CSE 203 Homework 2
Randomized algorithms
Answer Key

For each of the algorithm problems, design as asymptotically efficient an algorithm as possible. Give a correctness argument (explanation, if it is relatively simple, or proof if not) and time analysis. You may use any well-known algorithm or data structure, or algorithm from the text or from class, as a subroutine without needing to provide details. The first four problems are 10 points each, the last 20 points for five parts.

Batch RSA: The signature verification algorithm takes a public key e, N, a message m, a signature s and verifies that: $s^e \equiv m \ mod \ N$. If so, we say s is a valid signature of m. The batch verification problem is: given e, N and n supposed signed messages $s_1, m_1, s_2, m_2, ...s_n, m_n$, are all of the s_i's valid signatures of the corresponding m_i ? Give a probabilistic algorithm that decides this problem with error probability at most ϵ and uses $O((n + \ \log e) \ \log 1/\epsilon)$ modular multiplications. (Hint: you just have to return "Yes, they are all valid", or "No, at least one is invalid". You don't have to find the set of invalid signatures.)

Here's a simple algorithm: For each $1 \leq i \leq n$, flip a coin $c_i \in_U \{0, 1\}$. Let s be the product of the signatures s_i where the i'th coin came up heads, $c_i = 1$, and let m be the analogous product of all the messages. Check to see that $s^e = m mod N$ (using repeated squaring). The first step takes $O(n)$ modular multiplications and the second $\log e$.

If each $s_i^e = m_i$, then $s^e = (\pi_{\{i|c_i=1\}} s_i)^e = \pi_{\{i|c_i=1\}}(s_i^e) = \pi_{\{i|c_i=1\}}(m_i) = m$, where all arithmetic is mod N.

If $s_i^e \neq m_i$, fix all the c_j for $j \neq i$, and let s' and m' be the corresponding products. If $(s')^e = m'$, and $c_i = 1$, then $s^e = (s')^e (s_i)^e = m'(s_i)^e \neq (m' m_i) = m$. If $(s')^e \neq m'$, if $c_i = 0$, $s = s'$ and $m = m'$, and we reject. So in either case, we have a conditional probability of at least $1/2$ of rejection.

Repeating the above $\log 1/\epsilon$ times then gives us a false acceptance probability of at most ϵ.

Max Cut for degree 3 graphs Consider the Max Cut problem restricted to graphs where each node has at most 3 neighbors. Give the best poly-time probabilistic approximation algorithm you can (in terms of approximation ratio). If you can, go on to make your algorithm deterministic. (Hint: One approach is to consider a maximal matching in the graph. Your ratio should be at least 5/9.)

Solution 1: Find any maximal matching M. Note that for every edge u, v we add to M, we rule out at most 4 edges, 2 each incident to u and to v.

Thus, $|M| \geq m/5$. Pick a random cut as follows: For each $e = u, v$, with probability $1/2$ put $u \in L$ and $v \in R$, and with probability $1/2$, put $v \in L$ and $u \in R$. Then, for each unmatched node x, put it in L or R with equal probability. Note that each edge not in M has a $1/2$ probability of being in the cut. Each edge in M is in the cut. Thus, the expected size of the cut is $1/2(m - |M|) + |M| = 1/2(m + |M|) \geq 1/2(1.2m) = .6m$. So this method gives an expected cut of at least .6 of the edges, and so at least .6 of the maximum cut.

We can use the method of conditional expectations to derandomize this algorithm, and would get a deterministic greedy algorithm. We would first consider matched edges, and insert them into the current partial cut in the way that keeps the edge in the cut and maximizes the size of the partial cut. Then we would add the unmatched nodes one by one, again maximizing the cut. Since, not counting matching edges, each step keeps the number of cut edges greater than the number of same side edges, the same bound would hold.

Solution 2: Order the vertices at random. For each in order, add it to the side of the cut that maximizes the partial cut.

At each step, we add at least as many edges to the cut as edges not in the cut. In steps where the vertex has an odd number of neighbors that precede it in the ordering, we add at least one more edge to the cut. Thus, if C is the set of cut edges at the end, $|C| - (m - |C|) \geq$ number of vertices v with an odd number of neighbors before it. Let $Odds$ be the latter number. Then $E[C] \geq m/2 + 1/2E[Odds]$ by rewriting the previous inequality and taking the expectation of both sides.

Consider any vertex v and say that its neighbors are $u_1, ..u_k$ Among v and its neighbors, v is equally likely to be the 1'st, 2nd, 3rd, .. $k+1$st in the ordering. v has an odd number of previous neighbors if it is in any of the even positions, so this happens with probability $\lfloor (k+1)/2 \rfloor / (k+1)$. In our example, k is either 1,2, or 3, with the probabilities $1/2, 1/3$, and $1/2$ respectively. Let n_1 be the number of vertices of degree 1, n_2 of degree 2 and n_3 the number of vertices of degree 3. Then $m = 1/2n_1 + n_2 + 3/2n_3$ and and $E[Odds] = 1/2n_1 + 1/3n_2 + 1/2n_3 \geq 1/6n_1 + 1/3n_2 + 1/2n_3 = 1/3m$ Thus, $E[Cut] \geq m/2 + 1/6m = 2/3m$, so this gives an expected 2/3 approximation ratio.

