Exercise 1

We can consider different commodities (i.e. different i's) separately. Hereafter we omit i from the subscripts. $\{x_e\}$ can be viewed as a flow from s to t (here t represents the sink, not the threshold variable). In order to get a feasible solution for the first LP, we need to decompose the flow into s,t–paths. This can be done by iteratively looking for an s,t–path P with positive flow, *desaturating* it (taking away as much flow as possible), and setting the value of x_P to the amount of flow being taken away. At most a polynomial number of iterations is required because at least one edge becomes zero during each iteration. It is easy to verify that the two solutions have the same bottleneck and thus the same objective function value.

Exercise 2

We modify Karger's algorithm as follows. Run the algorithm until there are 2α "meta-nodes" left in the graph. Then output a uniformly at random cut surviving in the contracted graph. The probability that a cut of size αk survives the contraction can be computed to be $1/n(n-1)\cdots(n-2\alpha+1) \geq n^{-\alpha}$ following the analysis done in class. The probability that the cut is the final one output is at least $n^{-\alpha}$ times $2^{-2\alpha} = (4n)^{-\alpha}$.

Exercise 3

We divide the request sequence into phases as we did when showing LRU has a competitive ratio of k. As argued in class, OPT makes at least one cache miss eact time a new phase begins. We now claim that FIFO makes at most k cache misses per phase, thus showing that FIFO is k-competitive. To prove the claim, we observe that each phase contains only k distinct pages. If one of these pages causes a cache miss in the phase, it will never cause another cache miss in the same phase again. This is because when a cache miss occurs, that page is brought into the queue. That page will then remain in the queue for the entire phase as there are at most $k-1$ other pages to be pushed into the queue.

Problem 1

To approximate max-cut deterministically, consider an arbitrary ordering of the nodes $v_1, ..., v_n$. Starting with empty sets L and R, build a cut as follows. Consider each v_i in turn. At this point in our algorithm we have constructed two sets L_i and R_i. Of all the edges connecting v_i to L_i and R_i, greedily place v_i into one of the two sets so that at least half of the weight of these edges crosses the cut. At the termination of this algorithm, we have constructed a partion of the node set into L and R such that the total weight of edges crossing the cut is at least $1/2$ of the total weight of all edges in the graph. Since the total weight of edges in the graph upper bounds the total weight

of the max cut, we obtain a 2-approximation to max-cut.

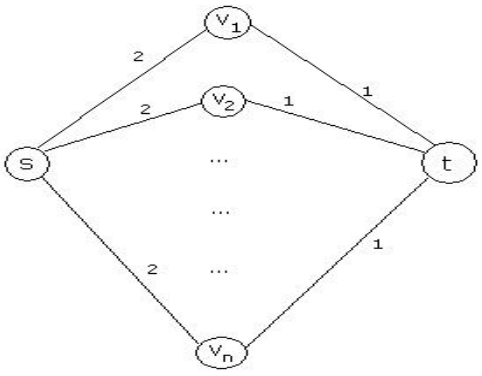
Here's a simpler randomized algorithm: place every node independently into L with probability $1/2$ and R with probability $1/2$. Consider any edge (u, v). The probability that u and v are placed in different sets is $1/2$, so by linearity of expectation, the expected weight of the cut is at least half the total weight of all edges in the graph. Hence we get a 2-approximation (in expectation) to max-cut.

Problem 2

Part a

We first show that the following weighted version of Karger's algorithm has only an exponentially low probability of picking an s-t min-cut: at every step, we pick a random edge (that does not connect the s-supernode to the t-supernode) with probability proportional to its weight and merge its endpoints.

Construct a graph consisting of node set $\{s, t, v_1, ..., v_n\}$, with edges connecting s to $v_1, ..., v_n$ with weight 2 and edges connecting $v_1, ..., v_n$ to t with weight 1. Then clearly the (only) min $s - t$ cut corresponds exactly to the one partition $(\{s, v_1, ..., v_n\}, \{t\})$. Consider any v_i. After the algorithm terminates, exactly one of (s, v_i) or (v_i, t) must be contracted. If (v_i, t) is the edge chosen for contraction, the resulting cut will place v_i in the same set as t and we will not get the min cut. Therefore to obtain the $s - t$ min-cut, we need our algorithm to contract the edges (s, v_i) for $i = 1, ..., n$. Since the probability of contracting one of these edges is exactly $1/2$, we get that the probability of obtaining an $s - t$ min-cut is $\frac{1}{2^n}$, which is exponentially low.



