DISCLAIMER: These notes are not necessarily an accurate representation of what I said during the class. They are mostly what I intend to say, and have not been carefully edited.

Review: list of topics for the final:

1. Divide-and-Conquer:
   
   (a) Asymptotic complexity: $O$, $\Omega$, and $\Theta$.
   (b) Setting up runtime recurrences, and solving them using Master theorem.
   (c) Binary search, merge sort.
   (d) EXCLUDED:
       
       i. fast multiplication,
       ii. runtime recurrences not solvable using Master theorem,
       iii. other ways of solving runtime recurrences such as guess-and-check.

2. Graphs:

   (a) Bellman-Ford algorithm for computing shortest paths, and its termination conditions (either with distance, or existence of negative cycle).
   (b) Definition of minimum spanning tree.
   (c) EXCLUDED:
       
       i. Shortest path algorithms faster than Bellman-Ford (e.g. BFS/ Dijkstra’s).
       ii. Extracting negative cycles from a non-terminating state of Bellman-Ford.
       iii. Details/correctness of Kruskal’s algorithm.
       iv. Cut/cycle property for minimum spanning trees.
       v. Directed acyclic graphs.

3. EXCLUDED: Dynamic Programming (this was extensively tested on Test 2).

4. Optimization

   (a) Definition of linear program, linear constraints.
   (b) Formulating linear / integer linear programs.
   (c) Formulation of max-flow, residual graphs, and the Ford-Fulkerson algorithm.
(d) EXCLUDED:
   i. Extracting minimum cut from maximum flows.
   ii. Proof / applications of max-flow/min-cut.

5. NP-hardness

   (a) Decision problems, conversion of decision problems.
   (b) Definition of $P$, $NP$.
   (c) Reductions, $NP$-hardness, and $NP$-completeness.
   (d) Definition of 3SAT, and statement of $3$-SAT $\rightarrow A$ for any problem $A$ that’s in $NP$.
   (e) EXCLUDED:
      i. Reductions from 3SAT.
      ii. Reductions to KNAPSACK.

6. Approximation algorithms

   (a) Definition of approximation ratios of minimization/maximization problems.
   (b) Proofs of approximation ratios. For a maximization problem, upper bound $OPT$ while lower bound the algorithm’s output, and vice versa for a minimization problem.

   It there is still time available, I would like to talk about a slightly different approach toward designing efficient algorithms: randomization.
   Recall the 2-approximation for maximum cut. What if we just picked a ‘random’ cut? That is, for each vertex $u$, we include it in $S$ with probability $1/2$.

   Then for an edge $uv$, there are 4 possibilities,

   1. $u \in S$, $v \in S$.
   2. $u \in S$, $v \notin S$.
   3. $u \notin S$, $v \in S$.
   4. $u \notin S$, $v \notin S$.

   Out of these, possibilityes 2 and 3 will result in $uv$ being included in $E(S, V \setminus S)$. As this does not dependent on which sides are the other vertices, the probability of $uv$ being cut is $1/2$.

   Then, we can use an important fact from probability: linearity of expectation. It states that for any two events, $X$ and $Y$ (which can even be dependent), the expectation of $X + Y$ is just the expectation of $X$ plus the expectation of $Y$. That is

   $$E_{X,Y}[X + Y] = E_{X,Y}[X] + E_{X,Y}[Y].$$
To formally define expectation, we need to discuss what is an event. An event, just like the analysis of $uv$ above, is just a distribution, or 'probabilities' over all outcomes, which for simplicity we just denote as

$$1, 2, \ldots, k.$$ 

For each of these outcomes, it has a value of $x_i$ and $y_i$, and we will also describe its probability as $p_i$. Of course, because we have all the outcomes, we have

$$\sum_{i=1}^{k} p_i = 1.$$ 

Then the expectation of $X$ can be defined as:

$$\mathbb{E}_{X,Y} [X] = \sum_{i=1}^{k} p_i x_i,$$

while the expectation of $X + Y$ is

$$\mathbb{E}_{X,Y} [X + Y] = \sum_{i=1}^{k} p_i (x_i + y_i) = \left( \sum_{i=1}^{k} p_i x_i \right) + \left( \sum_{i=1}^{k} p_i y_i \right).$$

This says that linearity of expectation is nothing more than rearranging summations. However, this probabilistic view of random max-cut allows us to do something quite interesting here: we can define a random variable $X_e$ for each edge $e$ that’s 0 if it’s not cut, and 1 if it’s cut.

By what we discussed before we have

$$\mathbb{E} [X_e] = \frac{1}{2},$$

which by linearity of expectation gives:

$$\mathbb{E} \left[ \sum_e X_e \right] = \sum_e \frac{1}{2} = \frac{m}{2}.$$ 

So in expectation we cut $m/2$ edges.

That is, a randomized set $S$ in expectation gets half the edges. Note however that this does not guarantee a 2-approximation: it may be the case that the distribution of edges cut by the algorithm looks like:

$$\begin{cases} 
m & \text{w.p. } \frac{2}{m+2} \\
m/2 - 1 & \text{w.p. } \frac{m}{m+2} 
\end{cases}.$$ 

The expectation of this is

$$m \cdot \frac{2}{m+2} + (m/2 - 1) \cdot \frac{m}{m+2} = m/2,$$

but to get answer that is $m/2$, we will need to run it about $m/2 = \Theta(m)$ times. So it’s significantly slower than the greedy algorithm that we discussed last Monday.