CSE 101 Homework 2
Greedy Algorithms
Due Tuesday, May 5

Directions: For each of the first four problems, a “high level” greedy strategy is
given. For some of the problems, the strategies give a correct (optimal) solution, and for
others, it sometimes gives incorrect (suboptimal) solutions. For each, decide whether
the greedy strategy produces optimal solutions. If it is, give a proof that it is correct, then
describe what data structures and preprocessing you would use to give an efficient
version, and give a time analysis.
(10 points, correctness proof, 10 points efficiency and correct time analysis)

If it is not correct, give a counter-example showing the strategy is incorrect.
Then still give an efficient version, as a heuristic. (10 points, counter-example,
10 points efficiency and correct time analysis)

Caravan stops
You are organizing a caravan crossing a desert. Your path is fixed. The caravan can only
travel \( m \) miles between stops at oases. You are given a list of oases \( \text{Oasis}[1..n] \),
each with its distance \( d_i \) in miles from the starting point. The last oasis, \( \text{O}_n \), is
the destination. You wish to choose the minimum size set of stops subject to the constraint
that there are no more than \( m \) miles between consecutive stops.

Candidate greedy strategy: Treat the start as an oasis with \( d_i = 0 \). At each stop, at
oasis \( \text{Oasis}[i] \), if the destination \( d_n \leq d_i + m \), go there directly. Otherwise, find
the oasis \( j \) of maximum \( d_j \) subject to \( d_j \leq d_i + m \). Make \( j \) your next stop, and repeat.

We use a “greedy-stays-ahead” style proof of correctness, in this case quite literally: we
show for the greedy set of stops \( i_1,..i_k \) and the optimal set of stops \( i'_1,..i'_{k'} \), that
\( d_{i_j} \geq d_{i'_j} \) for all \( j = 1..k' \).

For the base case, \( j = 1, i_1 \) is the last stop with \( d_i \leq m \), and \( d_{i'_1} \leq m \)
by the constaint. Thus, \( d_{i_1} \geq d_{i'_1} \).

Assume \( d_{i_j} \geq d_{i'_j} \). Then \( i_{j+1} \) is the last stop with \( d_{i_{j+1}} \leq d_{i_j} + m \), and
\( d_{i'_{j+1}} \leq d_{i'_j} \leq d_{i_j} + m \). Thus, \( i'_{j+1} \) is before \( i'_j \), and \( d_{i'_{j+1}} \leq d_{i_{j+1}} \).

Thus, by induction, \( d_{i_j} \leq d_{i'_j} \) for all \( j \).

In particular, at \( j = k' \), the optimal schedule gets to the end, so \( d_{i'_{k'}} = d_n \leq d_{i_{k'}} \).
Thus, in the same number of steps \( k' \), the greedy schedule has also reached the end. Thus, the greedy schedule
is also an optimal schedule.

To implement this quickly, we just need to keep a counter to where we are in the list of stops (since they are already sorted by distance \( d_i \), and increment the counter until it would point to a distance greater than the current distance + \( m \). We add the stop, and update the current distance to that for the current stop.
This takes at most $O(n)$ time.

**Maximum independent set** An independent set in an undirected graph $G = (V, E)$ is a set of nodes $S \subseteq V$, so that no two nodes in $S$ are adjacent in $E$, i.e., if $\{x, y\} \in E$, we cannot have both $x$ and $y$ in $S$. The maximum independent set problem is to find a largest independent set in a given graph.

Candidate greedy strategy: Pick the node $x$ of smallest degree, and put $x \in S$. Remove $x$ and all of its neighbors from $G$. Repeat until no nodes are left.

This is not always optimal. Consider a graph with 7 nodes, $a, b, c, d, e, f, g$. Let $a$ be connected to just $b$ and $c$, in addition both $b, c$ are connected to both $d, e$, and add all edges between $d, e, f$ and $g$. The smallest degree node is $a$, but if we pick $a$, we can only pick one of $d, e, f$, and $g$. A better solution is to pick $b, c$ and $f$.

To get a fairly efficient solution, we’ll use two arrays $\text{InGraph}$ says whether a node is still in the graph, $\text{Degree}$ computes its degree in the subgraph. We initialize $\text{InGraph}$ to all True and initialize $\text{Degree}$ by computing the degrees for all nodes (incrementing as we run through their adjacency lists). This takes total time $O(n + m)$. A subroutine $\text{Delete}(x)$ sets $\text{InGraph}[x]$ to False and then runs through all nodes adjacent to $x$ and if they are in the graph, decrements their degrees. This takes time $O(1 + \deg(x))$.

