1. Consider a rectangular array of distinct elements. Each row is given in increasing sorted order. Now sort (in place) the elements in each column into increasing order. Prove that the elements in each row remain sorted.

**Proof:** Let $A_k[i]$ be the $i$th element of the $k$th row in the initial array. Let $S_k[i]$ be the $i$th element of the $k$th row in the array after it is sorted.

**Claim 1** For all $i, j, k \in [1, n]$, $i < j$ implies that $S_k[i] \leq S_k[j]$.

**Proof of claim:** Let $C_i$ be the set of elements in the $i$th column, that is $C_i = \{A_k[i]|0 \leq k \leq n\}$. The claim will follow if we can show that at least $k$ elements are less than $S_k[j]$ in $C_i$. To see this, notice that the $k$th smallest element of $C_i$ will end up as $S_k[j]$.

Now, consider the elements $S_1[j], S_2[j], \ldots, S_k[j]$. Each such $S_l[j]$ started out as some $A_l[j]$. But $A_l[i] < A_l[j] < S_k[j]$. Therefore, we have shown that there are at least $k$ elements in $C_i$ which are no greater than $A_k[j]$ and we have therefore established the claim. □

But this claim is the same as saying that each row is sorted. □

2. Suppose that you are given a $k$-sorted array, in which no element is farther than $k$ positions away from its final (sorted) position. Give an algorithm which will sort such an array. Prove its correctness. Analyse its running time. Note: your algorithm should run faster than $\Theta(n \log n)$, that is, it should take advantage of the fact that the array is $k$-sorted.

**Answer:** The algorithm is as follows:

```
\textit{k-sort} (A)

Divide input array $A$ into blocks of size $k$: $B_i = A[(i-1)k+1] \cdots A[ik]$. 
for $i = 1$ to $n/k$ do
\textbf{Mergesort}(B_i, B_{i+1})
end
```

**Running time:** Each call to \textit{mergesort} is on an array of size $2k$, so it takes $\Theta(k \log k)$. Since $n/k$ such calls are made, the total time is $\Theta(n \log k)$.

**Correctness:**

**Claim 2** After \textit{mergesort}(B_i, B_{i+1}) has executed, blocks $B_1, \ldots, B_i$ are correctly sorted and blocks $B_{i+1}, \ldots, B_{n/k}$ are $k$-sorted.
Proof of claim:  Base case: For $i = 0$, that is before we call `mergesort` at all, the claim is trivially true.

Inductive hypothesis: Suppose the claim is true for $i' < i$. By induction, $B_1, \ldots, B_{i-1}$ are correctly sorted. After calling `mergesort`(\(B_i, B_{i+1}\)), $B_{i+2}, \ldots, B_{n/k}$ are still $k$-sorted. So we need to show that $B_i$ is sorted and $B_{i+1}$ is still $k$-sorted.

$B_i$ is sorted because the elements that end up in $B_i$ are the $k$ smallest elements of $B_i$ and $B_{i+1}$. `mergesort`(\(B_i, B_{i+1}\)) will correctly put the $k$ smallest elements in their place.

Now, suppose that $B_{i+1}$ is no longer $k$-sorted. Let $j$ be the index of the rightmost element of $B_{i+1}$ that is more than $k$ away from its correct position. First, $j$ cannot be the last position, since each element in $B_{i+1}$ must end up in either $B_{i+1}$ or $B_{i+2}$ (and all these positions are within $k$ of the last position). Now if there are $l$ things greater than $A[j]$ in $B_{i+1}$, then $A[j]$’s final position cannot be within $l$ of the right-hand side of $B_{i+2}$. But then its final position is no more than $k$ away, so $A[j]$ is within $k$ of its final position, so $B_{i+1}$ is $k$-sorted.

We conclude by noting that after calling `mergesort`(\(B_{n/k-1}, B_{n/k}\)). We have that $B_1, \ldots, B_{n/k-1}$ is sorted and $B_{n/k}$ is $k$-sorted. But clearly $B_{n/k}$ is also sorted, giving that the entire array is sorted. □

3. Consider once again the $k$-sorted array of the previous problem. Show that any comparison based algorithm for sorting an almost sorted array makes $\Omega(n \log k)$ comparisons.

Proof:  We must count the number of distinct $k$-sorted arrays. We will give a lower bound for this number. We will count all the $k$-sorted arrays so that elements need not cross blocks (as defined in algorithm above). We call all such arrays block $k$-sorted. To recap, in these arrays, when going from a block $k$-sorted array to a sorted array, the correct position for all elements in block $B_i$ is still within block $B_i$.

Now, within each block, any ordering of the elements is possible, so there are $k!$ ways to order each block. That means that the number of distinct block $k$-sorted arrays is $(k!)^{n/k}$. So the number of $k$-sorted arrays is at least $(k!)^{n/k}$.

So a decision tree for this problem has depth $\Omega(\log(k!)^{n/k}) = \Omega((n/k) \log(k!)) = \Omega((n/k)k \log k) = \Omega(n \log k)$. □