I ACCEPT NO RESPONSIBILITY FOR ERRORS ON THESE CRIB SHEETS
PLEASE TAKE THE CRIB SHEETS WITH YOU ON THE WAY OUT.

Data structures

- Binary heaps are implemented using a heap-ordered balanced binary tree. Binomial heaps use a collection of B_k trees, each of size 2^k. Fibonacci heaps use a collection of trees with properties a bit like B_k trees. (The operation HEAPIFY below makes a heap with n elements without doing n INSERTs.)

Procedure	Binary heap (worst case)	Binomial heap (worst case)	Fibonacci heap (amortized)
MAKE-HEAP	$O(1)$	$O(1)$	$O(1)$
HEAPIFY	$O(n)$	$O(n)$	$O(n)$
INSERT	$O(\lg n)$	$O(\lg n)$	$O(1)$
MINIMUM	$O(1)$	$O(\lg n)$	$O(1)$
EXTRACT-MIN	$O(\lg n)$	$O(\lg n)$	$O(\lg n)$
UNION	$O(n)$	$O(\lg n)$	$O(1)$
DECREASE-KEY	$O(\lg n)$	$O(\lg n)$	$O(1)$
DELETE	$O(\lg n)$	$O(\lg n)$	$O(\lg n)$

- Binary search trees implement all the operations of heaps (except UNION) in addition to SEARCH. Virtually all the operations take time $\Theta(\log n)$.
- The union-find data structure implements the operations UNION$(x, y, \texttt{label})$ and $\texttt{label}\leftarrow$FIND(x) on a collection of disjoint sets. Initially (before any UNION operation) each of n elements is in its own set of size one. A total of m disjoint set operations take time $O(m\alpha(m, n))$, where $\alpha(m, n)$ is a ridiculously slowly growing function. $\alpha(m, n)$ is $o(\log^*(m))$, $o(\log m)$, and $\Omega(1)$.

Sorting

- For comparison-based sorting algorithms, Heapsort and mergesort take time $O(n \log n)$, and quicksort takes expected time $O(n \log n)$. An information theoretic lower bound for any comparison based sort is $\log(n!) = \Omega(n \log n)$.
- For n numbers known to fall within a range $\{1, \ldots, N\}$, radix sort will take time $O(n + N)$. Linear time algorithms are often possible if more can be known about the numbers besides how to pairwise compare them.
- Order statistics (the k^{th} largest element of n elements, or the median of n elements) can be found in time $\Theta(n)$ by a comparison based algorithm. The algorithm chooses a pivot by recursively computing the median of the medians of $n/5$ subsets of 5 elements each. Once all elements are compared to the pivot, 1/4 of the elements can be discarded, as they are all known to be greater than (or, less than) the k^{th} largest element.

Exploring graphs

- Breadth first search (BFS) takes $O(E)$ time and finds shortest paths.
- Depth first search (DFS) takes $O(E)$ time. Also in this time you can have preorder numberings (or discover times), postorder numberings (or finish times), classification of edges as forward, back, back-cross or back-cross-tree edges.
- A topological sort of a dag can be found in $O(E)$ time by reversing the postorder numbers in a DFS.
- Strongly connected components (SCC's) can be found in $O(E)$ time. Note that the component graph, G^{SCC} is acyclic, so you can topologically sort it.

Minimum Spanning Trees (MST's)

- If $A \subset E$ is *part of* a MST, and S is a cut which no edge in A crosses, then the minimum edge across the cut can be added to A to yield part of a MST.
- Kruskal's grows a collection of trees by always adding the cheapest edge which connects two trees, taking time $O(E \lg V)$ to sort the edges.
- Prim's algorithm grows a single tree from a vertex by always adding the cheapest edge out from the tree, taking time $O(E + V \lg V)$ if a Fibonacci heap is used.

Shortest path problems

- Single-source shortest paths for non-negative edge weights can be found by Dijstra's algorithm. Like Prim's, grow a tree, always adding the vertex with the cheapest path from the source by extending the tree by only one edge. Takes $O(E + V \lg V)$ using a Fibonacci heap.
- Single-source shortest paths for edge weights which may be negative can be found using Bellman-Ford. Make V passes over the graph, updating shortest path estimates to each vertex *relaxing edges*. Either a negative cycle will be found or a shortest path tree in time $O(EV)$.
- All-pairs shortest path problems had a few algorithms:
  - Matrix-multiply like algorithms taking time $O(V^4)$ or $O(V^3 \log V)$.
  - Floyd Warshall takes time time $O(V^3)$. They use dynamic programming to solve

    $$d_{ij}^{(k)} = \text{the shortest path from } v_i \text{ to } v_j \text{ using paths going } \textit{through } \{v_1, \ldots, v_k\}$$

  - Johnson's algorithm first solves a single source shortest path problem from an added vertex, s, (with edges (s, v), $v \in V$ of weight 0) to reweight the edges by:

    $$\hat{w} = w(u, v) + \delta(s, u) - \delta(s, v)$$

    This results in positive edge weights, and Bellman-Ford can be used to take time $O(EV)$.

