Note: These are sketches of proofs. We will ask some of you to send us your solutions, and will try to flesh out the proofs as time goes on.

1. **Multiway Cut.** Recall the Multiway Cut problem: given a graph $G = (V, E)$ with each edge $e$ having a capacity $c_e$, and a subset $S = \{s_1, s_2, \ldots, s_k\} \subseteq V$ of terminals, find a set of edges $E_{\text{cut}} \subseteq E$ of minimum capacity such that each connected component of $G \setminus E_{\text{cut}}$ contains at most one terminal.

We saw a greedy algorithm in class that gave a $2(1 - \frac{1}{k})$-approximation for the problem. Here’s another “greedier” approximation algorithm that Mohit proposed in class:

Find a $s_i$-$s_j$ min-cut whose capacity is smallest amongst all $s$-$s'$ min-cuts with $s, s' \in S$. Delete all edges in this cut. Recurse on the resulting components.

Show that this algorithm is also a $2(1 - \frac{1}{k})$-approximation for the problem.

**Solution:** Consider a tree $T$ with a node for each subgraph we create during the course of the algorithm: if we split a component $C$ into two parts $C_1$ and $C_2$ when finding the minimum $s_i$-$s_j$ cut, we add edges $(C, C_1)$ and $(C, C_2)$ to this tree. You should convince yourself that this tree has $k$ leaves (each corresponding to a component in the final solution) and $k-1$ internal nodes (one for each cut we make during the algorithm; proving this uses the fact that any min-cut forms exactly two components.) A leaf of $T$ is labeled $i$ if it contains the terminal $s_i$. We assume that $T$ is rooted at the node corresponding to the entire graph.

Let us consider $\text{OPT}$, and let $O_i$ be the component that contains $s_i$ in this optimal solution. Recall that $\partial O_i$ denotes the set of edges in the cut $(O_i, V \setminus O_i)$. By renumbering, we can assume that $c(\partial O_k)$ is the largest cut, and hence $\sum_{i=1}^{k-1} c(\partial O_i) \leq 2 \left( 1 - \frac{1}{k} \right) \text{OPT}$.

Now let us look at the internal nodes in $T$: label internal nodes with $\{1, 2, \ldots, k-1\}$ in such a way that each internal node gets a distinct label, and an internal node $x$ is labeled $i$ only if some leaf below it has the label $i$. (Check that you can do this!) Recall that an internal node corresponds to some component $C$ breaking up into $C_1$ and $C_2$. If it is labeled $i$, then $s_i \in C$ and also in one of $C_1$ and $C_2$. (Let’s say $s_i \in C_1$.) There must be some terminal $s_j \neq s_i$ in $C_2$. Since the cut we found when splitting $C$ split $s_i$ and $s_j$, it must cost no more than $\text{MinCut}(s_i, s_j)$. Which in turn has cost at most $c(\partial O_i)$, since $\partial O_i$ separates $s_i$ from all other terminals. Hence, the cut at internal node labeled $i$ costs no more than $c(\partial O_i)$, and hence all the cuts sum up to at most $\sum_{i=1}^{k-1} c(\partial O_i) \leq 2 \left( 1 - \frac{1}{k} \right) \text{OPT}$.

2. **Hitting Set.** Given a set $U$, and a family $F = \{S_1, S_2, \ldots, S_k\}$ of subsets of $U$, a hitting set for $F$ is a subset $H \subseteq U$ such that $H \cap S_i \neq \emptyset$ for all $1 \leq i \leq k$. The (Minimum) Hitting Set problem seeks to find a hitting set of the smallest cardinality.
Show that the Set Cover problem discussed in class has a $\rho$-approximation if and only if the Hitting Set problem has a $\rho$-approximation.

**Solution:** Given an instance $(U, \mathcal{F})$ of set-cover, construct a universe $U'$ with an element $e'_i$ for each set $S_i \in \mathcal{F}$, and a family of sets $\mathcal{F}' = \{S'_1, S'_2, \ldots, S'_n\}$ with a set $S'_i$ for each element $e_j$ of $U$, such that $e'_i \in S'_j \iff e_j \in S_i$. Now verify that $(U, \mathcal{F})$ has a set-cover of size $K$ if and only if $(U', \mathcal{F}')$ has a hitting set of size $K$.

