Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute of Theoretical Computer Science
Bernd Gärtner, Mohsen Ghaffari, Rasmus Kyng, David Steurer

Algorithms, Probability, and Computing **Solutions for SPA 1** **HS20**

Solution 1

(a) We first describe the algorithm, which consists of four steps.

(1) We invoke $\mathcal{A}_{sample}$ with input G and sample probability $p = 1/\sqrt{n}$, thus obtaining a graph G' that contains each edge of G independently with probability p.

(2) Next, we compute a Minimum Spanning Forest T' of G' by invoking $\mathcal{A}_{\mathrm{MSF}}$.

(3) We invoke $\mathcal{A}_{T-heavy}$ to remove all T'-heavy edges from G. We call the resulting graph G''.

(4) We invoke $\mathcal{A}_{\mathrm{MSF}}$ with input G'' to obtain the Minimum Spanning Forest T of G'' and the algorithm outputs T.

We invoke each of the subroutines at most a constant number of times. Hence, it remains to argue that the algorithm indeed outputs the Minimum Spanning Tree of G with probability $1 - O(1/n)$. To that end, it suffices to show that both the graph G' and the graph G'' have $O(n^{3/2})$ edges with probability $1 - O(1/n)$. The reason for that is the following. In the final step, we output the Minimum Spanning Forest of G''. We obtained G'' by removing all the T'-heavy edges of G, but those edges are not contained in the Minimum Spanning Tree of G, as argued in the lecture. Hence, all the edges contained in the Minimum Spanning Tree of G are also edges in the graph G''. Therefore, our output T is indeed the Minimum Spanning Tree of G. We first use a Chernoff Bound to argue that G' contains $O(n^{3/2})$ edges with probability $1 - O(1/n)$. To that end, let E denote the set of edges of G. For each $e \in E$, let X_e denote the indicator variable for the event that e is contained in G' and let $X := \sum_{e \in E} X_e$. We have $\mathbb{E}[X] = p \cdot |E| \leq (1/\sqrt{n}) \cdot \binom{n}{2} \leq n^{3/2}$. Note that if $|E| \leq n^{3/2}$, then we trivially sample at most $O(n^{3/2})$ edges, and $|E| > n^{3/2}$ implies $\mathbb{E}[X] \geq n$. In that case, a Chernoff Bound implies

$$\Pr[X \geq 2n^{3/2}] \leq \Pr[X \geq (1+1)\mathbb{E}[X]] \leq e^{-\frac{1}{3}\mathbb{E}[X]} \leq e^{-\Theta(n)} = O(1/n).$$

Thus, at most $2n^{3/2} = O(n^{3/2})$ edges are sampled with probability $1 - O(1/n)$. Next, we argue that G'' contains $O(n^{3/2})$ edges with probability $1 - O(1/n)$. To analyze how many not T'-heavy edges remain, consider the following procedure.

```
(1) Let $E' = T' = F = \emptyset$.
(2) for $i = 1, \dots, m$ do:
(3)     if $e_i$ connects two components of $T'$ then
(4)         Flip a biased coin that comes up heads with probability $p = 1/\sqrt{n}$ to decide
            whether $e_i$ belongs to $G'$.
            If so, let $E' := E' + e_i$ and $T' := T' + e_i$, otherwise let $F := F + e_i$.
(5)     else
(6)         Flip a biased coin that comes up heads with probability $p = 1/\sqrt{n}$.
            If so, add $e_i$ to $E'$.
            (Observe that $e_i$ is $T'$-heavy and thus cannot belong to $F$.)
```

Note that this is the same procedure as in the script, except that this time we always throw a biased coin instead of a fair coin. The reason is that we don't sample each edge with probability $1/2$, but instead with probability $p = 1/\sqrt{n}$. As argued in the script, each edge in $E \setminus T'$ that is not T'-heavy is contained in F. Thus, we only need to show that F contains at most $O(n^{3/2})$ edges with probability $1 - O(1/n)$. Note that during the run of the algorithm, we can only add edges to F as long as the biased coin that we throw in line (4) came up heads strictly less than $n - 1$ times, as each time the coin comes up heads, we add one more edge to the forest T'. Hence, the probability that F contains more than $2n^{3/2}$ edges is upper bounded by the probability that out of $2n^{3/2}$ independent biased coin throws, the coin comes up heads less than n times. One can use a Chernoff bound to argue that this is indeed the case.

