1. (problem 1) This algorithm does not solve the problem of finding a maximum cardinality set of non-overlapping intervals. Consider the following intervals:

```
  A   B   C   D
  E
```

Obviously, the optimal solution is \{A, B, C, D\}. However, the interval that overlaps with the fewest others is E, and the algorithm will select E first, which precludes it from picking intervals B and C.

2. (problem 2)

(a) This algorithm does not solve the interval-coloring problem. Consider the following intervals:

```
  A
  B   C   D
  E   F   G
```

The optimal solution is to put A in one room, \{B, C, D\} in another, and \{E, F, G\} in another, for a total of 3 rooms. However, maximizing the number of classes in the first room results in having \{B, C, F, G\} in one room, and classes A, D, and G each in their own rooms, for a total of 4.

(b) This algorithm does solve the interval-coloring problem. Note that if the greedy algorithm creates a new room for the current class \(c_i\), then because it examines classes in order of start times, \(c_i\)'s start point must intersect with the last class in all of the current rooms. Thus when greedy creates the last room, \(N\), it is because the start time of the current class intersects with \(N - 1\) other classes. But we know that for any single point in any class it can only intersect with at most \(s\) other class, it must be then that \(N \leq s\). As \(s\) is a lower bound on the total number needed and greedy is feasible it is thus also optimal.

3. (problem 3)

(a) The greedy algorithm is not optimal for the problem of making change with the minimum number of coins when the denominations are 1, 5, 10, 20, 25, and 50. In order to make 40 Shillings, the greedy
algorithm would use three coins of 25, 10, and 5 shillings. The optimal solution is to use two 20-shilling coins.

(b) The greedy algorithm is optimal for the problem of making change with the minimum number of coins when the denominations are successive powers of some integer $p$. We prove this by contradiction. Assume there is some integer $D$ such that the greedy algorithm is not optimal for the set of denominations $D^0, D^1, \ldots, D^m$ when making change for $x$. For $i = 0, \ldots, n$, let $g_i$ denote the number of coins of denomination $D^i$ picked by the greedy algorithm, and let $t_i$ denote the number of denomination $D^i$ picked in an optimal solution. We have:

$$\sum_{i=0}^{n} g_i D^i = \sum_{i=0}^{n} t_i D^i = x.$$ 

Starting from $n$, let $k$ be the first index for which $g_k \neq t_k$. The greedy algorithm takes as many coins worth $D^k$ as it can, so it must be true that $g_k > t_k$. Furthermore, since both solutions have the same total value, $x$, the total value of $t_k - 1, t_k - 2, \ldots, t_0$ must “make up” for the lost value of (at least) one $D^k$. This can never happen. Note that

$$\sum_{i=0}^{k-1} t_i D^i \leq \sum_{i=0}^{k-1} (D - 1)D^i = \sum_{i=0}^{k-1} (D^{i+1} - D^i) = D^k - D^0 = D^k - 1$$

This says that if $t_0 D^0 + \cdots + t_m D^m = x$, then one of the $t_i$ for $i = 0, \ldots, k-1$ must be greater than $D-1$. However, if there are ever more than $D - 1$ of any denomination of coin (except the largest), then $D$ of those coins can be replaced by one coin of the next largest denomination. This contradicts our assumption that the $t_i$’s form an optimal solution, since if any $t_i$ differs from $g_i$, then there must be at least $D$ coins of some smaller denomination among the $t_i$’s.

4. (problem 4)

(a) This greedy algorithm is optimal. We prove by contradiction. Assume greedy is not optimal for input $I$, we pick the optimal solution, OPT, that is identical to greedy for the most consecutive gas stations. Consider the first gas station where the greedy solution, $G$, and OPT differ, call it station $k$. Say $G$ adds $g_k$ gas and OPT adds $o_k$ gas. We now create a new solution, OPT’ as follows: OPT’ is identical to OPT at every station except $k$ and $k+1$. Call the amount of gas OPT adds at station $k+1$, $o_{k+1}$. At station $k$, OPT’ only adds $g_k$ gas to the tank, and at station $k+1$, OPT’ adds $o_{k+1} + (o_k - g_k)$. Clearly, OPT’ is identical to $G$ for one more station, namely $k$. We claim that OPT’ is feasible and spends no more time filling the tank than OPT. Prior to station $k+1$, OPT’ is identical to $G$ thus, because $G$ makes it to $k+1$, OPT’ must make it to $k+1$. By the fact
that greedy adds the minimal amount of gas required to get from \( k \) to \( k + 1 \), and \( G \) and \( \text{OPT} \) differ at \( k \), it must be that \( o_k > g_k \), thus \( o_{k+1} + (o_k - g_k) > 0 \) meaning \( \text{OPT}' \) adds a valid amount of gas at \( k + 1 \). Further, because \( g_k + (o_{k+1} + (o_k - g_k)) = o_k + o_{k+1} \), \( \text{OPT}' \) has the same amount of gas in the tank as \( \text{OPT} \) after filing up at \( k + 1 \), namely \( o_k + o_{k+1} - g_k \). Because \( \text{OPT}' \) is identical to \( \text{OPT} \) after \( k + 1 \), \( \text{OPT}' \) never runs out of gas after \( k + 1 \). Finally, because the total gas put in the tank by \( \text{OPT}' \), over \( k \) and \( k + 1 \), is \( g_k + (o_k + 1 + (o_k - g_k)) = o_k + o_{k+1} \), \( \text{OPT} \) and \( \text{OPT}' \) add the same amount of gas in total over the two stations in which they differ, making their total time spent filling the same. Thus we have an optimal solution that is identical to greedy for one more station, a contradiction.