3. Better Makespan Minimization. In class, we saw that Graham’s List Scheduling algorithm gave a 2-approximation for the problem of scheduling $n$ jobs on $m$ parallel machines to minimize the makespan. Construct an instance on which this algorithm does no better than a $2 - \frac{1}{m}$-approximation.

**Solution:** Consider $m(m-1)$ jobs of unit size appearing first, followed by one job of size $m$. Then using list scheduling (in this order) gives a makespan of $2m - 1$, whereas the optimum schedule has a makespan of $m$.

Chris had proposed another “greedy” algorithm: sort all the jobs in non-increasing order of sizes, and run List Scheduling with this order of jobs. Show that this gives a 1.5-approximation for the problem.

**Solution:** Consider the job to finish last, say with processing time $p_j$. If this is the only job on its machine, we are optimal (since our makespan $= p_j \leq p_{\text{max}} \leq \text{OPT}$). Else, by the sorted order of scheduling jobs, there must be at least $m + 1$ jobs with processing time $\geq p_j$, and hence OPT is at least $2p_j$. Thus, we can now claim that the total makespan of our schedule is upper bounded by $(\sum_i p_i)/m + p_j \leq \text{OPT} + \text{OPT}/2 = 1.5\text{OPT}$.

One can actually prove a better approximation ratio. If $p_j \leq \text{OPT}/3$, then we clearly have a 4/3-approx. Else, we can show that if $p_j > \text{OPT}/3$ then greedy computes the optimum makespan. Indeed, since the jobs are all bigger than $\text{OPT}/3$, the optimum solution cannot have more than 2 jobs on any machine. But now it suffices to argue that the best way to place the (at most) $2m$ jobs while placing at most 2 jobs on each machine is to place the largest and smallest job on one machine, the second-largest and second-smallest job on the next, and so on. Here is a formal argument due to Virginia. We can assume that (a) in each machine the larger job was processed first, and (b) machines are sorted in decreasing order of the processing time of the first job. Also, since for the remainder of the argument it won’t matter that the job sizes are large we can add jobs of size 0 to OPT until all machines have exactly 2 jobs assigned to them.

Consider machines $i$ and $j$ with $i < j$ and processing times of the first jobs (in OPT) $p_i$ and $p_j$ respectively. By our above assumption, $p_j \leq p_i$. Let $q_i$ and $q_j$ be the second jobs in the machines. Suppose $q_i \geq p_j$, then we’ll have that $p_i + p_j \leq p_i + q_i \leq \text{OPT}$ and $q_i + q_j \leq p_i + p_j \leq \text{OPT}$ and so swapping the second job of $i$ with the first job of $j$ gives the same makespan. Also, clearly if $q_i > q_j$ one can swap the second jobs to obtain an optimum schedule. Hence WLOG we can assume that the $m$ largest jobs
appear as first jobs on the machines and the rest are arranged from the last machine to
the first one in order of decreasing processing times. This is exactly the schedule that
greedy obtains and hence when \( p_n > \frac{OPT}{3} \) greedy achieves the optimum.

4. **Steiner Tree.** An instance of the **Minimum Cost Steiner Tree** problem consists of a
graph \( G = (V, E) \) with positive costs \( c_e \) for each edge \( e \in E \), and a subset \( R \subseteq V \) of required
vertices or terminals. The goal is to find a minimum cost set of edges \( E_{ST} \subseteq E \) that spans
the set \( R \); i.e., any pair of terminals in \( R \) has a connecting path in \( E_{ST} \). Let \( |R| = k \).
Since the edge set \( E_{ST} \) has minimum cost, it will have no cycles and hence be a tree: this is
called a minimum cost Steiner tree on \( R \).

**Note:** Throughout the course, when given a “cost” function \( c \) on \( E \) (perhaps denoted
by \( c_e \) for edge \( e \) ), and a subset \( Y \subseteq E \) of edges, we will define \( c(Y) = \sum_{e \in Y} c_e \).

