Advanced Complexity Exam (2020-21)

You may answer in English or in French. If you answer in French, and you do not know the French equivalent of some English word that I am using, by all means do not invent your own translation: use the English word instead in that case.

Let me stress the value of rigor. In all arguments, you must: (a) stress the important idea; (b) give explicit values for all required bounds (errors, probabilities, time and space usage, and so on); (c) give explicit references to the results you use, preferably by name (e.g., "the Immerman-Szelepcsényi theorem").

In the whole question, # is a a separator (a letter) different from all other letters. Section 1 and Section 2 are largely independent, but not completely.

1 Lipton fingerprints and graph coloring

The only interesting question in this section is **Question 4**, and will account for more that half of the points of the whole question. (My solution to it takes almost two pages, which is far more than any of the other questions.) The preceding questions are simply warm-ups, the questions after that are easy variants, and will not account for much either.

Two lists $[x_1, x_2, \ldots, x_k]$ and $[y_1, y_2, \ldots, y_{k'}]$ are equal *up to permutation* if and only if $k = k'$ and there is a permutation π of $\{1, 2, \cdots, k\}$ such that $y_{\pi(1)} = x_1$, $y_{\pi(2)} = x_2$, $\ldots$, $y_{\pi(k)} = x_k$.

Given a prime number p, we will admit without proof that two lists $\ell \stackrel{\text{def}}{=} [x_1, x_2, \ldots, x_k]$ and $\ell' \stackrel{\text{def}}{=} [y_1, y_2, \ldots, y_{k'}]$ of natural numbers in $[0, p-1]$, with $k, k' < p$, are equal up to permutation if and only the *Lipton polynomial*:

$$P(\ell, \ell') \stackrel{\text{def}}{=} \prod_{i=1}^{k} (X - x_i) - \prod_{j=1}^{k'} (X - y_j), \tag{1}$$

which is a polynomial in $\mathbb{Z}/p\mathbb{Z}[X]$, is the zero polynomial. Indeed, the Lipton polynomial is zero if and only if the two polynomials $\prod_{i=1}^{k}(X - x_i)$ and $\prod_{j=1}^{k'}(X - y_j)$ have the same roots, counting multiplicities. We require $k, k' < p$ in order

to ensure that one can reason equivalently on polynomials or on polynomial functions, without running the risk of mistakes.

The *Lipton fingerprint* (in French: empreinte de Lipton) $Lip(\ell, r)$ of a list $\ell \stackrel{\text{def}}{=} [x_1, x_2, \ldots, x_k]$ as above, taken at r ($0 \leq r \leq p-1$), is the value of the polynomial $\prod_{i=1}^{k}(X - x_i)$ at $X \stackrel{\text{def}}{=} r \bmod p$; in other words, $\prod_{i=1}^{k}(r - x_i) \bmod p$.

We define the following *Lipton protocol*. Its purpose is to decide whether two lists $\ell \stackrel{\text{def}}{=} [x_1, x_2, \ldots, x_k]$ and $\ell' \stackrel{\text{def}}{=} [y_1, y_2, \ldots, y_{k'}]$ of natural numbers in $[0, p-1]$ (p prime, $k, k' < p$) are equal up to permutation, with high probability. The Lipton protocol draws a number r uniformly at random between 0 and $p-1$, and computes the fingerprints $Lip(\ell, r)$ and $Lip(\ell', r)$. We accept if the two fingerprints match (are equal), we reject otherwise.

Question 1 Show that the Lipton protocol does not make a mistake if ℓ and ℓ' are equal up to permutation; in other words, it accepts in this case.

> *The Lipton polynomial $P(\ell, \ell')$ is the zero polynomial. Its value at any number r is therefore equal to 0.*

Question 2 What is the probability that the Lipton protocol makes a mistake when ℓ and ℓ' are not equal up to permutation? I am expecting an easy upper bound, bounded by a polynomial in k and k' (when p is fixed) and tending to 0 as p tends to $+\infty$.