Finally, it is easy to see that this example can be transformed into an unweighted one. Replace every edge $e = (u, v)$ of weight 2 by two new nodes e_1 and e_2 and four edges (u, e_1), (u, e_2), (e_1, v), and (e_2, v). Then, once again the only min-cut corresponds to placing $v_1, ..., v_n$ with t, and the probability of picking a min-cut can again be shown to be exponentially small.

Part b

Consider the graph from part (a), modified so that all edges have equal weight 1. Then since all $s - t$ cuts have equal weight n, any s-t cut is an s-t min-cut. There are 2^n such s-t cuts.

Problem 3

Part a

We follow the proof the Claim 20.4.2 in the lecture notes. (We assume that initially the cache contains k arbitrary distinct pages.) Since there are only $N = k + 1$ distinct pages, the number of new pages in phase i, m_i, is always 1. Then the expected total number of cache misses in phase i is at most $1 + 1/k + 1/(k-1) + \cdots + 1/2 = H_k$. Let M be the number of phases. Then,

$$\mathbf{E}[\text{RAND-1-BIT-LRU}(\sigma)] \leq \sum_{i=1}^{M} H_k = H_k M.$$

Recall that OPT makes at least one cache miss each time a new phase begins. The total number of cache misses is at least M. Hence, randomized 1-bit LRU is H_k-competitive.

Part b

Assume that initially the cache contains page 1 and page 2. Consider the following request sequence: 3, 4, 2, 4. The optimal paging algorithm always keeps page 2 in the cache, resulting in 2 cache misses. Let us see what randomized 1-bit LRU does. The first three requests (3, 4 and 2) cause a cache miss each. The last request causes a cache miss with probability 1/2 because 4 is in the cache when 2 is inserted, and is evicted with probability 1/2. The expected total number of cache misses is 7/2, which is greater than $2H_2 = 3$. Hence the algorithm is not H_k competitive.

Problem 4

Part a

We claim that in any solution produced by FF and for any pair of bins, the total size of items in the two bins is ≥ 1. Suppose that this is not true and let A and B be two bins with total size < 1. Assume without loss of generality that B is placed after A. This is a contradiction, because if we were using FF, some item in B would have been placed in A as the total size was not exceeding 1. Now let FF be the number of bins used in the FF solution and OPT be the number of bins used in the optimal solution. Let s be the total size of items we want to put into bins. Then, by the above claim, we have $s \geq FF/2$. And clearly, $OPT \geq s$. We get $OPT \geq FF/2$, and hence FF is 2-competitive.

To obtain a bound of 7/4 on the c.r., we assume that FF loads n bins and that xn of them have load at least 2/3, and the rest of the $(1-x)n$ have load strictly less than 2/3. We will now focus on these $(1-x)n$ "light bins". The first light bin (and indeed every other one) has space more than 1/3. This implies that all light bins except the first contain items of size more than 1/3. But since the load in any light bin is less than 2/3, it cannot contain more than one item. This implies that OPT must also place each of these items one in each bin, and $OPT \geq (1-x)n - 1$. On the other hand, as argued earlier, the average load of these bins is at least 1/2, therefore, the total load in all bins is at least $2/3xn + 1/2(1-x)n$.

Together these imply that $OPT \geq \max\{(1-x)n - 1, (1/2 + x/6)n\}$ which is at least $4n/7 - 1$, which implies a weak c.r. of 7/4.

Part b

We give an example showing a gap of 3/2. Let the sequence of item sizes be 1/2, 1/4, 3/8, 1/2, 3/8. The optimal solution puts 1/2 and 1/2 in one bin and the other three items in another, requiring 2 bins in total. FF puts 1/2 and 1/4 in the first bin, 3/8 and 1/2 in the second bin, and the last 3/8 in the third bin. Hence, the gap is 3/2.