Solution 2

(a) For the sake of simplicity, we assume that the input hypergraph $G = (V, E)$ only consists of those edges that either join 2 or 3 vertices, and does not contain any singleton-edges. Our algorithm repeatedly contracts a hyperedge chosen uniformly at random until the graph contains at most 3 vertices. Afterwards, the algorithm chooses a vertex among the at most three remaining vertices uniformly at random. Note that each vertex in the contracted graph naturally corresponds to a set of vertices in the original graph. Let S denote the set of vertices that the randomly chosen node corresponds to. Our algorithm outputs the cut corresponding to the non-trivial partition $(S, V \setminus S)$. Let us first analyze the running time of the algorithm. The algorithm performs $O(n)$ contractions, taking $O(n) \cdot O(n^2) = O(n^3)$ time in total. Besides that, our algorithm needs to keep track for each vertex in one of the contracted graphs to which set of vertices in the original graph it corresponds to. This can be done in $O(n)$ time by computing the corresponding sets in a recursive manner as follows. At the beginning, each vertex in the original graph trivially corresponds to the singleton set only containing itself. Now, assume that during the process we contract some hyperedge e. For each vertex not contained in e, the set of vertices in the original graph it corresponds to does not change. Moreover, all vertices contained in e will form a new node in the contracted graph and this node corresponds to the set of vertices $\cup_{v \in e} S_v$, where S_v is the set of vertices that v corresponds to in the original graph. Hence, we can obtain in $O(n^3)$ time the set S. Afterwards, it remains to iterate through the $O(n^3)$ edges, checking for each edge whether it crosses the cut. Hence, the total runtime is $O(n^3)$, as desired. Now, for a given multihypergraph G and a given minimum cut C of G, let $p_{G,C}$ denote the probability that our algorithm indeed outputs C on the input graph G and for every $n \geq 2$, let

$$p_n := \inf_{G \text{ is an } n\text{-node multihypergraph}, C \text{ is a minimum cut of } G} p_{G,C}.$$

Note that we have $p_2 = 1$ and $p_3 \geq 1/3$. Now, consider an arbitrary n-node hypergraph G with $n \geq 4$ and let C denote an arbitrary minimum cut. Let us focus on the first contraction and let k denote the minimum cut size of G. The probability that no edge in C gets contracted can be lower bounded by

$$1 - \frac{|C|}{|E|} \geq 1 - \frac{k}{nk/3} = 1 - \frac{3}{n}.$$

Note that $|E| \geq \frac{nk}{3}$, as the degree of each vertex is at least k and each edge joins at most 3 vertices. Now, after the first contraction, we end up with a graph $\tilde{G}$, which has either $n-1$ or $n-2$ vertices. Moreover, given that the contracted edge is not in C, there exists a cut $\tilde{C}$ in the new graph that corresponds to C, i.e., there is a natural one to one correspondence between edges of C and $\tilde{C}$. As a contraction can only increase the size of the minimum cut, $\tilde{C}$ is a minimum cut in the contracted graph. Hence, we can deduce that

$$p_{G,C} \geq \left(1 - \frac{3}{n}\right) p_{\tilde{G},\tilde{C}} \geq \left(1 - \frac{3}{n}\right) \min(p_{n-1}, p_{n-2})$$

and therefore also

$$p_n \geq \left(1 - \frac{3}{n}\right) \min(p_{n-1}, p_{n-2}).$$

Now, by a simple induction, one can show that for $n \geq 4$, it holds that

$$p_n \geq \frac{n-3}{n} \frac{n-4}{n-1} \cdot \ldots \cdot \frac{1}{4} \cdot \frac{1}{3} = \Omega(1/n^3),$$

as desired.

(b) Let $\mathcal{C}$ denote the set containing all the minimum cuts. For each minimum cut $C \in \mathcal{C}$, let E_C denote the event that the algorithm from a) outputs C. Note that we have $\Pr[E_C] = \Omega(1/n^3)$. Moreover, $\Pr[E_C \cap E_{C'}] = 0$ for $C \neq C'$, i.e., the events are pairwise disjoint. Hence, we have

$$1 \geq \Pr[\cup_{C \in \mathcal{C}} E_C] = \sum_{C \in \mathcal{C}} \Pr[E_C] \geq |\mathcal{C}| \cdot \Omega(1/n^3).$$

Hence, we can deduce that $|\mathcal{C}| = O(n^3)$, as desired.