> *In that case, $P(\ell, \ell')$ is not the zero polynomial. Its degree is at most $\max(k, k')$. By Lagrange interpolation (the dimension 1 case of the Schwartz-Zippel lemma), it has at most $\max(k, k')$ roots, among p possible values (in $\mathbb{Z}/p\mathbb{Z}$). The probability of error is at most $\max(k, k')/p$.*

Question 3 Why does the Lipton protocol (a) only require space $O(\log p)$? (b) $O(\log p)$ random bits on average? (c) How much time does it take on average? You must state explicitly how you pick r at random, the space complexity of arithmetic operations, and the algorithm you use in order to evaluate the various Lipton fingerprints.

> *We draw r by rejection sampling: draw r uniformly at random among all $|p|$-bit numbers, until $r < p$. ($|p| = \Theta(\log p)$ is the bit size of p.)*
>
> *The average number of turns through a rejection sampling loop was seen during the lectures when we proved $\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP}$, and is the inverse of the probability of success, namely $2^{|p|}/p \leq 2$. We multiply this by the time needed to draw $|p|$ bits at random, namely $O(\log p)$. Hence the average time is $2O(\log p) = O(\log p)$.*
>
> *The number of random bits used is also $O(\log p)$, by the same argument.*

The space used so far is that used by r, that is, $|p|$. Then we must compute the two Lipton fingerprints. Each arithmetic operation requires space $O(\log p)$, because we use it on numbers $< p$, hence of $|p|$ bits. Time is $O(\log p)$ for additions (or subtractions) and $O(\log^2 p)$ for multiplications (although we can make the exponent smaller), even mod p, as we have seen in the lectures.

We compute each of the two Lipton fingerprints $\prod_{i=1}^{k}(r - x_i)$ mod p: this requires k subtractions and k multiplications, so the total if $O(k(\log p + \log^2 p)) = O(p \log^2 p)$, as we have seen in the lectures.

The space used is only $O(\log p)$, since we only need to keep on $|p|$-bit register to compute $r - x_i$ and another one to multiply it to a register, initialized to 1.

A *graph* $G \stackrel{\text{def}}{=} (V, E)$ is a pair of a finite set $V \stackrel{\text{def}}{=} \{1, 2, \cdots, N\}$ of so-called *vertices* (singular: vertex; in French: sommet) and a subset $E \subseteq V \times V$ or *edges* (French: arcs). Note that N is the number of vertices; all of them are represented as numbers in binary.

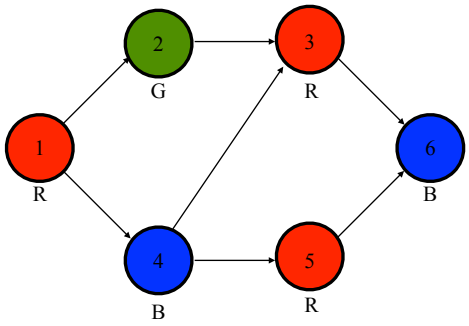
We represent G by its adjacency matrix. That is an $N \times N$ matrix of bits M, which we write as a sequence of N^2 bits. We will represent G as the word $0^N 1 M$, namely N zeros followed by a 1 (in order to specify N), followed by the N^2 entries of M. For any two vertices u and v (with $1 \leq u, v \leq N$), there is an edge $u \to v$ in G if and only if the bit at position $Nu + v$ in the word representation of the graph is equal to 1. Positions start at 0.

Let $Col \stackrel{\text{def}}{=} \{R, G, B\}$ be the set of so-called *colors*. A *3-coloring* of G is a map $col : V \to Col$ such that for every edge $u \to v$ in G, $col(u) \neq col(v)$. We will say that $col(u)$ is the *color* of u (with respect to col).

The *3-coloring problem* 3-COL is:
INPUT: a graph G;
QUESTION: does G have a 3-coloring?

This is an **NP**-complete problem under logspace reductions.

We will use a variant of the Lipton protocol in order to decide 3-COL between Arthur and Merlin, using as little resources (space, number of random bits) as we can. Let me use an example in order to explain the idea. Let us imagine that Arthur and Merlin wish to decide whether the following graph has a 3-coloring.



Matrice d'adjacence:

0 1 0 1 0 0
0 0 1 0 0 0
0 0 0 0 0 1
0 0 1 0 1 0
0 0 0 0 0 1
0 0 0 0 0 0

Of course, the answer is yes in this case, and I have taken the opportunity to show one possible 3-coloring.