(a) Let \( \hat{G} = (V, \binom{V}{2}) \) be a complete graph on the same set of nodes \( V \): set the cost \( \hat{c}_e \) of edge
\( e = \{u, v\} \) to be the shortest-path distance according to the edge “lengths” \( c_e \) between
\( u \) and \( v \) in \( G \). (\( \hat{G} \) is called the metric completion of \( G \).)
For any set \( R \) of terminals, show that \( T^* \), the optimal Steiner tree on \( R \) in \( G \) has the
same cost as \( \hat{T}^* \), the optimal Steiner tree on \( R \) in the metric completion \( \hat{G} \).

**Solution:** Simple observation: for any edge \( e = (u, v) \in E \), it is easy to see that
\( \hat{c}(e) \leq c(e) \): the shortest path distance between the endpoints of \( e \) can be at most
the length of \( e \). The tree \( T^* \) is also a Steiner tree in \( \hat{G} \) with the same cost, and
hence \( \hat{c}(\hat{T}^*) \leq \hat{c}(T^*) \leq c(T^*) \).

Now take \( \hat{T}^* \) and construct a subgraph \( H \) of \( G \) by taking the union of all shortest
paths \( P_{uv} \subseteq E \) corresponding to each edge \( \{u, v\} \in \hat{T}^* \). Since the total cost of
\( P_{uv} \) is exactly \( \hat{c}_{\{u,v\}} \), the total cost of edges in \( H \) is at most \( \hat{c}(\hat{T}^*) \). Note that \( H \)
spans \( R \), and hence the optimal Steiner \( T^* \) in \( G \) has cost at most the cost of \( H \);
i.e., \( c(T^*) \leq \hat{c}(\hat{T}^*) \).

(b) Consider the subgraph \( \hat{G}[R] \) be the graph induced by the set of terminals in the graph
\( \hat{G} \), and let \( \hat{T} \) be the minimum spanning tree in \( \hat{G}[R] \) according to the edge costs \( \hat{c}_e \).
Show that \( \hat{c}(\hat{T}) = \sum_{e \in \hat{T}} \hat{c}_e \), the cost of \( \hat{T} \), is at most \( 2(1 - \frac{1}{k}) \sum_{e \in \hat{E}} \hat{c}_e \).

**Solution:** Consider the tree \( T^* \). Start at some terminal in \( R \) and take an Euler
tour of this tree; every time you see a new terminal in \( R \), add an edge from it to
the previous terminal you saw. This gives a Hamilton cycle in \( \hat{G}[R] \) of cost \( 2\hat{c}(\hat{T}^*) \).
Now delete the longest edge on this cycle: this gives a spanning tree in \( \hat{G}[R] \) of
cost at most \( 2(1 - \frac{1}{k}) \sum_{e \in \hat{E}} \hat{c}_e \). Hence the MST \( \hat{T} \) in \( \hat{G}[R] \) has cost \( \hat{c}(\hat{T}) \) at most
as much.

(c) Show how to use the tree \( \hat{T} \) to find a tree \( T \) in \( G \) with cost \( c(T) = \sum_{e \in T} c_e \) at most
\( \hat{c}(\hat{T}) \). Infer that the algorithm you have developed is a \( 2(1 - \frac{1}{k}) \)-approximation to the
minimum cost Steiner tree problem.
Solution: Consider the tree $\hat{T}$: we use the same idea as in part (a) and replace each edge $(x, y) \in \hat{T}$ by a shortest path between $x$ and $y$ in $G$; this gives us a graph $H$ spanning $R$ with $c(H) \leq \hat{c}(\hat{T})$. Now we can go over all the edges of $H$: for each edge, if deleting it still leaves us with a connected graph spanning $R$, we delete it. What remains will be a tree spanning $R$, and this Steiner tree (which we will call $T$) has cost

$$c(T) \leq \hat{c}(\hat{T}) \leq 2\left(1 - \frac{1}{k}\right) \hat{c}(\hat{T}^*) = 2\left(1 - \frac{1}{k}\right) c(T^*).$$

(1)

This proves the claimed result.