(c) Let c be a fixed constant such that the algorithm from a) outputs each minimum cut with probability at least c/n^3. Our algorithm performs $\lceil n^3/c \rceil$ independent runs of the algorithm from a), outputting the size of the smallest encountered cut. The runtime of the algorithm is clearly $O(n^6)$. Moreover, the success probability of the algorithm is at least

$$1 - (1 - c/n^3)^{\lceil n^3/c \rceil} \geq 1 - e^{-(c/n^3)\lceil n^3/c \rceil} \geq 1 - e^{-1} \geq 1/2,$$

where we used that $1 + x \leq e^x$ for every $x \in \mathbb{R}$.

(d) As in the previous exercise, let c be a fixed constant such that the algorithm from a) outputs each minimum cut with probability at least c/n^3. Our algorithm performs $N = \lceil 4 \log n (n^3/c) \rceil$ independent runs of the algorithm from a). This way, it obtains a collection of N cuts. The algorithm outputs each cut that is contained in that collection and whose size is not strictly greater than the size of a different cut in the collection. The runtime of the algorithm is clearly $O(n^6 \log n)$. This runtime bound can even be achieved if one removes all duplicates by using hashing. We don't go into detail here. Now, let $\mathcal{C}$ denote the set containing all the minimum cuts and for each $C \in \mathcal{C}$, let A_C denote the event that the algorithm outputs C. We have

$$\Pr[A_C] \geq 1 - (1 - c/n^3)^N \geq 1 - e^{-(c/n^3)N} \geq 1 - e^{-4 \log n} = 1 - n^{-4}.$$

Hence, by a Union Bound we can lower bound the success probability of the algorithm by

$$1 - \Pr[\cup_{C \in \mathcal{C}} \overline{A_C}] \geq 1 - \sum_{C \in \mathcal{C}} \Pr[\overline{A_C}] \geq 1 - |\mathcal{C}| \cdot n^{-4} = 1 - O(1/n) \geq 1/2,$$

where in the last step we assumed that n is reasonably large.

(e) Our new algorithm is parameterized by two parameters N and t, which we only fix later. The algorithm repeats the following procedure N times and returns the minimum of all the N runs. In a single run, the algorithm repeatedly contracts an edge chosen uniformly at random, as long as the contracted graph has at least t vertices. Once the contracted graph has less than t vertices, we invoke the randomized algorithm and output the result of the randomized algorithm. By basically the same argument as in a), the probability that the correct minimum cut size is computed in a single run is lower bounded by

$$\frac{n-3}{n} \cdot \frac{n-4}{n-1} \cdot \ldots \cdot \frac{t-3}{t} \cdot 0.5 = \Omega(t^3/n^3),$$

and generally the output is always at least as large as the minimum cut size. Hence, running N independent runs of the procedure and outputting the minimum results in the correct minimum cut size with probability at least

$$1 - (1 - \Omega(t^3/n^3))^N \geq 1 - e^{-\Omega(t^3/n^3) \cdot N}.$$

Thus, there exists an $N = \Theta(n^3/t^3)$ such that our new algorithm succeeds with probability at least $1/2$. Hence, the overall runtime of the procedure is

$$N \cdot O(n^3 + t^\alpha) = O(n^6/t^3 + n^3 t^{\alpha-3}).$$

Now, we can choose t such that we minimize the runtime. We can do that by solving $n^6/t^3 = n^3 t^{\alpha-3}$ for t, which leads to $t = n^{3/\alpha}$. Note that here it is important that $\alpha > 3$, as if $\alpha < 3$, then t would be greater than n, which does not make any sense. The resulting runtime we obtain in this case is $O(n^3 \cdot n^{\frac{\alpha-3}{\alpha}}) = O(n^{3+\frac{3(\alpha-3)}{\alpha}})$, as desired.

(f) We first prove by induction that $a_i \geq 3$ for all $i \in \mathbb{N}_0$. We trivially have $a_0 \geq 3$ and moreover, given that $a_i \geq 3$, we also have

$$a_{i+1} = 3 + \frac{3(a_i - 3)}{a_i} \geq 3 + 0 = 3.$$

From $a_i \geq 3$ for all $i \in \mathbb{N}_0$, we can also directly deduce that the sequence $(a_i)_{i\geq 0}$ is decreasing monotonically, as

$$a_{i+1} = 3 + \frac{3(a_i - 3)}{a_i} \leq 3 + \frac{3(a_i - 3)}{3} = a_i.$$

As the sequence is bounded from below and monotonically decreasing, a look into the analysis script tells us that the sequence indeed converges and the limit is equal to the infimum of the sequence, denoted by a. For the sake of contradiction, assume that $a = 3 + \varepsilon$ for some $\varepsilon > 0$. Then, we obtain

$$a_{i+1} - a_i = 3 + \frac{3(a_i - 3)}{a_i} - a_i = \frac{6a_i - 9 - a_i^2}{a_i} = \frac{-(a_i - 3)^2}{a_i} \leq -\frac{\varepsilon^2}{6}.$$

This would imply that $|a_{i+1} - a_i| \geq \frac{\varepsilon^2}{6}$ for all $i \in \mathbb{N}$, a contradiction. Hence, we can conclude that $a = 3$ is the limit.

