Problem 21-1

(a)
The values in the *extracted* array would be [4, 3, 2, 6, 8, 1].

The following table shows the situation after the *i*th iteration of the **for** loop when we use OFF-LINE-MINIMUM. Because $j = m + 1$ in the iterations for $i = 5$ and $i = 7$, no changes occur in these iterations. (For this input, $n = 9$ and m, the number of extractions, is 6).

i	K_1	K_2	K_3	K_4	K_5	K_6	K_7	*extracted*					
								1	2	3	4	5	6
0	{4, 8}	{3}	{9, 2, 6}	{}	{}	{1, 7}	{5}						
1	{4, 8}	{3}	{9, 2, 6}	{}	{}		{5, 1, 7}						1
2	{4, 8}	{3}		{9, 2, 6}	{}		{5, 1, 7}			2			1
3	{4, 8}			{9, 2, 6, 3}	{}		{5, 1, 7}		3	2			1
4				{9, 2, 6, 3, 4, 8}	{}		{5, 1, 7}	4	3	2			1
5				{9, 2, 6, 3, 4, 8}	{}		{5, 1, 7}	4	3	2			1
6					{9, 2, 6, 3, 4, 8}		{5, 1, 7}	4	3	2	6		1
7					{9, 2, 6, 3, 4, 8}		{5, 1, 7}	4	3	2	6		1
8							{5, 1, 7, 9, 2, 6, 3, 4, 8}	4	3	2	6	8	1

(b)
We want to show that the array *extracted* returned by OFF-LINE-MINIMUM is correct, meaning that for $i = 1, 2, \ldots, m$, *extracted*[j] is the key returned by the jth EXTRACT-MIN call. We start with n INSERT operations and m EXTRACT-MIN operations. The smallest of all the elements will be extracted in the first EXTRACT-MIN after its insertion. So we find j such that the minimum element is in K_j , and put the minimum element in *extracted*[j], which corresponds to the EXTRACT-MIN after the minimum element insertion.

Now we reduce to a similar problem with $n - 1$ INSERT operations and $m - 1$ EXTRACT-MIN operations in the following way: the INSERT operations are the same but without the insertion of the smallest that was extracted, and the EXTRACT-MIN operations are the same but without the extraction that extracted the smallest element.

Conceptually, we unite I_j and I_{j+1}, removing the extraction between them and also removing the insertion of the minimum element from $I_j \cup I_{j+1}$. Uniting I_j and I_{j+1} is accomplished by line 6. We need to determine which set is K_l, rather than just using K_{j+1} unconditionally, because K_{j+1} may have been destroyed when it was united into a higher-indexed set by a previous execution of line 6.

Because we process extractions in increasing order of the minimum value found, the remaining iterations of the **for** loop correspond to solving the reduced problem. There are two other points worth making. First, if the smallest remaining element had been inserted after the last EXTRACT-MIN (i.e., $j = m + 1$), then no changes occur, because this element is not extracted. Second, there may be smaller elements within the K_j sets than

the one we are currently looking for. These elements do not affect the result, because they correspond to elements that were already extracted, and their effect on the algorithm's execution is over.

(c)
To implement this algorithm, we place each element in a disjoint-set forest. Each root has a pointer to its K_i set, and each K_i set has a pointer to the root of the tree representing it. All the valid sets K_i are in a linked list. Before OFF-LINE-MINIMUM, there is initialization that builds the initial sets K_i according to the I_i sequences.

• Line 2 ("determine j such that $i \in K_j$ ") turns into $j \leftarrow$ FIND-SET(i).
• Line 5 ("let l be the smallest value greater than j for which set K_l exists") turns into $K_l \leftarrow next[K_j]$.
• Line 6 ("$K_l \leftarrow K_j \cup K_l$, destroying K_j") turns into $l \leftarrow$ LINK(j, l) and remove K_j from the linked list.

To analyze the running time, we note that there are n elements and that we have the following disjoint-set operations:

• n MAKE-SET operations
• at most $n - 1$ UNION operations before starting
• n FIND-SET operations
• at most n LINK operations

Thus the number m of overall operations is $O(n)$. The total running time is $O(m\ \alpha(n)) = O(n\ \alpha(n))$.