(b) This greedy algorithm is not optimal. Without loss of generality we can assume the car starts at \( A \) with an empty tank. Consider the input of \( x_1 = 0, x_2 = 5, x_3 = 6 \), further, assume that \( c, F, \) and \( r \) are such that a full tank of gas takes you 5km. The greedy algorithm will fill the tank twice but filling the tank only at \( x_1 \) then adding just enough at \( x_2 \) to go 1km will give a lower total time filling up.

5. (problem 5)

(a) This algorithm is not optimal for the problem of covering points with unit intervals. Let the points to be covered be \( A = -\frac{1}{3}, B = \frac{1}{3}, C = \frac{1}{3}, D = 2 \), \( E = \frac{2}{3}, \) and \( F = \frac{3}{2} \). The algorithm that tries to maximize the number of points covered by the first interval will cover \( B, C, D, E \) with the first interval, which forces it to use at least 3 intervals total. The points can, however, be covered with two intervals, \([ -\frac{1}{3}, \frac{1}{3} ]\), and \([ \frac{2}{3}, \frac{3}{2} ]\).

(b) This algorithm is optimal for the problem of covering points with unit intervals. Assume there is a set of points \( A = \{ a_1, \ldots, a_n \} \) such that the solution obtained by the greedy algorithm is not optimal. Call the greedy solution \( G = \{ g_1, \ldots, g_n \} \) and the optimal solution \( T = \{ t_1, \ldots, t_n \} \). Assume the intervals are numbered in increasing order of left endpoint. Starting at the leftmost interval in \( G \), compare \( G \) and \( T \). Let \( k \) be the number of the first interval for which \( g_k \neq t_k \). By the definition of the greedy algorithm, it must be the case that \( g_k > t_k \) (meaning that \( g_k \) begins further to the right than \( t_k \)). Create solution \( T' \) by replacing interval \( t_k \) with \( g_k \). Since for \( i = 1, \ldots, k-1 \), \( g_i = t_i \), solution \( T \) will continue to cover all the points in \( A \). If \( g_k \) overlaps any other interval \( t_j \) in \( T \), shift \( t_j \) to the right until it no longer overlaps \( g_k \). Continue shifting intervals in \( T' \) to the right until there are no more overlaps. Note that \( T' \) continues to cover all points in \( A \). By repeating the above process, we can make \( T = G \), contradicting our assumption that \( G \) is not an optimal solution.
6. (problem 9)

(a) This algorithm is incorrect for the problem of minimizing the average difference between the heights of skiers and their skis. Let \( p_1 = 5, \quad p_2 = 10, \quad s_1 = 9, \quad \text{and} \quad s_2 = 14. \) The algorithm would pair \( p_1 \) with \( s_2 \) and \( p_2 \) with \( s_1 \) for a total cost of \( \frac{1}{2}(1 + 9) = 5 \). Pairing \( p_1 \) with \( s_1 \) and \( p_2 \) with \( s_2 \) yields a total cost of \( \frac{1}{2}(4 + 4) = 4 \).