Then in each iteration, we run through the nodes and find the smallest $\text{Degree}$ of a node that is $\text{InGraph}$. We then add this node $x$ to $S$ and delete it and all of its neighbors, until no nodes are $\text{InGraph}$.

Note that each node is deleted exactly once, so the total time for all delete subroutines is $O(\sum_{x} (1 + \deg(x))) = O(n + m)$. Each time we sweep the array, it takes $O(n)$ time, and we do it $|S| + 1 \leq n$ times, so the total time for all the sweeps is $O(n^2)$. This gives time $O(n + m + n^2) = O(n^2)$ since there are at most $O(n^2)$ edges in an undirected graph. However, it could be considerably faster if the independent set found is small.

**Oxen pairing** Consider the following problem: We have $n$ oxen, $Ox_1, \ldots, Ox_n$, each with a strength rating $S_i$. We need to pair the oxen up into teams to pull a plow; if $Ox_i$ and $Ox_j$ are in a team, we must have $S_i + S_j \geq P$, where $P$ is the weight of a plow. Each ox can only be in at most one team. Each team has exactly two oxen. We want to maximize the number of teams.

Candidate Greedy Strategy: Take the strongest and weakest oxen. If together they meet the strength requirement, make them a team. Recursively find the most teams among the remaining oxen.
Otherwise, delete the weakest ox. Recursively find the most teams among
the remaining oxen.

This strategy is optimal. We’ll prove it using the following two lemmas:

Lemma 1: Let \( s \) be the strongest ox, and \( w \) the weakest. If \( s + w < P \),
then there is no set of teams that assigns \( w \) to any team.

Proof: If \( \text{Teams} \) is a set of teams in which \( w \) is assigned to a team with
some ox \( s' \), since \( s' \leq s \), \( s' + w \leq s + w < P \), the team could not actually
pull the plow. This contradiction proves the lemma.

Lemma 2: Let \( s \) be the strongest ox, and \( w \) the weakest. Assume \( s + w \geq P \).
Let \( \text{Teams}_2 \) be a set of disjoint teams that can all pull the plow and
not pair \( s \) with \( w \). Then there is a set of disjoint teams \( \text{Teams}_1 \) that
can all pull the plow, which assigns \( w \) to a team with \( s \) and is so that
\(|\text{teams}_1| \geq |\text{teams}_2| \).

Proof: If \( \text{Teams}_2 \) does not assign both \( s \) and \( w \) to teams, let \( \text{Teams}_1 \) be
\( \text{Teams}_2 \) less any team that includes \( s \) or \( w \), together with the team
\( (s, w) \). Since \( s + w \geq P \), and \((w, s)\) is the only team we added, this is
a set of disjoint teams that can all pull the plow. Since at most one of
\( s, w \) were in a team we deleted at most one team, and added one team, so
\(|\text{teams}_1| \geq |\text{teams}_2| \).

Otherwise, let \( \text{Teams}_2 \) partner \( s \) with \( x \) and \( w \) with \( y \). Let \( \text{Teams}_1 =
\text{Teams}_2 - \{(s, x), (w, y)\} \cup \{(x, y), (w, s)\} \). \( \text{Teams}_1 \) is a set of disjoint
teams, and \((w, s)\) can pull the plow by our assumption. Now, since \( w \) and \( y \)
can pull the plow, and \( x \) is at least as strong as the weakest ox \( w \), \( x \) and
\( y \) can pull the plow. Thus, all teams that we added can pull the plow.
Since we deleted two teams and added two teams, \(|\text{teams}_1| = |\text{teams}_2| \).
Thus, we have proved the lemma in both cases.

We can now prove that the greedy strategy is optimal:

Theorem: There is no set of legal teams \( \text{Teams}_2 \) greater than that pro-
duced by the greedy strategy.

We prove this by strong induction on \( n \), the number of oxen. Assume
that the greedy strategy is optimal on all sets of size \( \leq n \). Then on a set
\( \text{Oxen} \) of size \( n \), let \( w \) and \( s \) be the weakest and strongest oxen. Assume
\( \text{Teams}_2 \) is larger than the greedy strategy’s solution \( \text{GreedyTeams} \). If
\( w + s < T \), then by lemma 1, neither \( \text{Teams}_2 \) nor \( \text{GreedyTeams} \) contains
a team with \( w \). Thus, by the induction hypothesis, \( \text{GreedyTeams} \) is the
best solution for \( \text{Oxen} - \{w\} \), and \( \text{Teams}_2 \) is some solution for \( \text{Oxen} - \{w\} \),
so \(|\text{Teams}_2| \leq |\text{GreedyTeams}| \).

