Problem 1. The peak demand for the resource occurs at a time that is covered by the largest number of intervals, a number that can be computed in $O(n \log n)$ time as follows.

1. Initialize a counter variable *count* to be 0.
2. Sort the $2n$ endpoints and go through the sorted list: If the endpoint encountered is an s_i (a start time) then increment *count*, if it is an f_i then decrement *count*. While doing this, keep track in a variable *max_count* of the largest value achieved by *count*.
3. Return *max_count* as the answer.

Problem 2. Create B = a sorted version of A (by increasing values). Then give A and B as input to the LCS software: The returned LCS is an LIS of A; it is increasing because B itself is sorted, and it is of maximum possible length because an LCS has (by definition) maximum length.

Question 3. The table is:

0	48	352	1020	1096
	0	114	792	978
		0	684	936
			0	2394
				0

Question 4. Let $S_i = \{I_1, \ldots, I_i\}$, i.e., S_i is the subset of S none of whose intervals extends to the right of r_i. Let C_i be a maximum-weight acceptable subset of S_i that contains interval I_i; in other words the intervals of C_i have maximum total weight subject to the constraints of being nonoverlapping and including I_i. Note that if we had all of $\{C_1, C_2, \ldots, C_n\}$ then we could compute the answer in linear time by simply choosing the largest of the C_is. Hence it suffices to compute all the C_is.
Observe that $C_1 = w_1$ and that, for $i > 1$, we have

$$C_i = w_i + \max_{k: r_k < l_i} C_k$$

which immediately implies an $O(n^2)$ time algorithm for computing all the C_is (by computing each of $C_1, C_2, \ldots, C_n$ in that order, according to the above equation).

Question 5.

1. 3 workers and 3 tasks, and $S = (W_1, T_1), (W_1, T_2), (W_2, T_1)$. The algorithm produces $M = 1$ because it assigns W_1 to T_1 and leaves W_2 and T_2 unassigned, whereas an optimal solution achieves $Opt = 2$ by assigning W_1 to T_2 and W_2 to T_1.

2. When the algorithm assigns W_i to T_j, this can prevent a better solution from assigning W_i to another task, and from assigning another worker to T_j, so the total damage from including the W_i to T_j assignment is the loss of two other assignments: one assigment by the algoritm prevents at most 2 other assignemts, a ratio of 1:2. This implies a ratio of at most 2 between the size of an optimal solution and the size of what the algorithm produces.