We encode a 3-coloring candidate function *col* as a list of words *pos*#*b*#*c* where:

- *pos* is a natural number in binary such that $N + 1 \leq pos \leq N^2 + N$; in other words, *pos* is a position in the M part of the input word representing G;
- $b \in \{0, 1\}$; 0 means "source" and 1 means "target";
- $c \in Col$.

Such a word is meant to say that *pos* is the position of an edge $u \to v$ (namely, that $pos = Nu + v$ and that $u \to v$ really is an edge of G), and that c is the color, either of its source u (if $b = 0$) or of its target v (if $b = 1$). In the example such a list might be:

$$\begin{array}{lll} [\ \ 8\#0\#R, 8\#1\#G, & & (\text{edge } (1,2)) \\ \ \ 10\#0\#R, 10\#1\#B, & & (\text{edge } (1,4)) \\ \ \ 15\#0\#G, 15\#1\#R, & & (\text{edge } (2,3)) \\ \ \ 24\#0\#R, 24\#1\#B, & & (\text{edge } (3,6)) \\ \ \ 27\#0\#B, 27\#1\#R, & & (\text{edge } (4,3)) \\ \ \ 29\#0\#B, 29\#1\#R, & & (\text{edge } (4,5)) \\ \ \ 36\#0\#R, 36\#1\#B &] & (\text{edge } (5,6)) \end{array} \tag{2}$$

(The comments between parentheses on the right are just comments, and are not part of the words, or of the whole list of words. Numbers are shown in decimal notation for readability.)

The protocol is in two phases. In each one, Merlin sends a list, which he claims are equal up to permutation. Each of these two lists is meant to be sorted, but according to different criteria.

In *phase 1*, Merlins sends a list ℓ_1 as above to Arthur, and claims that ℓ_1 is sorted by increasing vertex numbers. The *vertex number* of *pos*#*b*#*c*, where $pos = Nu + v$, is u if $b = 0$, v if $b = 1$. Merlin also claims that the colors are

consistent, namely that any two words with the same vertex number are given the same color. In the example, an honest Merlin would play:

$$\begin{array}{ll} [\ \ 8\#0\#R, 10\#0\#R, & \text{(vertex 1)} \\ \ \ 8\#1\#G, 15\#0\#G, & \text{(vertex 2)} \\ \ \ 15\#1\#R, 27\#1\#R, 24\#0\#R, & \text{(vertex 3)} \\ \ \ 10\#1\#B, 27\#0\#B, 29\#0\#B, & \text{(vertex 4)} \\ \ \ 29\#1\#R, 36\#0\#R, & \text{(vertex 5)} \\ \ \ 24\#1\#B, 36\#1\#B\ \ ] & \text{(vertex 6)} \end{array}$$

In *phase 2*, Merlin sends a similar list ℓ_2. He claims that ℓ_1 and ℓ_2 are equal up to permutation, but that ℓ_2 is sorted by *increasing positions* k on the input tape, that all the edges are listed exactly twice, once for each end of each edge, and that the colors of those two ends are distinct. In the example, an honest Merlin would play the list (2) mentioned earlier.

Question 4 Using this idea, show that 3-COL is in the class $\mathbf{IP}_1$ of interactive proofs such that:

(a) the playing order is **MAM** (Merlin plays, then Arthur, then Merlin; then Arthur decides);

(b) Merlin's answers have polynomial size, and Arthur can only read them from left to right when his turn comes, and cannot go back;

(c) Arthur does not ask *any* question when his turn comes (and therefore does not reveal any random bit, although it will have to draw some bits at random);

(d) Arthur works in logarithmic space (not counting the space used by the communication tape between Arthur and Merlin, or by the input tape) and polynomial time;

(e) Arthur only draws a logarithmic amount of random bits.

We agree that the final verification phase is done by Arthur, so all resource constraints that apply to Arthur are also relevant for the final verification phase. You will use the notations G, V, E, n, N, M, etc., given above. Let me remind you that "logarithmic" means $O(\log n)$, where n is the size of the input. Also, as usual, the communication tape between Arthur and Merlin is reused, hence erased at each turn.

