Algorithms: Assignment sheet 3

Due date: November 26, 2025

---

1. Euclid's gcd algorithm takes two numbers a and b such that $a > b > 0$, and determines their gcd by computing the following sequence starting with $r_0 = a$ and $r_1 = b$.

$$\begin{array}{lll} r_2 = r_0 \bmod r_1 & q_2 = r_0 \operatorname{div} r_1 & (0 < r_2 < r_1) \\ r_3 = r_1 \bmod r_2 & q_3 = r_1 \operatorname{div} r_2 & (0 < r_3 < r_2) \\ & \cdots & \\ r_k = r_{k-2} \bmod r_{k-1} & q_k = r_{k-2} \operatorname{div} r_{k-1} & (0 < r_k < r_{k-1}) \end{array}$$

   The sequence r_i is strictly decreasing, implying that the algorithm terminates in a finite number of stages. Termination occurs when $r_{k-1} \bmod r_k = 0$ (that is, r_k divides r_{k-1}).

   (i) Show that $r_k = \gcd(a, b)$.

   (ii) Show that this is a polynomial time algorithm.

   (iii) Let F_n be the nth Fibonacci number. Show that the worst case for Euclid's algorithm is when a and b are consecutive Fibonacci numbers. If $a = F_{n+1}$ and $b = F_n$, then the number of stages k equals n. Noting that $F_n \approx \phi^n/\sqrt{5}$, where ϕ is the golden ratio 1.618..., prove that the running time of this algorithm is polynomial in the lengths of a and b.

   (iv) Show that for all $a > b > 0$, there exist integers x and y such that

$$\gcd(a, b) = ax + by.$$

   Moreover, x and y can be computed in polynomial time.

2. We choose a number $a \in \{1, \ldots, n-1\}$ uniformly at random in Miller-Rabin algorithm. How do we perform this step in $O(\mathsf{poly}(\log n))$ time?

3. Let n be a *non-Carmichael* odd composite number. Consider the Miller-Rabin algorithm on input n. Show that with probability $\geq 3/4$, (one invocation of) this algorithm returns the answer "composite".

4. Design a randomized polynomial time algorithm for determining a non-trivial factor of n, given a composite number n and $\phi(n)$, where $\phi(n) = |\mathbb{Z}_n^*|$. That is, the running time of your algorithm should be $\mathsf{poly}(\log n)$ and with probability $\geq 3/4$, it should return a factor of n.

   Hint: Pick an $a \in \{1, \ldots, n-1\}$ uniformly at random. The non-trivial case is when $a \in \mathbb{Z}_n^*$. Use the idea for Carmichael numbers in Miller-Rabin algorithm.

5. We are given an $n \times n$ 0-1 matrix M whose determinant value is known to be between -2^n and 2^n. There is a simple algorithm for computing the determinant of an $n \times n$ matrix using $O(n^3)$ arithmetic operations and it works over any field.

   Arithmetic on $O(\log n)$ bit numbers takes unit time – so when all arithmetic operations in the above algorithm are on $O(\log n)$ bit numbers, then the running time of this algorithm is $O(n^3)$. However when numbers are large, then arithmetic operations cannot be assumed to take unit time; for example, doing arithmetic on n-bit numbers takes $\Theta(n)$ time.

   Show a randomized algorithm with expected running time $O(n^3 \cdot \mathsf{poly}(\log n))$ and success probability $\geq 1 - 1/n$ for determining whether M is singular or not.

   [Hint: You can take the following fact for granted. For any m, the probability that an integer chosen uniformly at random in $\{1, \ldots, m\}$ is prime $\approx c/\ln m$ for some constant c.]

6. We have seen an efficient Monte Carlo algorithm for testing if a given number is prime. In several applications (for example, the RSA crypto scheme), it is necessary to pick large prime numbers at random. Suggest an efficient Monte Carlo algorithm for generating a random $\Theta(\log n)$ bit length prime.

7. Alice has a copy of a long n-bit file $A = \langle a_{n-1}, a_{n-2}, \ldots, a_0 \rangle$, and Bob similarly has an n-bit file $B = \langle b_{n-1}, b_{n-2}, \ldots, b_0 \rangle$. Alice and Bob wish to know if their files are identical, that is, if $a_i = b_i$ for all $0 \leq i \leq n-1$. Alice can send her file A to Bob and Bob can compare A and B and check if their files are the same or not, and convey the yes/no answer to Alice. However this involves $n+1$ bits of communication, which is expensive. Alice and Bob want to communicate just $\Theta(\log n)$ bits and know the correct answer with high probability.

   So they are ready to use a probabilistic check. Design a probabilistic check such that if $A = B$, then Alice and Bob decide that they are equal, while if $A \neq B$, then Alice and Bob decide that $A \neq B$ with probability at least 0.999.

   Note that computation at Alice's end or at Bob's end comes for free - it is *communication* that is expensive.

   [Hint: Use the fact that there always exists a prime $p > 1000n$, in fact, there is a always prime sandwiched between $1000n$ and $2000n$. Also use the fact that a non-zero polynomial of degree n has at most n distinct roots.]

8. Given an undirected graph G, an edge colouring is an assignment of colours to the edges of G such that no two edges incident to the same vertex get the same colour. The edge colouring problem asks for an edge colouring using the minimum number of colours. This is a hard problem to solve. Here we are interested in the *online edge colouring* problem where the vertex set V of the graph G is fixed and the edges in the graph are presented to us in an online manner, one after another.

   As each edge e is specified, our algorithm must assign this edge e a colour and this colour of e cannot be changed henceforth. While the offline edge colouring of G knows all the edges of E while deciding on their colours, the online edge colouring algorithm has to decide on the colour of each edge e without any knowledge of the future edges. Design a 2-competitive algorithm for the online edge colouring problem.

9. Recall the online paging problem: the cache size is k. Suppose the total number of distinct items in memory is $k+1$. Show that the competitive ratio of the Marker algorithm is H_k.

10. We are given a set $S = \{t_1, \ldots, t_n\}$ of integers where $n^5 \leq t_i \leq n^{10}$ for all $i \in \{1, \ldots, n\}$. We also have a target value T. Give a Las Vegas algorithm that runs in expected $O(n)$ time to decide if there exists a pair $t_i, t_j \in S$ such that $t_i + t_j = T$.