Linear programming

- In linear programming, the goal is to optimize a linear objective function subject to linear inequality constraints. No algorithm were discussed, but polynomial time algorithms exist for linear programming.
- In integer linear programming, the goal is to find an integer solution to a linear programming problem. No polynomial time algorithm is known nor is likely to exist for this NP-complete variant.

Flow networks and maximum flows

- Capacities satisfy $c(u, v) \geq 0$. Flows satisfy

  Capacity constraints : $f(u, v) \leq c(u, v)$
  Skew symmetry: $f(u, v) = -f(v, u)$
  Flow conservation: $\forall u \in V - \{s, t\} : \sum_{u \in V} f(u, v) = 0$
- The residual capacities are given by $c_f(u, v) = c(u, v) - f(u, v)$.
- The min-cut max-flow theorem proves the maximum flow is equal to the minimum capacity over all cuts. Further, if there are no augmenting paths in the residual graph, a maximum flow has been obtained.
- Ford-Fulkerson finds paths from s to t in the residual graph to augment the flows until no more paths can be found, taking time $O(Ef^*)$, where f^* is the value of the max-flow.
- Edmonds-Karp improves on this by always choosing the shortest augmenting path (i.e., fewest edges), finding the max-flow in time $O(VE^2)$.

Number theoretic algorithms Throughout, define β to be the length of bits in all the numbers involved.

- The greatest common divisor $\gcd(a, b) = \gcd(b, a \bmod b)$, yielding Euclid's algorithm taking $O(\beta)$ arithmetic operations.
- If $\gcd(a, b) = d$ then there is an x and y so that $ax + by = d$. Euclid's algorithm can be adjusted to calculated x and y efficiently.
- (Chinese Remainder Theorem) If $\gcd(n_1, n_2) = 1$ and $n = n_1 n_2$, then there is a one-to-one mapping between numbers $a \bmod n$ and pairs $(a_1 \bmod n_1, a_2 \bmod n_2)$ so that $a_i = a \bmod n_i$. To compute a, find x and y so that $n_1 x + n_2 y = 1$ and notice that $(1, 0)$ maps to $n_2 y \bmod n$ and $(0, 1)$ maps to $n_1 x \bmod n$. So, (a_1, a_2) maps to $a_1 n_2 y + a_2 n_1 x \bmod n$.
- (Fermat's Little Theorem) For p prime, $1 \leq a < p$, $a^{p-1} \equiv 1 \pmod{p}$.
- A pseudo-prime test is to check if $2^{n-1} \equiv 1 \pmod{n}$; if yes output "prime?", otherwise output "composite!". Very few composites look like primes. A randomized primality test chooses k random values of a in the range $1 \leq a < n$. For each, calculate if $a^{\frac{n-1}{2}} \equiv \pm 1 \pmod{n}$. If one is not ± 1 output "composite!". If all are 1 output "composite?". Otherwise output "prime?". This test fails with probability $\leq \frac{1}{2^k}$.
- In the *RSA public-key cryptosystem,* a participant creates her public and private keys with the following precedure.
  1. Select at random two large prime numbers p and q.
  2. Compute n by the equations $n = pq$.
  3. Select a small odd integer e that is relatively prime to $\phi(n) = (p-1)(q-1)$.
  4. Compute d as the multiplicative inverse of $e \bmod \phi(n)$. (I.e., $de \equiv 1 \pmod{\phi(n)}$.)
  5. Publish the pair $P = (e, n)$ as her *RSA public key.*
  6. Keep secret the pair $S = (d, n)$ as her *RSA secret key.*

  To encode message M, compute $M^e \bmod n$. To decode cybertext C, compute $C^d \bmod n$

CS-170 Final May 18, 1995
David Wolfe

You may use the backs of pages if you need more space, but please indicate for the grader you've done so: For example, "Continued on back of page 4". Please tear off the crib sheets, but leave the rest of the exam stapled.

1.	Multiple choice and True-False	/30
2.	Compare primality tests	/15
3.	Compare heap structures	/15
4.	Deduce DFS forest	/30
5.	Ford-Fulkerson exam problem	/30
6.	Arbitrage homework problem	/30
7.	RSA computation	/40
8.	Bound recurrence	/20
9.	MAXIMUM weight spanning tree	/20
10.	A number above median	/40
11.	Minimum spanning tree problem	/50
Total	(Extra points possible)	/115

1. (30 points) Check one box for each of the following.