(b) The algorithm is correct for the problem of minimizing the average difference between the heights of skiers and their skis. The proof is by contradiction. Assume the people and skis are numbered in increasing order by height. If the greedy algorithm is not optimal, then there is some input \( p_1, \ldots, p_n, \ s_1, \ldots, s_n \) for which it does not produce an optimal solution. Let the optimal solution be \( T = \{(p_{1,n}, s_{n(1)}), \ldots, (p_{n,n}, s_{n(n)})\} \), and note the output of the greedy algorithm will be \( G = \{(p_1, s_1), \ldots, (p_n, s_n)\} \). Beginning with \( p_1 \), compare \( T \) and \( G \). Let \( p_i \) be the first person who is assigned different skis in \( G \) than in \( T \). Let \( s_j \) be the pair of skis assigned to \( p_i \) in \( T \). Create solution \( T' \) by switching the ski assignments of \( p_i \) and \( p_k \), where \( p_k \) is the person who was assigned \( s_i \) in \( T \). Note that by the definition of the greedy algorithm, \( s_i \leq s_j \). Also note that by def of \( p_i, \ p_i \leq p_k \). The total cost of \( T' \) is given by

\[
Cost(T') = Cost(T) - \frac{1}{n}(|p_i - s_j| + |p_k - s_i| - |p_i - s_i| - |p_k - s_j|)
\]

There are six cases to be considered. For each case, one needs to show that \(|p_i - s_j| + |p_k - s_i| - |p_i - s_i| - |p_k - s_j| \geq 0\).

Case 1: \( p_i \leq p_k \leq s_i \leq s_j \).

\[
|p_i - s_j| + |p_k - s_i| - |p_i - s_i| - |p_k - s_j| = (s_j - p_i) + (s_i - p_k) - (s_i - p_i) - (s_j - p_k) = 0
\]

Case 2: \( p_i \leq s_i \leq p_k \leq s_j \).

\[
|p_i - s_j| + |p_k - s_i| - |p_i - s_i| - |p_k - s_j| = (s_j - p_i) + (p_k - s_i) - (s_i - p_i) - (s_j - p_k) = 2(p_k - s_i) \geq 0
\]

Case 3: \( p_i \leq s_i \leq s_j \leq p_k \).

\[
|p_i - s_j| + |p_k - s_i| - |p_i - s_i| - |p_k - s_j| = (s_j - p_i) + (p_k - s_i) - (s_i - p_i) - (p_k - s_j) = 2(s_j - s_i) \geq 0
\]
Case 4: \( s_i \leq s_j \leq p_i \leq p_k \).
\[
|p_i - s_j| + |p_k - s_i| - |p_i - s_i| - |p_k - s_j| = (p_i - s_j) + (p_k - s_i) - (p_i - s_i) - (p_k - s_j) = 0
\]

Case 5: \( s_i \leq p_i \leq s_j \leq p_k \).
\[
|p_i - s_j| + |p_k - s_i| - |p_i - s_i| - |p_k - s_j| = (s_j - p_i) + (p_k - s_i) - (p_i - s_i) - (p_k - s_j) = 2(s_j - p_i) \geq 0
\]

Case 6: \( s_i \leq p_i \leq p_k \leq s_j \).
\[
|p_i - s_j| + |p_k - s_i| - |p_i - s_i| - |p_k - s_j| = (s_j - p_i) + (p_k - s_i) - (p_i - s_i) - (s_j - p_k) = 2(p_k - p_i) \geq 0
\]

7. (Problem 12) The algorithm is correct for the problem of building an \( n \times n \) matrix with zeros and ones such that the sum of all ones in the \( i \)th row is \( r_i \) and the sum of all ones in the \( i \)th column is \( c_i \) for all \( 1 \leq i \leq n \). The proof is by contradiction. Assume there is some input \( \{r_1, \ldots, r_n\}, \{c_1, \ldots, c_n\} \) for which the greedy algorithm does not give the correct solution. Call any correct matrix \( T \) and the matrix generated by the greedy algorithm \( G \). Let \( i \) and \( j \) be two numbers such that \( g_{ij} \neq t_{ij} \). Let \( g_{ij} = 1 \); this implies that \( t_{ij} = 0 \). By the definition of the problem, there must be a number \( k \neq j \) such that \( g_{ik} = 0 \) and \( t_{ik} = 1 \). Create matrix \( T' \) by making \( t_{ij} = g_{ij} \). \( T' \) is not a feasible solution; column \( j \) has too many ones and column \( k \) has too few. Since the greedy algorithm placed a 1 in \( g_{ij} \) and a 0 in \( g_{ik} \), it must be true that \( c_j \geq c_k \). Therefore, the number of ones in column \( k \) of \( T' \) is at most \( c - 1 \) and the number in column \( j \) is exactly \( c_j + 1 \). There must be at least one number \( l \neq i \) such that \( g_{lj} = 0, t_{lj} = 1, g_{lk} = 1, \) and \( t_{lk} = 0 \). Create a new matrix \( T'' \) by making \( g_{lj} = t'_{lj} \) and \( g_{lk} = t'_{lk} \). Columns \( j \) and \( k \) now have the correct number of ones. Matrix \( T'' \) is now a feasible solution that is closer to \( G \) that \( T \). Contradiction.

