Unlike divide-and-conquer and dynamic programming algorithms, greedy algorithms DO NOT divide a problem into smaller subproblems. Instead, a greedy algorithm builds a solution by making a sequence of choices that look best ("locally" optimal) at the moment without regard for past or future choices (no fixing of previously bad choices). The generalized outline of a greedy algorithm is:

Repeatedly:
1. Make the next (irrevocable) choice based on some locally optimal greedy criterion
2. Reject this choice and continue if it is not feasible for this choice to lead to a solution
3. If this choice resulted in a solution, break

1. Recall the coin-change problem: Given a set of coin types (e.g., \{1, 5, 10, 25, 50\}) and an amount of change (e.g., 29-cents) to be returned, determine the fewest number of coins for this amount of change.
   a) What sequence of choices need to be made to solve this problem?

b) What locally optimal greedy criterion could you use to make the next choice?

c) When would we reject a choice as infeasible?

d) How would we know when we have reached a solution?

e) Do you get a "globally optimal" solution if you use this algorithm for coin types of \{1, 5, 10, 12, 25, 50\} and a change amount of 29-cents?

2. Suppose you had a map of settlements on the planet X

```
  a          b
   \       / \   \   \   \  
  h   i   e   f   d
   \       /     /     \\
    g       e
```

We want to build tunnels that allow us to travel between any pair of cities. Because resources are scarce, we want the total length of all tunnels built to be minimal. What would be some characteristics of the resulting "graph" after all the cities are connected?
Prim’s algorithm for determining the minimum-spanning tree (MST) of an undirected graph $G = (V, E)$ is an example of a greedy algorithm. The general idea of Prim’s algorithm is:

Starts with any vertex, say $v_1$, in the initial, partial MST $= (Y, F)$ with $Y = \{v_1\}$ and $F = \{\}$ (i.e., no edges yet).

Repeatedly grow the MST by adding one vertex at a time until it contains all the vertices in $V$

1. Choose a vertex not in the partial MST (from $V - Y$) that is closest to some vertex in the MST
2. Add the chosen vertex to the partial MST, i.e., add it to the set $Y$
3. Add the edge that connected the chosen vertex to the MST to $F$

a) Trace the above algorithm on the below graph to find the MST.

b) What data structure could be used to efficiently determine that selection?