We can repeat several times and take the best, or use the method of conditional expectations to find an ordering where $Odds \geq E[Odds]$. Given a partial ordering, we can compute the number of elements already in odds, and then for each vertex with k neighbors not yet in the ordering, the probability will either be $\lfloor (k+1)/2 \rfloor$ or the ceiling of the same quantity, depending on whether the number of neighbors already in the order is even or odd. Given a partial ordering, we can compute this conditional expectation over all at most n ways of extending it with a new vertex, and take the largest.

Popular elements: Let $a_1, ..a_n$ be a stream of not necessarily distinct elements from $\{0,1\}^k$. Say that x is *ϵ-popular* if $a_i = x$ for at least ϵn values of i. Give a randomized algorithm that uses memory $O(poly(k + \log n + \log(1/\epsilon))/\epsilon)$ that returns a set of strings so that with probability .99, the following two properties hold: every element returned is $\epsilon/2$-popular, and every ϵ-popular element is returned. (Hint: the memory requirement above allows you to store approximately $\log(1/\epsilon)/\epsilon$ elements from the stream, and associated counters.)

First, assume we know n. Let $p = 4\log(100/\epsilon)/(\epsilon n)$.

We first show how to solve the problem with expected memory $pn(\log n + k)$, then argue that even if we have a fixed memory supply of this order, we succeed with high probability. We'll maintain a set S of strings, and for each string $x \in S$, a counter $1 \leq c(x) \leq n$. Each string takes k bits to store, and each counter $\log n$ bits to store. So we need to ensure that the expected size of S is at most pn.

Initially, S is empty. For each element a_i, if $a_i \in S$, we increment $c(a_i)$. If not, we flip a coin of bias p and if it comes up heads, we add a to S and set its counter to 1. Otherwise, we do nothing.

At the end, we return each string in S with counter at least $\epsilon/2n$.

The above algorithm flips a coin of bias p at most n times and adds an element to S if the coin is heads. The expected size of S is thus at most pn. Applying Theorem 4.1 of Motwani-Raghavan (page 68) with $\mu = pn$ and $\delta = 3$, the probability that the actual size is greater than $4pn$ is at most $(e^3/4^4)^{pn} < (1/4)^{pn} = 2^{-2(4\log(100/\epsilon)/\epsilon)} < 2^{-48}$ since $\epsilon \leq 1$ and $\log 100 > 6$. This is pretty darn small. So if we have a fixed memory of $4pn(\log n + k) = O((\log 1/\epsilon)(\log n + k)/\epsilon)$, and abort if we overflow this memory, the probability of overflow is at most $2^{-48} < 10^{-12}$.

Since we only return elements whose counters are at least $\epsilon/2n$, we only return elements that are $\epsilon/2$-popular. On the other hand, there are at most $1/\epsilon$ ϵ-popular elements. For each such element, if we start a counter by the time it appears $\epsilon/2n$ times, then the final counter will reach $\epsilon/2n$ and we will return it. The probability that we never start a counter in the first $\epsilon/2n$ occurences is $(1-p)^{\epsilon/2n} \leq e^{-\epsilon pn/2} = e^{-\log 100/\epsilon} < \epsilon/100$. Thus, the probability that ANY of the $1/\epsilon$ popular elements are not returned is at most $(1/\epsilon)\epsilon/100 = .01$.

If we don't know n in advance, we can add x_i to S with probability $p_i = 2\log(100/\epsilon)/(\epsilon i)$ (or 1, if $p_i > 1$.) The expected size of S grows to $(2\log(100/\epsilon))/\epsilon(\sum_{i=1}^{i=n} 1/i) = O(\log(1/\epsilon)/\epsilon \log n)$, which adds another

$\log n$ factor to the memory required. Otherwise, the analysis is identical, since the odds of storing an element only increases.

On-line makespan : Consider the makespan problem with m machines when restricted to items of either size 1 or 2. Call this problem $1,2$-makespan. In the on-line version, jobs arrive in an arbitrary order, and need to be immediately assigned to one of the machines.

a Prove that no deterministic on-line algorithm for $1,2$ makespan has approximation ratio less than $3/2$.

Consider the behavior of the algorithm with 2 machines on an input with two jobs of size 1. If the algorithm puts them both on the same machine, it has makespan 2 for an input requiring only makespan 1, so its ratio is at least 2. If it puts them on different machines, and then it recieves a job of size 2, then it will have makespan 3 for an input of optimal makespan 2 (putting the job of size 2 on one machine and the two of size 1 on teh other), so its ratio is at least $3/2$.

b Give and analyze a deterministic on-line $1,2$ makespan algorithm with approximation ratio $3/2$.