(g) Exercise c) together with a simple induction implies that there exists a randomized algorithm with a running time of $O(n^{a_i})$ for every $i \in \mathbb{N}$. As the limit of the sequence $(a_i)_{i\geq 0}$ is 3, for every $\varepsilon > 0$, there exists an $i_\varepsilon \in \mathbb{N}$ with $a_{i_\varepsilon} \leq 3 + \varepsilon$. Hence, for every $\varepsilon > 0$, there exists a randomized algorithm with a running time of $O(n^{3+\varepsilon})$. However, as $a_i \neq 3$ for every $i \in \mathbb{N}$, we cannot infer that there exists a randomized algorithm with a running time of $O(n^3)$.

Solution 3

(a) In each recursion level, at most $n - 1$ comparisons are performed and hence the total number of comparisons during an execution with maximum recursion depth $O(\log n)$ is bounded by $O(n \log n)$.

(b) First, assume that $X \geq \log_{3/2}(n)$. This implies that either x participates in at most $100 \log_{3/2}(n) = O(\log n)$ recursive calls, or among the first $100 \log_{3/2}(n)$ recursive calls, there are $\log_{3/2}(n)$ distinct recursion levels that reduce the problem size of the subproblem that contains x by at least a $(2/3)$-factor. This implies that after $100 \log_{3/2}(n)$ recursion levels, the subproblem that contains x contains at most

$$n \cdot (2/3)^{\log_{3/2}(n)} = 1$$

elements. Hence, it only consists of x. Therefore, also in this case x participates in at most $O(\log n)$ recursion levels. Now, it remains to argue that $\Pr[X < \log_{3/2}(n)] = O(1/n^2)$. We do this by using the Chernoff bound variant stated at the top of the special assignment (Theorem 2). Let $i \in [n]$ be arbitrary and $x_0, x_1, \ldots, x_{i-1} \in \{0, 1\}$ be arbitrary. We have

$$\mathbb{E}[X_i | X_0 = x_0, X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] \geq 1/3.$$

To see why, assume that we not only condition on the event $X_0 = x_0, X_1 = x_1, \ldots, X_{i-1} = x_{i-1}$, but we additionally condition on all the randomness, i.e. the concrete pivot choices, for the recursion levels 0 up to $i - 1$. Fixing the pivot choices for those recursion levels, we either deterministically obtain the guarantee that x participates in at most $i - 1$ recursion levels, or the size of the subproblem that x is part of in the i-th recursion level is k, for some fixed value

$k \in [n]$. Now, with probability at least $1/3$, among the k elements, we choose a pivot p in the "middle third", i.e. such that at least $\lceil k/3 - 1\rceil$ elements are strictly smaller than p and at least $\lceil k/3 - 1\rceil$ elements are strictly larger than p. If we indeed choose such a pivot, then the problem size of the problem that contains x drops indeed by at least a $(2/3)$-factor, as desired. Now, let $\delta = 1/2$. We get

$$\Pr[X \le \log_{3/2}(n)] \le \Pr[X \le (1-\delta)100\log_{3/2}(n)(1/3)] \le e^{-(1/2)^2 \cdot 100\log_{3/2}(n)\cdot(1/3)} = O(1/n^2),$$

as desired.

(c) This follows from exercise b) together with a Union Bound. More precisely, for each element x, let B_x denote the event that x participates in $\omega(\log n)$ recursive calls. Exercise b) implies that $\Pr[B_x] = O(1/n^2)$. Hence, we obtain

$$\Pr[\text{"Recursion depth is bounded by } O(\log n)\text{"}] = 1 - \Pr\left[\bigcup_x B_x\right] \ge 1 - \sum_x \Pr[B_x] = 1 - O(1/n).$$

(d) According to exercise c), the maximum recursion depth is $O(\log n)$ with probability $1-O(1/n)$. In that case, exercise a) implies that quicksort performs $O(n\log n)$ comparisons.

