finding the pivot. For each l in part (iii.), determine what the corresponding term will be.

      Note: Once the pivot is computed, your algorithm for Question 6 should be able to find the k^{th} smallest element. The only difference would be that, instead of the actual median, you would be using the median-of-medians.

   (d) Specify the termination condition(s) for the *kSelect* algorithm.

   (e) For each $l \in [2, 4]$, show that the algorithm has $\omega(n)$ worst case complexity. Also, for any constant $l \geq 5$, show that the algorithm runs in $\Theta(n)$ time.