1 The Expensive Matrix Multiplication Problem

Finding the optimal way of multiplying matrices, a fundamental operation in many algorithms, has been an important area of study in theoretical computer science. As of 2023, the best peer-reviewed bound on the computational complexity of matrix multiplication was shown to be $\mathcal{O}(n^{2.3728596})$. That is, given two $n \times n$ matrices over some field, the number of elementary operations required to compute their product is upper bounded by $Mn^{2.3728596}$ for all $n \geq n_0$, where M, n_0 are constants and $M > 0$. This can be interpreted as the "time" required for the algorithm to finish running.

Consider the set of $n \times n$ matrices over the field $\mathbb{F}_2 = \{0,1\}$, where $0+0 = 1+1 = 0$, $1+0 = 0+1 = 1$, $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$, and $1 \cdot 1 = 1$. For any matrix A of this form, computing A^k for some constant positive integer k would have complexity $\mathcal{O}(n^{2.3728596})$ by simply applying the current best algorithm $k-1$ times.

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad A^2 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Though, if we are just interested in a specific entry on the matrix A^k, there is a faster way to do this. More specifically, consider the following problem:

Problem 1 (25 Points)**.** Show that one can find the (i,j)-entry of A^k, where A is an $n \times n$ matrix over $\mathbb{F}_2$, in time $\mathcal{O}(n^2)$. Please describe the algorithm clearly and give a full analysis of its correctness and time complexity.

Remark. We assume elementary operations (basic arithmetic operations, comparison between two numbers, writing to and reading from a static array, etc.) to take constant time (i.e. have complexity $\mathcal{O}(1)$)

2 Online Secretary Problem

2.1 One Secretary Needed

Suppose you wish to hire a secretary among n potential candidates. You consider your hiring process a success only if you hire the (unique) strongest candidate in the pool. However, you may only interview the candidates one by one, in random order, with no knowledge of their strength until after the interview, and you may hire the t-th candidate if and only if you make them an offer after their interview and before the next candidate's interview.

With these rules in place, you settle on the following strategy:

- For a certain value $T(n)$, you interview the first $T(n)$ candidates but do not make any of them an offer.
- You offer the job to the first candidate (if any) who is stronger than all of the first $T(n)$ candidates.

Problem 2.1 (10 Points)**.** Find a choice of $T(n)$ that ensures a high probability of success of at least $1/e$.

2.2 An Additional Opening

Suppose now there are two secretary positions open for hire. Again, you are faced with n candidates, and you consider your hiring process a success only if one of the two secretary positions is given to the (unique) strongest candidate. The restrictions on the hiring process remains the same: the candidates are interviewed in a random order, and you can only hire a candidate immediately after their interview.

Problem 2.2 (20 Points)**.** Design an algorithm that helps to achieve your goal. Please describe the algorithm clearly and give a full analysis of the corresponding success probability. Show your reasoning.

2.3 Team Hire

With an expanding company, you now need to hire k secretaries in total from n candidates, following the same restriction. To form a strong team, you want to make the total score of the secretaries hired as large as possible, assuming that the strength of each secretary can be evaluated as a unique nonnegative score. During a discussion with your colleagues, someone provided the following recursive algorithm:

- `select_candidates(n, k)`
  - If $k = 1$, follow the optimal strategy of problem 2.1 to hire the candidate
  - If $k > 1$, generate a random sample m from the binomial distribution $B(n, 1/2)$, where $Pr(m = l) = \binom{n}{l}(\frac{1}{2})^n$. Hire up to $\lfloor k/2 \rfloor$ candidates by applying select_candidates(m, $k/2$). Starting from the $(m+1)^{th}$ candidate, select every candidate that has a score larger than the $\lfloor k/2 \rfloor^{th}$ largest score among the first m candidates. Stop when a total of k candidates are hired or all n candidates have been interviewed.

Problem 2.3 (20 Points)**.** Verify that the total score of the candidates hired using this algorithm has an expected value of at least $(1 - 5/\sqrt{k})$ times the actual optimal.

Hint. Use induction on k. Define the "modified score" of a candidate to be 0 if they are not one of the top k and remain the same otherwise. Prove that the expected modified score is at least $(1 - 5/\sqrt{k})$ times the optimal.

3 Minimizing Error in Randomized Algorithms

Suppose you have an algorithm `A` that takes an input vector $\mathbf{x} \in \{0,1\}^n$. The notation $\{0,1\}^n$ implies that each of the n entries of $\mathbf{x}$ are either 0 or 1. The algorithm `A` will output an answer $y \in \{0,1\}$, with the independent probability of outputting the correct answer as $p \in (1/2 + \varepsilon, 1)$ for some $\varepsilon \in (0, 1/2)$.

Problem 3 (25 Points)**.** Use algorithm `A` to construct an algorithm `A'` in polynomial time complexity that outputs the correct answer with probability at least $1 - 2^{-n}$ for sufficiently large n. Please describe the algorithm clearly, and give a full analysis of the correctness of the algorithm and its time complexity.