Due: beginning of class on Thu. June 6, 2002

Important notes:

- Answers to ALL problems must include a proof of correctness of the algorithms. Depending on the problem and your specific solution, the proof may be easy, and consists of a simple English explanation. If correctness is highly non-trivial, then a more rigorous mathematical argument is required.

- Algorithms and theorems covered in class or in the relevant portion of the textbook can be used without proofs, if an appropriate citation is included. However, you should not use material that is not going to be covered, or has not been covered yet in the course. E.g., reducing a problem to linear programming and invoking a polynomial time solution to linear programming is not an acceptable solution.

- You can use other textbooks as general study tools, but not as tools to solve specific problems. In particular, searching books or the web for problems similar (or identical) to those assigned, is not acceptable. If you are using study tools other than the textbook, occasionally, you may find the solution to problems similar to those assigned. In any case you should write your solution without using such references. Moreover, if you are using any of the ideas you found in the reference you should give proper credit (including citation of book or webpage).

- You are allowed to talk to other students about the problems, but solutions should be written individually. Moreover, if you talked to any student about the problems, you should acknowledge that in your solution. E.g., if A and B talked about problem 2, then A should write “I discussed problem 2 with B”, and B should write “I discussed with A”. I expect the “discussion graph” to be symmetric, e.g., if A and B talked about some problem, I expect to find an acknowledgment note on both homeworks. (If you forgot to write a proper acknowledgment in a previous homework, please report to the instructor, and explain why you didn’t do so.)

Problem 1

Problem 17-2 from the textbook [Making binary search dynamic] Parts (a) and (b). (Part (c) is not required.)

Notice: part be requires the analysis of both the worst case and amortized running time. You can use any of the methods described in the book to analyze the amortized running time. The important thing is that you provide a precise and clear analysis.
Problem 2

Problem 24-3 from the textbook [Arbitrage] Parts (a) and (b).

This is a classic problem, and you might be able to find a solution in other textbooks or the web. Please refrain from searching for such a solution, instead of solving the problem on your own.

Problem 3

In this problem you are asked to design a graph data structure supporting the following operations:

- **CREATE(n)**: create an empty directed graph with \( n \) nodes and no edges.
- **INSERT(e)**: insert edge \( e \) to the current graph. If edge \( e \) is already present, then the graph is not modified.
- **COUNT(i,j)**: return the number of directed paths in the graph from node \( i \) to node \( j \). Notice that this number can be 0 (if nodes are not connected) or \( \infty \) (if the graph contains cycles).

You should describe to different ways to implement this data structure, as follows:

**Part a.** Assume COUNT queries are much more frequent than INSERT. In this case, you want to make COUNT run as fast as possible, possibly at the expense of the insertion running time. Give a solution where \( \text{COUNT} \) takes \( O(1) \) time, and \( \text{INSERT} \) is as fast as possible. (I can get \( O(n^2) \) time, but maybe you can do better.)

**Part b.** Now consider the opposite situation: you want INSERT to take \( O(1) \), and make COUNT as fast as possible. (The best I can do is \( O(m) \) time, where \( m \) is the number of edges in the current graph.)

In both cases, you should provide a clear description of all three procedures (CREATE, INSERT, COUNT), a proof of correctness of the algorithm, and analysis of the worst case running times. Running times can be a function of the initial number of nodes \( n \), as well the number \( m \) of edges in the current graph at the time the operation is invoked.