	ALWAYS TRUE	SOMETIMES TRUE	NEVER TRUE
In a connected graph, $E = O(V)$			
In a connected graph, $E = \Omega(V)$			
If a directed graph has a depth first forest with no back edges then the graph has n strongly connected componenents			
For a fixed graph G, the Edmonds-Karp algorithm does at least as well as Ford-Fulkerson			

	TRUE	FALSE
$\log^*(m) = O(\alpha(m, n))$		
For data encryption to be secure against eavesdroppers, two parties must meet in a secure environment to agree on secret keys.		
A heap can be designed which takes $O(n)$ per INSERT and $O(1)$ per DELETE-MIN		
A heap can be designed which takes $O(1)$ per INSERT and $O(1)$ per DELETE-MIN		
In a Fibonacci heap, we could assign $O(1)$ amortized cost per DELETEMIN and per DELETE, as long we assign $O(\log n)$ amortized cost per INSERT.		

2. (15 points) Alice is deciding between the pseudo-prime test and the randomized primality test.

   (a) What is the strongest reason you can think of to use a pseudo-prime test?

   (b) What is the best reason you can think of to use the randomized primality test?

3. (15 points) Hubert is deciding between using binary heaps or Fibonacci heaps.

   (a) What is the strongest reason you can think of to use a binary heaps?

   (b) One good reason for Fibonacci heaps?

4. (30 points) A depth-first search is performed on a graph whose vertices are $V = \{A, B, C \ldots, L\}$. The vertices listed in order of their preorder-numbers (or in order of their discover times) is:

   $$\{A, B, C, D, E, F, G, H, I, J, K, L\}$$

   The vertices listed in order of their postorder-numbers (or finish-times) is:

   $$\{C, D, E, B, H, I, G, F, A, K, L, J\}$$

   Draw all the trees in the depth-first search forest.

5. (30 points) Let $G = (V, E)$ be a flow network with source s, sink t, and suppose each edge $e \in E$ has capacity $c(e) = 1$. Assume also, for convenience, that $|E| = \Omega(V)$.

   (a) Suppose we implement the Ford-Fulkerson maximum-flow algorithm by using depth-first search to find augmenting paths in the residual graph. In terms of V and E only, what is the worst case running time of this algorithm on G?

   (b) Suppose the maximum flow for G has been computed, and a new edge e with unit capacity (i.e., $c(e) = 1$) is added to E. Describe how the maximum flow can be efficiently updated. (Note: It is not the value of the flow that must be updated, but the flow itself.) Analyze your algorithm.

6. (30 points) Suppose that we are given n currencies, $c_1, \ldots, c_n$, and an $n \times n$ table R of exchange rates such that one unit of currency c_i buys $R[i, j]$ units of currency c_j. Give an efficient algorithm to determine whether or not there exists a sequence of currencies $c_{i_1}, c_{i_2}, \ldots, c_{i_n}$ such that

$$R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1] > 1$$

(This would mean there is an arbitrage opportunity to turn one unit of c_1 into more than one unit by just exchanging currencies.) Analyze the running time of your algorithm.

7. (40 points) For this problem, **all answers should be actual numbers** (not just expressions), and you should show your work. No calculator is permitted. You will not lose credit for a little arithmetic error as long as I can follow your work.

   Bob wants to send Alice the message $M = 5$ using RSA. Bob knows Alice's public key is $(e, n) = (7, 33)$.

   (a) (10 points) What other public keys could Alice have chosen given her choice of $n = 33$?

   (b) (10 points) What is the encrypted message that Alice receives?

   (c) (15 points) What is Alice's secret key that she uses for decryption.

   (d) (5 points) Decrypt Bob's message as Alice would using her secret key.

   If, after all calculations, you think you have made an arithmetic error in parts a-c, admit why you think you have, but don't spend time correcting the error.

8. (20 points) Consider the following recurrence relation:

$$\begin{aligned} T(n) &= 2T(3n/4) + n \\ T(n) &= 1 \qquad (\text{for } n \leq 2) \end{aligned}$$

Prove that $T(n) = \Omega(n^2)$

9. (20 points) Give an efficient algorithm to find the MAXIMUM weight spanning tree of a graph $G(V, E)$. Determine your algorithm's running time. (I will not specify the running time you should be looking for.)

10. (40 points) Given a list of n numbers $(a_1, \ldots, a_n)$, you want to find any number in that list greater than the MEDIAN.

    (a) Give an upper bound on the exact number of comparisons required (i.e., O(n) is not an appropriate answer) by describing an algorithm and analyzing it.

    (b) Give a lower bound on the absolute number of necessary comparisons.

11. (50 points)

    (a) (25 points) For a weighted graph G, prove that the maximum weight edge in a cycle is not in a minimum spanning tree of G. You will receive no credit if your proof is unclear.

    (b) (25 points) Given a connected graph with exactly $V + 10$ edges, find its minimum spanning tree in $O(V)$ time. (You may use the result in part (a), even if you get part (a) wrong.)

MORE SPACE IF REQUIRED

MORE SPACE IF REQUIRED

MORE SPACE IF REQUIRED