If \( w + s \geq T \), then by lemma 2, there is a solution \( \text{Teams}_1 \) which like
the greedy solution, pairs \((w, s)\) and \(|\text{Teams}_1| \geq |\text{Teams}_2| \). Then, since
\( \text{GreedyTeams} - \{(w, s)\} \) is an optimal solution for \( \text{Oxen} - \{s, w\} \) by the
induction hypothesis, and \( \text{Teams}_1 - \{(w, s)\} \) is a legal solution for this
problem, $|\text{Teams}_1 - \{(w, s)\}| \leq |\text{GreedyTeams} - \{(w, s)\}|$ so $|\text{Teams}_1| - 1 \leq |\text{GreedyTeams}| - 1$, so $|\text{Teams}_1| \leq |\text{Teams}_2| \leq |\text{GreedyTeams}|$. So the greedy solution also produces optimal teams on a set of size $n$.

By induction, the greedy solution is optimal for all $N$.

To get an efficient version of the algorithm, first sort the oxen by strength. We either delete the weakest or both the weakest and strongest, so the set that is left is of the form $\text{Oxen}[i..j]$. We just need to keep track of $i$ and $j$. The following algorithm, after the input is sorted, does so:

1. $\text{Teams} \leftarrow \emptyset$
2. $I \leftarrow 1, J \leftarrow n$.
3. While $I < J$ do:
4. \hspace{1em} IF $\text{Oxen}[I] + \text{Oxen}[J] \geq T$ THEN $\text{Teams} \leftarrow \text{Teams} \cup \{(I, J)\}$, $I++, J--$.
5. \hspace{1em} ELSE $I++$.
6. Return $\text{Teams}$.

Since $J - I$ always decreases by at least one, the above loop executes at most $n - 1$ times, so the above loop takes $O(n)$ time. However, we need to spend $O(n \log n)$ time to sort the inputs, which gives $O(n \log n)$ total time.

**Cookie assignment** Consider the following problem:

You are baby-sitting $n$ children and have $m > n$ cookies to divide between them. You must give each child exactly one cookie (of course, you cannot give the same cookie to two different children). Each child has a greed factor $g_i, 1 \leq i \leq n$ which is the minimum size of a cookie that the child will be content with; and each cookie has a size $s_j, 1 \leq j \leq m$. Your goal is to maximize the number of content children, i.e., children $i$ assigned a cookie $j$ with $g_i \leq s_j$.

**Part 1 : 10 points for counter-example** Below is a greedy strategy for this problem that is not guaranteed to produce optimal solutions. Give an example where it fails to produce an optimal solution.

Candidate Strategy one: Assign the largest cookie to the greediest child. Repeat.

Say there are two children, with greeds 2 and 3, and two cookies of sizes 1 and 2. If we give the cookie of size 2 to the greediest child, we must give the cookie of size 1 to the other child, and neither are content. But if we give the largest cookie to the less greedy child, at least that one small child is content, minimizing malcontentedness.
Part 2: 5 pts Here is a greedy strategy that does produce optimal solutions. Illustrate the algorithm on the counter-example you found in Part 1.

Candidate Strategy two: Look at the greediest child. If the largest cookie makes the child content, give the child the largest cookie. Otherwise, give the child the smallest cookie.

On the above example, since the largest cookie doesn’t make the greediest child content, we instead give that child the smaller cookie. Then we give the larger cookie to the less greedy child, making that child content.

Part 2: 10 points For the optimal strategy, Candidate Strategy 2, state and prove a “modify-the-solution lemma” for the case when the greediest child is content with the largest cookie. (As a hint, I do the same with the case when the greediest child is not content with the largest cookie as an appendix to the exam).

Lemma: Assume \( g_1 \) is the largest greed factor of any child, and that \( s_1 \), the largest size cookie, is at least \( g_1 \). Then there is an optimal assignment that assigns child 1 cookie 1.

