1. **Solution:** This problem asked for a run of Dijkstra’s algorithm on the below graph.

![Graph](image)

In particular, we are asked to find the order in which the nodes are discovered, and the distance $d(s, v)$ to each node $v$. We get the following.

<table>
<thead>
<tr>
<th>order</th>
<th>vertex</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertex</td>
<td>S</td>
<td>A</td>
<td>C</td>
<td>B</td>
<td>F</td>
<td>G</td>
<td>D</td>
<td>H</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>distance</td>
<td>0</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>13</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

Note that in Dijkstra’s, we may visit $D$ and $H$ in either order, depending on how we break ties.

**Explanation:** See Figure 1 for a complete run of the algorithm. Note that in that figure, blue nodes are in $R$; red nodes are not in $R$ but a node in $R$ has an edge to them; and light grey nodes are outside $R$ and have no edge to them from any node in $R$. Blue edges are those used in shortest paths; red edges are those which cross from inside $R$ to outside $R$; light grey edges lie entirely outside $R$; and dark grey edges lie inside $R$ but are not used in any shortest paths.

2. (a) **Solution:** Earliest end first and latest start first yield optimal solutions.

**Explanation:** Note that the problem is fundamentally symmetric – if we pick some very large number $N$, and replace every interval $[s, f]$ by $[N - f, N - s]$ we end up with exactly the same problem, but reversed horizontally (with respect to time, that is). Thus, the strategies latest start first and latest end first are equivalent to earliest end first and earliest start first, respectively. As discussed in class and in the notes, shortest job first, fewest conflicts first, and earliest start first all fail to produce optimal schedules on some instances; earliest end first, however, always produces an optimal schedule.

(b) **Solution:** Prim’s and Kruskal’s will sometimes, but not always, add edges to the MST in the same order.

**Explanation:** For an example, consider the below graph.

![Graph](image)

In the above graph, the (unique) MST consists of the edges $\{A, B\}$ and $\{B, C\}$, and Kruskal’s will add them to the MST in that order. If we start Prim’s from either $A$ or $B$, it too will add these edges in that order; if we start Prim’s from $C$, however, it will add the edges in the reverse order of $\{B, C\}$ and then $\{A, B\}$. 
(c) **Solution:** Gets ahead/stays ahead and exchange arguments.

**Explanation:** We saw three main greedy proof techniques when talking about scheduling. Any two of: gets ahead/stays ahead; exchange argument; and structural properties would be an acceptable answer.

(d) **Solution:** The companies can only sometimes force a particular matching; their ability to do this depends on the particular preferences of the applicants. The applicants, however, can always force any agreed upon matching to occur.

To see this, first consider the applicants. If they want to force a particular matching, each applicant should just put they company they agreed to be matched with as their top preference. Since all applicants' top preferences are distinct, no company ever sees two different applicants, and so no rejections can or will occur.

Now, consider the companies. If the applicant’s preferences happen to be as above (all distinct), then there is nothing the companies can do to change the outcome. On the other hand, in some cases the companies can force any matching they like. For example, if all of the applicants have the same preference list, then the companies may use the same strategy outlined above to achieve the agreed upon matching.

3. **Solution:** Let $G$ denote the described graph which has the set $T$ (the set of spanning trees of $G$) as vertices. We prove that $G$ is always connected. Let $T$ and $T'$ be any two distinct spanning trees of $G$. We will show that $G$ contains a path between them. Let $k$ be the number of edges of $G$ that are present in $T$ but not in $T'$; note that this must also be the number of edges in $T'$ but not in $T$. We will iteratively construct a sequence $T = T_0, T_1, T_2, \ldots, T_k = T'$ of $k$ trees such that every pair $\{T_i, T_{i+1}\}$ is an edge in $G$. We begin by setting $T_0 = T$. Now, given $T_i$, if $T_i$ does not equal $T'$ we construct $T_{i+1}$ as follows. Let $e$ be any edge that is present in $T'$ but not in $T_i$. Consider adding $e$ to $T_i$; since it isn’t originally in $T_i$, it will create a cycle. Furthermore, since $e$ is in $T'$, and $T'$ is a spanning tree, we know that at least one edge in this created cycle must not appear in $T'$. Let $e'$ be such an edge. Now, since $e'$ is on the cycle created by $e$, we may remove it without breaking connectivity, and hence arrive back at a spanning tree. We set $T_{i+1}$ to this new tree:

$$T_{i+1} = T_i \backslash \{e'\} \cup \{e\}.$$

Note that $T_i$ and $T_{i+1}$ each only have one edge the other does not, and so $\{T_i, T_{i+1}\}$ is an edge in $G$. Finally, observe that $T_{i+1}$ has one more edge in common with $T'$ than $T_i$ does. Thus, after $k$ steps of the above process, we will produce a tree $T_k$ which is exactly $T'$. We therefore conclude that $G$ contains a path from $T$ to $T'$. Since our choice of spanning trees was arbitrary, it follows that $G$ contains a path between any pair of spanning trees (vertices), i.e. it is connected.