The acceptance conditions are:

(f) If G has a 3-coloring, then Merlin has a way of making the protocol accept.

(g) Otherwise, whatever Merlin's strategy is, the protocol will accept with probability at most 1/4.

You will pay special care to the details. For instance, time must be polynomial in the *worst* case, not on average. Also, you can test whether a given number p is prime in various ways, see Appendix A, but not all of them may fit our purposes here. Finally, say explicitly how all of Merlin claims are to be verified, and explain why all the various constraints on resources are met, how many registers you need, holding numbers in binary or in unary, and so on.

As far as (d) is concerned, polynomial time is automatic: if Arthur only uses log space and terminates, then it terminates in polynomial time.

The basic idea is to let Merlin play according to phase 1 as described earlier. Then Arthur verifies Merlin's claims in log space, draws r mod p uniformly at random, and computes $Lip(\ell_1, r)$. Then Merlin plays according to phase 2. In order to decide, we verify Merlin's claims for that new phase, again in log space, and we compute $Lip(\ell_2, r)$. If $Lip(\ell_1, r) = Lip(\ell_2, r)$, then we accept, otherwise we reject.

If G has a 3-coloring, then $Lip(\ell_1, r) = Lip(\ell_2, r)$ if Merlin plays honestly, so (f) holds. Otherwise, the probability that we make a mistake, namely that $Lip(\ell_1, r) = Lip(\ell_2, r)$, is at most k/p, where k is the length of the lists ℓ_1 and ℓ_2. Note that k is at most $2N^2$, hence at most $2n$.

There are plenty of technical details to solve.

- *It does not make sense to compute $Lip(\ell_1, r)$ and $Lip(\ell_2, r)$ per se, because ℓ_1 and ℓ_2 are not lists of natural numbers in $[0, p-1]$. But we can encode each word pos#b#c as a string of $\log n + O(1)$ bits, which we can interpret as a natural number, in binary, of size $\log n + O(1)$.*
- *Next, we must find p so large that the error bound k/p be at most $1/4$. For technical reasons, I will require that it is at most $1/5$ (see below). We have $k = O(n)$, so we want $p \geq 5k$, namely we require a prime number of at least $\log n + O(1)$ bits.*

  We also require p to be larger than the length of ℓ_1, the length of ℓ_2, and all the codes of elements of those lists. As far as the lengths of lists are concerned, they are at most $2N^2$, namely at most $2n$, so we require $p > 2n$. It suffices to require p to has at least $\lceil \log_2 n \rceil + 1$ bits, and that is again of the form $\log n + O(1)$.

  Let us require p to be a number of size $\max(\lceil \log_2(5k) \rceil, \lceil \log_2 n \rceil) = \log n + O(1)$ bits exactly. This always exists by Bertrand's postulate.
- *We must obtain p in log space. The simplest way is to require Merlin to give us p in the first turn, in addition to ℓ_1. One can also draw p at random and use rejection sampling.*

- *We must check that p has the right size, and that it is prime. It may seem that we could use any of the primality tests of Appendix A, except number 2 (we do not have universal non-determinism). However, we need to use the Miller-Rabin test. This is the only one that runs in sublinear space (in fact, log space) in the size of p, that is, in space $O(\log n)$. The Agrawal-Kayal-Saxena test uses polynomial, not linear, space: if the degree of the polynomial is a, that would mean $O(\log^a n)$ space, which is not acceptable.*

- *Drawing r at random (as well as p, if drawn at random) can be done in average time $O(\log n)$, and a constant number of turns through the rejection sampling loop (2). But we want a procedure that terminates* always. *In order to achieve this, we can bound the number of turns through the loop by some number K to be found later. If Arthur has not found $r < p$ after K turns of the loop, it stops the whole game and accepts. This incurs an additional error, whose probability is at most $2/K$, by Markov's inequality.*

  We require $K \geq 40$, so that $2/K \leq 1/20$. When added to the error $1/5$ (see item 2), this makes at most $1/4$, which is the desired final probability of error.

