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. (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.

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 $OPT$ differ at $k$, it must be that $o_k > g_k$, thus $o_{k+1} + (o_k - g_k) > 0$ meaning $OPT'$ adds a valid amount of gas at $k + 1$. Further, because $g_k + (o_{k+1} + (o_k - g_k)) = o_k + o_{k+1}$, $OPT'$ has the same amount of gas in the tank as $OPT$ after filling up at $k + 1$, namely $o_k + o_{k+1} - g_k$. Because $OPT'$ is identical to $OPT$ after $k + 1$, $OPT'$ never runs out of gas after $k + 1$. Finally, because the total gas put in the tank by $OPT'$, over $k$ and $k + 1$, is $g_k + (o_{k+1} + (o_k - g_k)) = o_k + o_{k+1}$, $OPT$ and $OPT'$ 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.

6. (a) This algorithm is correct for the problem of minimizing the total sum of all line penalties. The proof is by contradiction. Assume there is an optimal solution $T$, and call the output of the greedy algorithm $G$. Let $s_i$ be the penalty of the $i$th line of solution $S$. Let $j$ be the number of the first line in $T$ that is different from the $j$th line in $G$. By the definition of the algorithm, $g_i < t_i$. Create a new solution $T'$ by moving the first word of line $i + 1$ in $T$ to the end of line $i$. Let $l$ be the length of this word. Note that $t_{i+1}' = t_{i+1} + l$ and $t_i' = t_i - l$. Therefore, the the total sum of all line penalties in $T'$ is the same as the total sum of all line penalties of $T$. $T'$ is more like greedy than $T$, and has the same total penalty. Contradiction.

(b) This algorithm is incorrect for the problem of minimizing the maximum line penalty. Let $L = 5$, and consider the words “AAA”, “BB”, “CC”, and “DDDD”. The greedy algorithm produces

<table>
<thead>
<tr>
<th>Word</th>
<th>Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>penalty = 0</td>
</tr>
<tr>
<td>CC</td>
<td>penalty = 3</td>
</tr>
<tr>
<td>DDDD</td>
<td>penalty = 1</td>
</tr>
</tbody>
</table>

for a maximum line penalty of 3. The optimal solution is
AAA    penalty = 2
BBCC   penalty = 1
DDDD   penalty = 1

for a maximum line penalty of 2.

7. (Solution by Eric Gratta, but I (Max Bender) personally think this solution is a lot longer than it needs to be) 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 $k$. 

3
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.

9. (a) This algorithm is incorrect for the problem of minimizing the average difference between the heights of skiers and their skis. Let $p_1 = 5$, $p_2 = 10$, $s_1 = 9$, and $s_2 = 14$. 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 \( T = \{(p_1, s_{\alpha(1)}), \ldots, (p_n, s_{\alpha(n)})\} \), 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

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

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

\[
|p_i - s_j| + |p_k - s_i| - |p_i - s_i| - |p_k - s_j| = \\
(s_j - p_i) + (p_k - s_i) - (p_i - s_i) - (p_k - s_j) = \\
2(s_j - p_i) \geq 0
\]
10. SRPT is correct. For contradiction assume SRPT is not correct and thus SRPT’s output, $G$, is not optimal for some $I$. Call $OPT$ the output that is the same as greedy for the most unit time intervals. Call time $k$ the first time when $G$ and $OPT$ disagree. Call the job that $G$ runs at $k$, $j_g$ and the job that $OPT$ runs $j_o$. Call the amount of work remaining prior to time $k$ for $j_g$, $m_g$ and for $j_o$, $m_o$. Note that because $OPT$ and $G$ are feasible and identical prior to $k$, it must be that, in both $OPT$ and $G$, at $m_g$ time units and locations $\geq k$, $j_g$ is run and at $m_o$ time units at locations $\geq k$, $j_o$ is run. Note that these times may not be contiguous and different for $OPT$ and $G$. We create a new optimal solution $OPT'$ as follows. $OPT'$ is identical to $OPT$ for all time units $< k$ and all time units $\geq k$ such that jobs other than $j_o$ or $j_g$ run. For the $m_g + m_o$ time units located at time $\geq k$, we fill them in order earliest to latest first with $m_g$ units of $j_g$ and then $m_o$ units of $j_o$. First we claim $OPT'$ is feasible. Because no job other than $j_g$ and $j_o$ changes, all other jobs are completed in $OPT'$. Because $j_o$ and $j_g$’s remaining units of work have simply been reordered within the time units $OPT$ completed both jobs, there must be enough time units as $OPT$ completed both jobs. Second, we claim that $OPT'$ is more like $G$. By the fact that $OPT$ and $G$ are identical prior to $k$ and $G$ schedules $j_g$ at $k$, $OPT$ must have at least 1 unit of $j_g$ left prior to $k$. Therefore, by the definition of $OPT'$ there will be a unit of $j_g$ at $k$, thus making $OPT'$ like $G$ for one more step than $OPT$. Lastly we show that $OPT'$'s total completion time is no larger than that of $OPT$. As only jobs $j_g$ and $j_o$ have changed, all other completion times remain the same. By the fact that $OPT$ and $G$ are identical prior to $k$, and by the fact that SRPT picked $j_g$ over $j_o$, it must be that $m_g \leq m_o$. Because $OPT'$ uses the same specific time units to schedule $j_g$ and $j_o$ as $OPT$, $OPT'$ must finish $j_g$ no later than $OPT$ finishes the first of the two jobs (regardless of which one $OPT$ finishes first). Finally, $OPT'$ and $OPT$ must finish the second of the two jobs at the same time as they use the same time intervals to complete both jobs. Thus the sum of the completion times of $j_g$ and $j_o$ cannot go down in $OPT'$. Because $OPT'$ is at least as optimal as $OPT$ and like greedy for one more step, we have a contradiction to the choice of $OPT$, thus SRPT must be optimal.

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 $G$ than $T$. Contradiction.

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