Problem 5

Part a

We prove a lower bound on the competitive ratio by giving a specific bad sequence of calls to the algorithm. We first give a call from v_0 to v_D. If the algorithm rejects this call, then we terminate the sequence. OPT accepts the call and has value 1, whereas the algorithm gets a value of 0. On the other hand, if the algorithm accepts the call, then we give D other calls of the form (v_i, v_{i+1}) for $i = 0, ..., D-1$. OPT rejects the first call and accepts the remaining, obtaining a value of D, whereas ALG cannot accept any calls after the first. In either case, the competitive ratio is $\Omega(D)$.

Note that this argument only shows that any deterministic algorithm must have a strong competitive ratio of $\Omega(D)$. By dividing up the line into m equal parts, and presenting a sequence as above to the algorithm for each portion of length D/m, we can show that no deterministic algorithm can have a weak competitive ratio of at most D/m with an additive term of at most m.

Part b

First note that if all calls have between 2^i and 2^{i+1} edges, then any call admitted by our algorithm can block at most 3 future calls admitted by OPT. (In the worst case, our algorithm admits a long call with 2^{i+1} edges, which overlaps with at most 3 mutually-non-overlapping "short" calls of 2^i edges admitted by OPT.) Therefore, the algorithm's competitive ratio in this case is 3. Now, let m_i denote the number of calls admitted by OPT with between 2^i and 2^{i+1} edges. Then, the expected number of calls admitted by our algorithm is at least

$$\sum_i \Pr[\text{ALG picks the integer i}]\frac{1}{3}m_i = \sum_i \frac{1}{k}\frac{1}{3}m_i = \frac{1}{3k}\sum_i m_i = \frac{1}{3k}OPT$$

Therefore, our algorithm is $3k$-competitive.

Problem 6

We first show that if all experts have equal weight at the beginning of a block, then the number of mistakes made by the algorithm is at most $O(m + \log n)$. Without loss of generality, suppose that all experts have weight 1 at the beginning of the block. Let W_t be the total weight of experts at the beginning of round t (counting from the start of the block). Then $W_1 = n$. Consider a

round t in which the algorithm predicts wrongly. Let I be the set of experts that made a mistake in this round. Then $\sum_{i \in I} w_i > \sum_{i \notin I} w_i$, where w_i is the weight of expert i at the beginning of this round. This implies that $\sum_{i \in I} w_i > W_t/2$. Now we penalize some of the experts in I. Let I' be the subset of experts in I with weight less than $W_t/4n$. Then, the total weight of I' is at most $W_t/4n \times n = W_t/4$, and so the total weight of $I \setminus I'$ is greater than $W_t/2 - W_t/4 = W_t/4$. Penalizing all the experts in $I \setminus I'$, we see that the reduction in total weight is at least $W_t/8$. Let k be the number of mistakes made by the algorithm in the block. The total weight of all experts at the end of the block is at most $W_1(7/8)^k = n(7/8)^k$. Note that the best expert has weight at least $1/2^m$ at the end of the block. Hence we have $n(7/8)^k \geq 1/2^m \implies k = O(m + \log n)$.

Now, we can still bound k similarly even if the experts have different weights at the beginning. Note that the following is an invariant throughout the algorithm: Let $w_{i,t}$ be the weight of expert i at the beginning of some round t. Then $w_{i,t} \geq W_t/8n$. We can show this by induction. Clearly this holds initially. Suppose that $w_{i,t-1} \geq W_{t-1}/8n$. If expert i was not penalized in round $t-1$, $w_{i,t} = w_{i,t-1} \geq W_{t-1}/8n \geq W_t/8n$, because the total weight never increases. Otherwise, if expert i was penalized, we know that $w_{i,t-1} \geq W_{t-1}/4n$ and so $w_{i,t} = w_{i,t-1}/2 \geq W_{t-1}/8n \geq W_t/8n$. We now bound k. At the end of the block, the best expert has weight at least $W_1/(8n \cdot 2^m)$. Hence we have $W_1(7/8)^k \geq W_1/(8n \cdot 2^m) \implies k = O(m + \log n)$.