Solution 4

Let ℓ denote the non-vertical query line. We start by computing the two tangents of C parallel to ℓ in $O(\log n)$ time, as seen in class. Let ℓ_{upper} be the upper tangent and ℓ_{lower} be the lower tangent. If ℓ lies above ℓ_{upper}, then our algorithm can simply return the precomputed area of the whole polygon. If ℓ lies below ℓ_{lower}, then the algorithm simply returns zero. Hence, it remains to consider the case where ℓ lies strictly between ℓ_{lower} and ℓ_{upper}. Let $i_{lower} \in \{0, 1, \ldots, n-1\}$ such that the vertex $p_{i_{lower}}$ intersects the lower tangent and $i_{upper} \in \{0, 1, \ldots, n-1\}$ such that the vertex $p_{i_{upper}}$ intersects the upper tangent. Now, imagine you start from $p_{i_{lower}}$ and you go in ccw-direction along the edges of the polygon until you reach $p_{i_{upper}}$. As $p_{i_{lower}}$ lies below ℓ and $p_{i_{upper}}$ lies above ℓ, one encounters two consecutive vertices p_k and $p_{(k+1) \bmod n}$ for some $k \in \{0, 1, \ldots, n-1\}$ such that p_k lies below ℓ and $p_{(k+1) \bmod n}$ lies above ℓ. Given that we stored the points in ccw-order in an arrary, we can actually find p_k and $p_{(k+1) \bmod n}$ in $O(\log n)$ time with a binary search. The rough idea is the following. In the beginning, p_k could be any vertex between $p_{i_{lower}}$ and $p_{i_{upper}}$ (considering the ccw-direction). Now, let q denote the vertex directly in-between $p_{i_{lower}}$ and $p_{i_{upper}}$. Then, depending on whether q lies above or below the query line ℓ, we can either rule out the upper or the lower half of the vertices and therefore recurse on the remaining vertices. As the number of potential values for k roughly decreases by a factor of $1/2$, we can find p_k in $O(\log n)$ time. Now, similarly, starting from $p_{i_{upper}}$ and going in ccw-direction along the edges until one arrives at $p_{i_{lower}}$, one encounters two consecutive vetices p_t and $p_{(t+1) \bmod n}$ for some $t \in \{0, 1, \ldots, n-1\}$ such that p_t is above ℓ and $p_{(t+1) \bmod n}$ is below ℓ. Similarly as before, we can do a binary search to find p_t in $O(\log n)$ time. Let q denote the point on the line segment between p_k and $p_{(k+1) \bmod n}$ that intersects ℓ and r denote the point on the line segment between p_t and that intersects ℓ. The area below the query line ℓ can now be computed as the sum of two terms. The first term is the area of the convex polygon with vertices q, r, $p_{(t+1) \bmod n}$ and p_k. The second term is the area of the convex polygon $C[(t+1) \bmod n, k]$, where for every $i, j \in \{0, 1, \ldots, n-1\}$, we define $C[i, j]$ as the convex polygon with vertices $p_i, p_{(i+1) \bmod n}, p_{(i+2) \bmod n}, \ldots, p_j$. Note that the area of the first polygon can be computed in $O(1)$ time, as it only consists of four vertices. Furthermore, we can precompute the area of $C[i, j]$ for every $i, j \in \{0, 1, \ldots, n-1\}$ and hence we can look-up the area of $C[(t+1) \mod n, k]$ in $O(1)$ time.

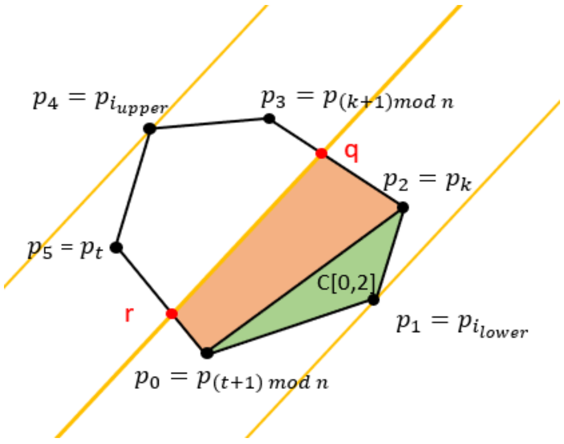

Figure 1: The figure illustrates the case where the query line lies between the two tangent lines.