COMPSCI330 Design and Analysis of Algorithms
Final Exam

Guidelines

- **Describing Algorithms** If you are asked to provide an algorithm, you should clearly define each step of the procedure, establish its correctness, and then analyze its overall running time. There is no need to write pseudo-code; an unambiguous description of your algorithm in plain text will suffice. If the running time of your algorithm is worse than the suggested running time, you might receive partial credits.
- **Timing** Exam starts at 2:00 pm and ends at 5:00 pm.
- **Extra Problem** Problem 6 is an extra problem. It is significantly harder than the other problems and no partial credit will be given. Please do not attempt this problem before you are reasonably confident with other problems.

Name: _____ Duke ID: _____

For grading use only:

Problem	1	2	3	4	5	6*	Total
Score							

Problem 1 (New Recursions)**.** (20 points) Consider the following algorithm that is similar to quick sort and quick selections. In particular, it also uses a pivot to partition the array into two parts, where the left part contains numbers smaller than the pivot and the right part contains numbers larger than the pivot. After recursing on both subarrays, the algorithm uses a MERGE step on the left and right part. See the description below:

```
ALG(A[])
    If length(A) == 1 Then return
    Pick a random pivot p in array A
    Partition A such that A[i] = p, A[1..i-1] contains elements smaller than p
         and A[i+1..n] contains elements larger than p
    ALG(A[1..i-1])
    ALG(A[i+1..n])
    MERGE(A[1..i-1], A[i+1..n])
```

Here the MERGE step takes two arrays as input, and the amount of times it takes is equal to the product of the lengths of the two arrays. Show the expected running time of this algorithm is Cn^2 for some constant C.

Hint: You can use the following formulas for summations:

$$\sum_{i=1}^{n}(n-k)(k-i) \approx n^3/6; \qquad \sum_{i=1}^{n} i^2 \approx n^3/3.$$

This page is intentionally left as blank.

Problem 2 (Finding Papers)**.** (20 points) Rong is not a very organized person. He is trying to look for a paper that he printed out some time ago, and he is staring at n drawers. Unfortunately, he does not remember which drawer did he put the paper in. He has some vague memory that allows him to estimate the probability p_i that there is a copy of the paper at drawer number i. Whether drawer i has a copy of the paper or not is completely independent of whether other drawers contain copies of the papers (again, Rong is not very organized and he might have printed several copies of the same paper and put them into different drawers). Given p_i for all i, please design an algorithm that outputs an ordering to search, so that the expected number of drawers opened before finding a copy of the paper is minimized. (If none of the drawers contain a copy of the paper Rong will just print the paper again, and in that case the number of drawers opened is n.) Make your algorithm as fast as you can and prove that it is optimal.

This page is intentionally left as blank.

Problem 3 (Balanced Colors)**.** Given a bipartite graph G with n vertices and m edges, we would like to color its vertices using two colors (red and blue) so that no two adjacent vertex have the same color. In addition, say the coloring is k-balanced if the number of red and blue vertices are both at most k. You need to design an algorithm that output the smallest possible k such that a k-balanced coloring exists for this graph G. As an example, in the graph below, we could color vertices A, E, F red, and B, C, D blue, that is a 3-balanced coloring. On the other hand, we can also color A, D as red, B, C, E, F as blue, but that is just a 4-balanced coloring and hence a worse solution. The correct output for this example should be 3.

Your algorithm should run in $O(n^2)$ time.

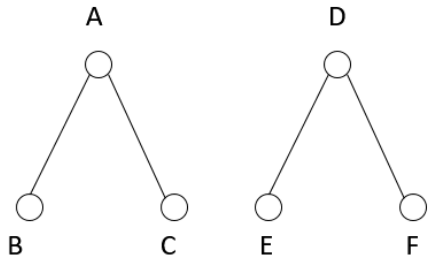

Figure 1: Example

This page is intentionally left as blank.