Consider the algorithm that assigns the next job to the machine with least currently assigned work. First, for jobs of size 1 and 2, we claim that this algorithm maintains the following invariant: the heaviest load of a machine is at most the lightest load + 2. If the invariant is true at a time, and a job arrives of size at most 2, then since it is assigned to the lightest loaded machine, the lightest load only increases, and the heaviest load is either the old heaviest load or the old lightest load + the new job. In either case, the new heaviest load is at most 2 + the old lightest load, which is at most 2+ the new lightest load. So the invariant is maintained.

At the end of the algorithm, let H_{Greedy} be the makespan of our algorithm. If the lightest load is also H_{Greedy}, then the total work for all jobs is mH_{Greedy}, and the optimal makespan is also H_{Greedy}. Otherwise, the lighest load is at least $H_{Greedy} - 2$, so the total work is at least $(m-1)(H_{Greedy} - 2) + H_{Greedy} > m(H_{Greedy} - 2)$. So the optimal schedule has makespan $H_{OPT} > H_{Greedy} - 2$. Since both H_{Greedy} and H_{Opt} are integers, $H_{Opt} \geq H_{Greedy} - 1$ or $H_{Greedy} \leq H_{Opt} + 1$. If $H_{Opt} \geq 2$, this means $H_{Greedy} \leq 3/2 H_{Opt}$. Otherwise, $H_{Opt} = 1$, and the input is a set of at most m jobs of size 1; the greedy algorithm scheudles each of these jobs on a distinct machine and achieves the optimal makespan.

c Remember that an algorithm has *competitiveratio* R if there is a constant B with $Cost(Algorithm(x)) \leq R * Cost(Opt(x)) + B$ for every

x. Give a deterministic on-line 1,2 makespan algorithm with competitive ratio 1 (Hint: to make later parts easier, try proving $B = 1$; $B = 2$ should be immediate).

This was already done in part b. In fact, it followed from the invariant that the maximum load differs from the minimum by at most 2.

d A probabilistic algorithm has approximation ratio R if $E(Cost(Algorithm(x))) \leq RCost(Opt(x))$ for every instance x. Prove that no probabilistic on-line 1-2 makespan algorithm has approximation ratio less than 4/3.

Again, consider the behavior of the algorithm with 2 machines on an input with two jobs of size 1. The algorithm puts them both on the same machine with some probability $0 \leq p \leq 1$, and on different machines with probability $1-p$. If we end there, the algorithm has makespan 2 for an input requiring only makespan 1 with probability p, and makespan 1, with probability $1-p$, so its ratio is at least $2p + 1(1-p) = 1+p$. Thus, if $p \geq 1/3$, its ratio is at least $4/3$. If $p \leq 1/3$ and it then recieves a job of size 2, then it will have makespan 3 for an input of optimal makespan 2 (putting the job of size 2 on one machine and the two of size 1 on teh other) with probability at least 1-p, so its ratio is at least $1/2(2p+3(1-p)) = 1/2(3-p) \geq 4/3$ for $p \leq 1/3$. So in either case, the ratio is at least $4/3$.

e Give and analyze a probabilistic on-line 1-2 makespan algorithm with approximation ratio 4/3, i.e., better than any deterministic algorithm. (Hint: look at the algorithm in part c. What inputs are bad for this algorithm? Devise a second algorithm that is good for this type of input. The probabilistic algorithm picks one of the two deterministic algorithms, with some probability p, and the other with probability $1-p$, to get the best average.)

Consider the following PairOnes algorithm. On an input of size 2, assign it to the least loaded machine. On an input of size 1, if there is a machine of load equal to the least load + 1, assign it to that machine; otherwise assign it to the least loaded machine. PairOnes has the same invariant that Greedy had, that the biggest load is at most 2 more than the smallest. So just as before $H_{PairOnes} \leq H_{Opt} + 1$.

The probabilistic algorithm *Random* is: flip a coin before any input arrives with bias 2/3. If heads, run Greedy, otherwise, run PairOnes. If $H_{Opt} \geq 3$, then both H_{Greedy} and $H_{PairOnes}$ are less than $4/3H_{Opt}$, so we achieve ratio 4/3 either way.

If $H_{Opt} = 1$, then $H_{Greedy} = 1$ and $H_{PairOnes} = 2$, because we have at most m distince jobs of size 1. Thus $E[H_{Random}] = 2/3H_{Greedy} + 1/3H_{PairOnes} = 4/3 = 4/3H_{Opt}$.

If $H_{Opt} = 2$, then $H_{Greedy} \leq 3$. Also if there are k jobs of size 1 and l of size 2, $k + 2l \leq 2m$. Since PairOnes will only put a job of size 1 on a new machine if each machine with at least one job is of load 2, it will have at most $\lceil k/2 \rceil$ machines with ones, and at most l machines with 2s. Thus, until the last jobs, there will always be an empty machine, and if there is no empty machine for the last job, it must be a job of size 1 placed on a machine with load 1. Thus, the final makespan for $PairOnes$ will also be 2. Thus, the ratio in this case is $(3(2/3) + 2(1/3))/2 = 4/3$.

So in all cases, the ratio of *Random* is at most $4/3$.