The case where \( g_{ij} = 0 \) and \( t_{ij} = 1 \) is nearly identical.

8. (Problem 7) The following greedy algorithm selects the optimal output for all inputs: At each step in the sequence where a page in fast memory needs to be replaced by one in slow memory, replace the page whose next use is at the latest point in the sequence.
Proof: Let $G$ be the greedy algorithm described by the theorem. Suppose $G$ is non-optimal for some input sequence.

Let $Opt$ be the optimal algorithm (having the fewest number of swaps) that agrees the most with $G$.

Let $k$ be the first swap where $Opt$ and $G$ disagree on which page to swap into fast memory. Let’s label the page swapped out of fast memory by $G$ at step $k$ as $A$, the page swapped out of fast memory by $Opt$ as $B$, and the page that needs to be swapped in by both as $C$.

We know that, by definition of the greedy algorithm, $B$ will appear sooner than $A$ after step $k$. Let’s call the step where $B$ next occurs step $i$, and the step where $A$ next occurs step $j$, where $i$ occurs before $j$. Let’s call the steps between step $k$ and step $i$ region $x$, and the steps between step $k$ and step $j$ region $y$, where $y$ includes step $i$ and region $x$ (Ideally there should be a picture illustrating these definitions.).

Suppose that there exists a solution $Opt'$ that is identical to $Opt$, except for at step $k$, $Opt$ makes the same decision as $G$ and selects $A$ to be replaced. Is $Opt'$ still an optimal solution? In order for $Opt'$ to be optimal it must have the same number of swaps as $Opt$, meaning that the change of the decision at $k$ did not affect the number of swaps that had to occur for the algorithm to be feasible.

After step $k$, the fast memory of $Opt$ contains at least pages $A$ and $C$, having swapped out page $B$ (we assume the problem applies to fast memories of size 2 or greater, since a 1-page fast memory would only have one feasible solution of swapping at every non-repeated page). This means that $Opt$ makes at least 1 swap by step $i$ where $B$ needs to be swapped into fast memory. $Opt'$ instead must swap by step $j$ where $A$ needs to be swapped into fast memory. It must be the case (to meet requirements for optimality) that no extra swaps were incurred by the decision of $Opt$ to replace $B$ at step $k$.

At step $k$, the fast memories of $Opt$ and $Opt'$ are identical, except for the page which contains either $A$ or $B$. With this information, we might try to infer that any swaps not involving $B$ that $Opt$ needs to make in region $y$ will hold true for $Opt'$ as well, but all cases must be considered. These cases must concern pages $A$ and $B$, for they account for the only difference between the fast memories of $Opt$ and $Opt'$.

Consider the first of such a case where $Opt$ and $Opt'$ take different actions in region $x$. This could only occur if some page in the sequence within $x$ (not $A$ or $B$) caused the page $A$ or $B$ to be replaced, otherwise $Opt$ and $Opt'$ would not be taking different actions. This action, however, would
make the fast memories of both solutions identical with an equal number of swaps taken and allow $Opt'$ to remain optimal.

Now consider a second case where both solutions agree up to step $i$, where $Opt$ must perform a swap to put $B$ in fast memory. If $Opt$ replaces $A$, then $Opt$ and $Opt'$ have the same fast memory with $Opt$ having made 1 extra swap, contradicting the notion that it is indeed optimal for all inputs. So, let’s assume that $Opt$ replaces some other page in fast memory that we call $Z$ (which could be anything other than $A$ or $B$, including $C$). $Opt'$ in this case could take the opportunity to also replace $Z$ and return $A$ to fast memory, showing that both solutions can have the same fast memory by step $i$ with the same number of swaps.

Following step $i$, then, $Opt$ and $Opt'$ are identical, and $Opt'$ is one step closer to $G$. This, however, conflicts with the premise that $Opt$ was the closest solution to $G$ and causes a contradiction.

It may also be useful to consider the case that there is no step after $k$ where $A$ occurs in the sequence. In this case, $Opt'$ would not have to make any swap at step $i$ to exactly conform to the fast memory of $Opt$ (step $j$ does not exist and $A$ does not need to be in fast memory), but $Opt$ would necessarily have to make a swap at step $i$. This suggests that $Opt$ would be non-optimal for any case where $B$ occurs in the sequence following step $k$ and $A$ does not.