College of Computer & Information Science
Northeastern University
CS5800: Algorithms

Spring 2011
Handout 18
23 March 2011

Midterm

Instructions.

- **Duration, Problems, and Points:** There are four problems, for a total of 40 points. Read all the problems through first and attack them in the order that allows you to make the most progress.
- **Presentation of solutions:** While describing an algorithm, you may use any of the algorithms covered in class or in the text as a subroutine, without elaboration.
- **A note on grading:** Your grade on any problem will be determined on the basis of the correctness of your solution and the clarity of the description. In case you are not able to present an algorithm that has the desired properties, give the best algorithm you have designed. Show your work, as partial credit may be given. If you need extra space, use the blank facing page.
- Best of luck!

Question	Score	Maximum
1		10
2		8
3		10
4		12
Total		40

Name: _____

Problem 1. (10 points) Three-way mergesort

Consider the following variant of Mergesort. If the array has more than two elements, it recursively sorts the first one-third, the middle one-third, and the last one-third. Then, it merges the first and middle one-thirds, and finally merges the first two-third with the last one-third. If the array has at most two elements, then it trivially sorts the array. For completeness, here is the pseudocode for sorting the array $A[\ell \ldots r]$.

```
3-WAY-MERGESORT(A, ℓ, r):
if r − ℓ + 1 > 2:
  m ← ⌊(r − ℓ + 1)/3⌋
  3-WAY-MERGESORT(A, ℓ, ℓ + m − 1)          [Sort first one-third]
  3-WAY-MERGESORT(A, ℓ + m, ℓ + 2m − 1)     [Sort middle one-third]
  3-WAY-MERGESORT(A, ℓ + 2m, r)             [Sort last one-third]
  MERGE(A, ℓ, ℓ + m − 1, ℓ + 2m − 1)        [Merge first and middle one-thirds]
  MERGE(A, ℓ, ℓ + 2m − 1, r)                [Merge first two-third with last one-third]
else:
  if A[ℓ] > A[r]:
    swap A[ℓ] ↔ A[r]
```

Write a recurrence relation for the worst-case running time of 3-WAY-MERGESORT. Solve the recurrence relation.

(*Note:* Assume that the worst-case running time for merging two lists is the sum of the lengths of the two lists. Thus, for instance, the running time of $\textsc{Merge}(A, x, y, z)$, which merges $A[x \ldots y]$ and $A[y + 1 \ldots z]$, is $z - x + 1$.)

Answer: The recurrence relation is the following.

$$T(n) = 3T(n/3) + 5n/3,$$

for $n \geq 3$ and $T(n) = \Theta(1)$ for $n = 1$.

Solving it by the recursion tree approach, we obtain a contribution of $5n/3$ at each level. We have $\log_3 n$ levels, so we obtain $T(n) = \Theta(n \log n)$.

Problem 2. (8 points) Perfect squares

Recall that we have efficient algorithms for integer arithmetic; we can add, subtract, multiply, and divide two n-bit integers in time $O(n)$, $O(n^2)$, and $O(n^2)$, respectively.

Give an algorithm to determine whether a given n-bit positive integer N is a *perfect square*; that is, N equals k^2 for some integer k. State the running time of your algoruthm.

The more efficient your algorithm is in terms of its worst case running time, the more credit you will get. There is no need to prove the correctness of the algorithm or prove your stated running time.

(*Note:* The "square-root" operation is unavailable in integer arithmetic. Use the division algorithm. Assume that the division algorithm takes a dividend and a divisor as input, and returns the quotient and remainder as output.)

Answer: We perform a binary search over the interval $[1..N]$ to determine whether there exists a k such that $k^2 = N$. The number of iterations is at most $\log N \leq n$, since N is an n-bit number. Each iteration takes $O(n^2)$ time since it involves one multiplication, one addition, one division by 2, and one comparison. So the total time is $O(n^3)$. Here is the pseudocode for completeness. The main call is to PERFECT-SQUARE$(N, 1, N)$.

```
PERFECT-SQUARE(N, i, j)
if i = j:
  if i^2 = N:
    return "yes"
  else return "no"
k ← ⌊(i + j)/2
if k^2 ≥ N:
  return PERFECT-SQUARE(N, i, k)
else:
  return PERFECT-SQUARE(N, k + 1, j)
```

Problem 3. (10 points) Identifying kingpins

You work for a public relations firm and have just acquired complete information on the Washington lobbying network. In particular, you have a list V of n persons (congressmen, bureaucrats, lobbyists, businessmen, etc.) and for each person i in V, a list of persons in V that i can *influence*. (Note that it is possible that i can influence j but j cannot influence i.)

Having just studied graph algorithms, you immediately capture the above information by a directed graph G with V as the set of vertices and the set E of edges defined as follows.

$$E = \{(i, j) : i \text{ can influence } j\}.$$

We call a person i a *kingpin* if for every other person $j \in V$, there is a path from i to j in G.

Give a polynomial-time algorithm to determine *all* kingpins in the given lobbying network. If there are no kingpins, then your algorithm must indicate so. State the running time of your algorithm. The more efficient your algorithm is in terms of its worst case running time, the more credit you will get.

Answer: One algorithm is to run a DFS in the lobbying network from each node (separately) to see whether all other nodes are visited (if so, then the start node is a kingpin). Running time is $O(|V|(|V| + |E))$.

A faster linear time algorithm is to compute the strongly connected components of the graph, determine a source component, and then return all the vertices in the component if this was a unique source in the DAG of SCCs. Running time is $O(|V| + |E|)$.

Problem 4. ($3 \times 4 = 12$ points) Minimum spanning trees and shortest paths

For each of the following statements, indicate whether it is true or false. If you claim that the statement is true, give a brief proof; otherwise, give a counterexample.

(a) True or False: If T is a minimum spanning tree of an undirected graph G with positive weights on edges, then the unique path connecting any two vertices u and v in T is a shortest path between u and v in G.

Answer: False. Consider a graph with four vertices u, v, w, and x. We have edges (u, w), (w, x), (x, v), each of weight 1, and edge (u, v) of weight 2. There is only one MST with the edges (u, w), (w, x), and (x, v). The unique path frmom u to v in the MST is of length 3. The shortest path between u and v in the graph is of length 2.

.

(b) Let G be a directed graph with integer edge weights (possibly negative). Let P be a shortest path from s to t in G.
True or False: If the weight of every edge in G is doubled (i.e., multiplied by 2), then P remains a shortest path from s to t.

Answer: True. Let G' be the new graph. For any path P, the weight of P in G is exactly twice the weight of P in G'. So a path P from s to t is shortest in G if and only if it is also the shortest in G'.

(c) Let T be a minimum spanning tree of an undirected graph G with positive weights on edges. Let (u, v) be an edge in T, and let S be a subset of V such that $u \in S$ and $v \notin S$.
True or False: The weight of (u, v) is the minimum among all edges that have one endpoint in S and the other endpoint in $V - S$.

Answer: False. Consider a graph G with four vertices u, v, w, and x with edges (u, v), (v, w), (u, w) of weight 1 and edge (u, x) of weight 2. Every MST contains the edge (u, x). Consider the cut where S equals $\{u\}$. The weight of (u, x) is 2 while that of every other edge crossing the cut is 1.