Let \( \text{Assign}' \) be any optimal assignment. If \( \text{Assign}' \) assigns child 1 cookie 1, then we are done. If \( \text{Assign}' \) assigns child 1 cookie \( j \neq 1 \), then it either does not give any child cookie 1 or gives cookie 1 to some child \( i > 1 \). In the first case, we let \( \text{Assign} \) give cookie 1 to child 1 and leave the other cookies as they are in \( \text{Assign}' \). Since child 1 is content with cookie 1, and the other children have the same cookies they had before, there are at least as many content children and so \( \text{Assign} \) is also optimal. In the second case, define \( \text{Assign} \) to give cookie 1 to child 1, cookie \( j \) to child \( i \), and the other children get the same cookie as \( \text{Assign}' \) gives them. \( g_1 \geq g_i \) since child 1 is the greediest child. Thus, if child 1 is content with cookie \( j \), then so is child \( i \), and by assumption child 1 is content with cookie 1. Thus if both children were content in \( \text{Assign}' \), both are content in \( \text{Assign} \), and \( \text{Assign} \) makes the same number of children content as \( \text{Assign}' \). If not, at least child 1 is content in \( \text{Assign} \) so \( \text{Assign} \) makes at least as many children content as \( \text{Assign}' \). Thus, in all cases \( \text{Assign} \) is an assignment that makes at least as many kids happy as \( \text{Assign}' \) and is hence also optimal, and gives 1 to 1.

Part 4: 10 points For the optimal strategy (Strategy 2), describe an efficient algorithm that carries out the strategy. Your description should specify which data structures you use, and any pre-processing steps. Give a time analysis.

Let’s sort the children by greed, and the cookies by size, with largest first. Note that we always assign children either the largest or smallest cookies. Thus, the remaining cookies are \( C_i, \ldots C_J \) for some
1 ≤ I ≤ J ≤ m. Let’s keep counters for the I and J to keep track of this interval.

The algorithm becomes:

CookieGive (Children[1..n], Cookies[1..m]).

1. Sort Children by greed, Cookies by size, largest to smallest.
2. I ← 1, J ← m.
3. FOR K = 1 to N do:
4. IF sI ≥ gK THEN Assign[K] = I, I ++
5. ELSE Assign[K] = J, J --.
6. Return Assign.

The time to sort are $O(n \log n + m \log m)$, and the inner loop is $O(n)$, giving a total time of $O(m \log m)$ since $m ≥ n$.

**Appendix: The malcontent case**

Lemma: Let $g_1, \ldots, g_n$ be the greed factors of $n$ children, ordered from greediest to least greedy, and let $s_1, \ldots, s_m$ be the sizes of the $m$ cookies, from largest to smallest. Assume $g_1 > s_1$. Let $A'$ be an assignment of cookies to children so that child 1 is not assigned cookie $m$. Then there is an assignment $A$ of cookies to children that assigns cookie $m$ to child 1, so that the number of children $i$ assigned a cookie $j$ with $g_i ≤ s_j$ by $A$ is at least the number of such children in $A'$.

Proof: Let $A'$ assign child 1 cookie $j < m$. In the first case, assume that the smallest cookie $m$ is not assigned to any child by $A'$. Let $A$ assign all children except child 1 the same cookie as $A'$ does, but assign child 1 cookie $m$. Then except for child 1, the same set of children are content in $A$ as $A'$. Since $g_1 > s_1 ≥ s_j ≥ s_1$, child 1 is not content in either $A$ or $A'$. Thus, $A$ and $A'$ have the same number of content children.

In the other case, cookie $m$ is assigned to a child $i > 1$ by $A'$. Then let $A$ assign all children except 1 and $i$ the same cookies as $A'$ does, and assign child 1 cookie $m$ and assign child $i$ cookie $j$. Again, all children except 1 and $i$ are equally content in $A$ and $A'$, and child 1 is not content in either assignment. If child $i$ is content in $A'$, $g_i ≤ s_m ≤ s_j$, so child $i$ is content in $A$. Thus, $A$ has at least as many content children as $A'$.

**Implementation** Implement the greedy algorithm for maximum independent set and test it on random graphs where each possible edge is in the graph with probability $1/2$. What is the average size of the independent set it finds for graphs of different sizes? (Try $n$ as many powers of 2 as you can.) How do you conjecture the size will grow as a function of $n$?

For larger $n$ even the lowest degree node will be connected to about $1/2$ the other nodes. That means that each time you remove a node, the graph
will shrink by about a factor of 2. So you should see that the size of the independent set you find grows around \( \log n \). The biggest independent set in a random graph however is about twice this size.