**Explanation:** The description of this problem was a bit complex; to make this concrete, consider the below example of a particular graph $G$ and the graph $G$ we define on it’s spanning trees; $G$ is on the left and $G$ is on the right. The nodes of $G$ indicate the spanning tree they correspond to pictorially, with included edges shown as in the orientation of $G$ on the left.
The key idea in this problem is recognizing that it closely corresponds to how we made exchange arguments when discussing MSTs; specifically, the cut property for MSTs and proving the optimality of Kruskal’s algorithm. Note that an edge in $G$ corresponds exactly to the sort of exchange operation we performed: we added an edge to a spanning tree to create a cycle, and then removed a different edge from that cycle to get back to a spanning tree. Thus, the fact that the graph is connected is quite important to our proof of optimality for Kruskal’s. In that proof, when trying to convert an arbitrary optimal tree to the greedy one, we did not need any structural properties of optimal spanning trees; this was exactly because we can always find this sort of path between two spanning trees. The construction of Kruskal’s insures that there is always a path between it and any other spanning tree where costs are decreasing along the path in the direction toward the Kruskal’s spanning tree.

4. For this problem, we need to move boxes from one location to another; each box $i$ has a weight $w_i$, and we move them using a cart that can support at most $W$ weight total at a time. We want to minimize the number of trips needed with the cart to move all the boxes, subject to the constraint that if $i < j$, box $j$ cannot be moved on a (strictly) earlier trip with the cart than box $i$. The problem asks us to show that the greedy strategy of always putting as many boxes on the cart as we can (subject to the previously mentioned constraint) produces the smallest possible number of trips. Observe that if at any point in time we have moved $k$ boxes, they must have been the boxes with indices 1 through $k$. This suggests a natural measure for a gets ahead/stays ahead argument: the number of boxes moved during the first $t$ trips with the cart. To make this formal, let $g(1), g(2), \ldots, g(m)$ be the number of boxes after 1, 2, $\ldots$, $m$ trips with the cart under the greedy solution, and $o(1), o(2), \ldots, o(\ell)$ be the number of boxes after 1, 2, $\ldots$, $\ell$ trips with the cart under some optimal solution. We assume $o(\ell) = n = g(m)$, in other words the optimal and greedy solutions use $\ell$ and $m$ cart trips total, respectively. Note that we must have $m \geq \ell$. We prove the following claim:

**Claim 1.** For all $t \leq \ell$, $g(t) \geq o(t)$.

**Proof.** We prove this by induction on $t$. For $t = 1$, the greedy algorithm places the largest number of boxes that can fit on the cart, and so this must hold. So assume it holds for $t - 1$; we will show that it must hold for $t$ as well. Now, we know $o(t - 1) \leq g(t - 1)$. If $o(t) \leq g(t - 1)$ as well, we are already done – no matter how few boxes the greedy algorithm takes on the $t^{th}$ cart trip, we will have that $g(t) \geq g(t - 1) \geq o(t)$, exactly as desired. So we assume that $o(t) > g(t - 1)$, and consider the boxes taken by the optimal solution on the $t^{th}$ cart trip. We must have that

$$W \geq \sum_{i=o(t)-1}^{o(t)} w_i = \sum_{i=o(t)-1}^{g(t-1)} w_i + \sum_{i=g(t-1)}^{o(t)} w_i \geq \sum_{i=g(t-1)}^{o(t)-1} w_i.$$

Thus, we can see that it is feasible for the greedy algorithm to carry boxes $g(t-1)+1, g(t-1)+2, \ldots, o(t)$ in a single cart trip. Since the greedy algorithm always takes as many boxes on each cart trip as it feasibly can, we may conclude that $g(t) \geq o(t)$. \hfill \Box

From the above, we immediately get the optimality of the greedy solution: it implies that $g(\ell) \geq o(\ell) = n$, and so the greedy algorithm will have moved all of the boxes after $\ell$ trips, the fewest trips any algorithm can use (since the $o(\cdot)$ values correspond to a greedy solution).
Figure 1: Run of Dijkstra’s algorithm for problem 1