Problem 4 (Weighted Matching)**.** (a) (10 points) Consider the classroom assignment problem again where we try to assign courses to classrooms. We form a bipartite graph between courses and classrooms, such that if a course can be assigned to a classroom, there is an edge between them. In addition, the edges now have weights $0 \leq w_{i,j} \leq C$: a higher weight means the classroom is more suitable for the course. Write a linear program that finds a matching between courses and classrooms that maximizes the sum of weights of matching edges. (In this problem you do not need to prove the linear programming will return an integer solution.)

(b) (10 points) Suppose now we want to first maximize the number of courses that have classrooms, and after that maximize the sum of weights of matching edges. Describe how to modify the linear program above to solve this new problem. (As an example, suppose there are two courses x_1, x_2 and two classrooms y_1, y_2, the weight for $(x_1, y_1), (x_2, y_2)$ are both equal to 1, the weight for (x_1, y_2) is equal to 100 and (x_2, y_1) is not an edge. Then the solution to the previous problem will just take the edge (x_1, y_2), while the solution to this problem should take edges (x_1, y_1) and (x_2, y_2).) To get all 10 points you need to use a *single* linear program to solve this problem.

This page is intentionally left as blank.

Problem 5 (Colorful Path Revisited)**.** (a) (10 points) Consider the following problem called DISJOINT SETS: Given n subsets $S_1, S_2, ..., S_n$ of $\{1, 2, ..., m\}$, and an integer k, decide whether there are k sets out of these n sets that are mutually disjoint (no two sets among these k share a common element). Based on the fact that INDEPENDENT SET is NP-complete, show DISJOINT SETS is NP-hard. (Hint: Vertices in INDEPENDENT SET corresponds to sets in DISJOINT SETS)

(b) (10 points) Recall in the midterm we talked about the colorful path problem: given a graph G whose edges have one of q different colors, find a path from s to t such that consecutive edges do not have the same color. Now we will modify the problem slightly, and we would like to make sure ALL edges on the path from s to t have different color.

More formally, define COLORFUL PATH problem as: given a graph G whose edges are directed and have one of q different colors and vertices s, t, decide whether there is a path from s to t such that all edges along the path have different colors. Show that COLORFUL PATH is NP-hard. (Hint: You can use the fact that DISJOINT SETS is NP-hard even if you couldn't solve (a).)

This page is intentionally left as blank.

Problem 6 (Team Competition)**.** (20 points)
This is an extra problem. Please do not start thinking about it before you feel confident about other problems. Partial credit will not be given for this problem.

Sport X is usually played between two players (think of tennis/badminton/etc.), however it has a very special team format. In this format, each team has n players. A match between two teams involves a sequence of games. For every game, each team chooses one player simultaneously (they don't know which player the other team is going to choose) to play. The winning player can no longer play in the later games. The match ends when all the players in one of the teams have won a game (and that team is the winner).

As an example, if $n = 2$, the first team has players A, B, while the second team has players C, D. Say in the first game A wins against C. Then in game 2, player B must play, while the second team can choose between C and D. If B wins again then the first team wins the match; otherwise the match goes to game 3 and whoever wins that game wins the match.

Now you are the coach for one of the teams. You know the players very well and can predict the probability of game outcomes. This is given as an $n \times n$ matrix A, where $0 \leq A_{i,j} \leq 1$ is the probability that player i on your team wins, if he/she plays against player j on the other team. You know the opponent coach is also very smart and has access to the same matrix. Design an algorithm that computes the probability for your team to win using the best possible strategy. Your algorithm should run in time 2^{cn} for some constant c.

Note that the best possible strategy should be adaptive: you need to describe which player will play under every possible situation (every possible outcome of previous games). For example, for the first game you might ask player i to play with probability p_i. If in the first game your player 3 beats opponent's player 5, for the next game you might decide to ask player i to play with probability q_i (and $q_3 = 0$ because the third player in your team has already won a game).

This page is intentionally left as blank.