COS 226	Algorithms and Data Structures	Spring 2003

Final Solutions

1. **Analysis of algorithms.**

   There exists a constant $c > 0$ such that *for any* input of size N, the algorithm takes *at most* cN steps.

   This is equivalent to Definition 2.1 in Sedgewick. Less mathematical definitions were also accepted as long as they addressed the meaning of *worst-case* and *big-Oh*.

2. **String searching.**

   (a) For X's choice A, brute-force is a factor of 1 faster than right-left.

   (b) For Y's choice C, right-left is a factor of M faster than brute-force.

   We assume $M \leq N$. For random bitstrings, both algorithms have an average case running time of $\Theta(N)$. For random ASCII characters, brute force takes $\Theta(N)$ time on average, while right-to-left scan can be no more than a factor of M or 128 faster. For input type C, brute force takes $\Theta(MN)$ time, while right-to-left scan takes only $\Theta(N)$ time on average.

3. **Geometric algorithms. (8 points)**

   (a) iv. $O(N \log N)$

   First determine whether there are any intersections among the N line segments that make up the two polygons using S1. If there are, then A cannot be completed inside B. If there are no intersections, then this either A is inside B, B is inside A, or A and B are disjoint. Second, pick a point p in A and use S3 to determine whether p is in B. If so, then A is inside B.

   (b) iii. $O(N)$

   Use the Graham scan algorithm to compute the convex hull of the N points. If the convex hull is equal to B, then A is inside B; otherwise A is not inside B. We awarded significant partial if you stopped here and claimed an $O(N \log N)$ algorithm by using a convex hull algorithm as a black box.

   However, sometimes you can design faster algorithms by looking inside the black box. The Graham scan convex hull algorithm actually runs in linear time, except for the initial phase where you sort by angle. Here, we can sort in linear time since the points in A are sorted and the points in B are sorted. We can merge two sorted lists into a sorted whole in linear time ala merge sort.

4. **Minimum spanning tree.**

   (a) `1-6 6-5 5-7 5-8 8-4 1-2 2-3`

   (b) `5-7 4-8 1-6 5-6 5-8 1-2 2-3`

5. **Max flow, min cut.**

   (a) `s-4-7-3-2-5-t`

   (b) 27

   (c) Yes, it's optimal. The min cut is {s, 3, 4, 7} has capacity = 10 + 7 + 10 = 27.

6. **Data compression.**

   A Huffman encoding for a message produces an encoding that uses the fewest bits among any prefix free code. The 2-bit binary code a = 00, c = 01, g = 10, t = 11 is a prefix free code that uses $21 \times 2 = 42$ bits. Thus, a Huffman code uses will use fewer than 43 bits.

7. **Algorithmic design.**

   Use Kruskal's algorithm. The algorithm takes $O(E \log^* V)$ time plus the time for sorting. Since all of the weights are between 1 and V, we can sort in $O(V)$ time using key-indexed counting.

8. **Theory vs. practice.**

(a) Sorting:	I. Insertion sort	`II.`	Worst-case
	II. Mergesort	`II.`	Practice
(b) Sorting:	I. Quicksort	`II.`	Worst-case
	II. Mergesort	`I.`	Practice
(c) Min spanning tree:	I. Prim with binary heap	`II.`	Worst-case
	II. Prim with Fibonacci heap	`I.`	Practice
(d) Priority queue:	I. Binary search tree	`II.`	Worst-case
	II. Binary heap	`II.`	Practice
(e) Linear programming:	I. Ellipsoid algorithm	`I.`	Worst-case
	II. Simplex algorithm	`II.`	Practice
(f) Convex hull:	I. Package wrap	`II.`	Worst-case
	II. Graham scan	`II.`	Practice

9. **Choosing the right algorithms and data structures.**

   (a) i

   (b) ii

   (c) iv

   (d) ii

10. **Linear programming.**

$$
\begin{array}{lrcrcrcrcl}
\text{minimize} & 26A & + & 30B & + & 20C & + & 8000M & & \\
\text{subject to:} & A & + & B & + & C & + & M & = & 1 \\
 & 0.45A & + & 0.50B & + & 0.40C & + & 100M & \geq & 0.45 \\
 & 4A & + & 1B & + & 0.6C & & & \geq & 3.25 \\
 & 4A & + & 1B & + & 0.6C & & & \leq & 5.50 \\
 & A & , & B & , & C & , & M & \geq & 0
\end{array}
$$

You could multiply through by 10 or 100 to end up with integers, but that's not necessary. We note that the $\leq$ constraint is actually redundant since no raw material is more than 5.5% silicon anyway. this example. Borrowed from Bradley, Hax, and Magnanti.

11. **Reductions.**

(a) To solve the global min cut problem, replace each undirected edge with two anti-parallel directed edges. Then for each pair of vertices s and t, find a min cut separating s and t using the Ford-Fulkerson algorithm, say in $O(E^2V)$ time using the shortest augmenting path rule. Return the cut with the smallest capacity among all the pairs that you try. The overall running time is $O(E^2V^3)$.

In fact, it suffices to calculate the min cut separting s and t for one fixed s and $V-1$ choices of t. This follows since the graph is undirected and s must lie on one side of the cut or the other. This speeds up the algorithm above by a factor of V. Interestingly, there are deterministic algorithms that solve the global min cut problem in $O(EV + V^2 \log V)$ time.

(b) i, ii, iii, iv

If you polynomial reduce A to B, this means that (i) if B is solvable in polynomial time, then so is A. It is not a refined enough concept to differentiate between linear and quadratic algorithms. For that, you need linear time reductions.