- *Arthur must verify the format of Merlin's answers. Notably, it must check that the length of the lists provided by Merlin do not exceed $2N^2$. This requires scanning the list in time $2N^2 = O(n)$, using a binary counter of size $O(\log n)$.*

  Arthur must also verify the format of each entry pos#b#c, and notably that the size of each does not exceed p. It is enough to verify that pos $\leq N^2$ (or simply pos $\leq n$), that b holds just one bit, and that $c \in \{R, G, B\}$.

- *Arthur checks Merlin's claims in phase 1 in log space. The point is that the consistency condition and the sorting condition only require one to remember the last word pos#b#c encountered while scanning the list ℓ_1.*

  In order to avoid some recomputation, we can also remember the vertex number i of pos#b#c. When we read the next one, pos′#b′#c′:

  - *We compute the vertex number i' of pos′#b′#c′. This is done by a Euclidean division of pos′ by N, returning the remainder (if $b' = 1$) or the quotient (if $b' = 0$). We have got time, but not much space. Since N is given in unary on the input tape, the simplest solution is to copy pos′ into a register R', and to repeatedly substract N from R', incrementing u' (in binary, initialized to 0), until R' is no longer ≥ 1. At the end of this*

process, R' will hold the end vertex v' of the supposed edge $u' \to v'$ at position pos'. We compute i', the required vertex number, as being u' if $b' = 1$, and $R' = v'$ otherwise.
In order to subtract a unary number (N) from a binary number (pos'), it is enough to repeatedly subtract one.
That only requires a constant number of counters. Note that we must represent i, i', u' and R' in binary in order to only use log space.

- *If $i = i'$, then we check that $c = c'$ (consistency).*
- *Otherwise, we verify that $i < i'$ (sorting condition).*

The sorting condition allows us to guarantee that we only need to check the consistency of adjacent *words $pos\#b\#c$.*

- *Meanwhile, Arthur computes $Lip(\ell_1, r)$ as in **Question 3**. That requires one to store the code of the current word $pos\#b\#c$, plus logarithmically many bits for the intermediate registers. Explicitly, we keep a register R, initialized to 1. Each time Arthur reads a new word $pos\#b\#c$ from ℓ_1, it computes its code, subtracts it from r, and multiplies the result by R mod p, storing the final result in R again.*
- *In order to verify Merlin's claims in phase 2 (except for the equality of the Lipton fingerprints, which we will deal with later), we simply read the adjacency matrix M from left to right, and we verify that ℓ_2 enumerates the following in the same order, namely for each position pos in M:*
  - *either exactly two words $pos\#0\#c_0$ and $pos\#1\#c_1$ (in this order, even; that simplifies the checking task), with the right value of pos in both cases, with $c_0 \neq c_1$; it must also check that the bit at position pos on the input tape is 1, namely that the edge indeed exists!*
  - *or nothing, if the bit at position pos on the input tape is 0.*

  We reject as soon as any check fails, namely if the value of pos is not the expected one in the first case, for example; but also if at the end of the scan of M, there remains words not yet seen in ℓ_2 (although this is not really required).
- *Meanwhile, Arthur computes $Lip(\ell_2, r)$, as in **Question 3**, or more precisely as in the computation of $Lip(\ell_1, r)$, and that still only requires logarithmically many bits.*
- *Finally, deciding whether the two stored Lipton fingerprints are equal is trivial.*

Question 5 Modify the previous interactive proof slightly, and show that 3-COL is also in the class $\mathbf{IP}_2$ of interactive proofs where:

(a) the playing order is: Arthur plays once, then Merlin plays polynomially many times;

(b) each of Merlin's answers is exactly 1 bit long,

and conditions (c)–(g) are unchanged.

> *First, Arthur has to produce p himself, and can no longer count on Merlin for this task. (If you insisted on counting on Merlin, there was a more complicated solution: draw three, or four candidates depending on the actual probability bounds you aim four, for the values of r, at random uniformly among bit strings of the same length as the awaited number p; store them all, then once Merlin has given his number p, check that its size is what we expected, and pick for r the first of the candidate values that is strictly less than p—or accept if none of them is.)*
>
> *Instead of drawing r at random after Merlin produces ℓ_1, Arthur draws it at random before. That does not change anything, since Merlin must play without knowing anything about r anyway.*
>
> *Then, Merlin will provide the lists ℓ_1 and ℓ_2 one bit at a time. This is possible since the Lipton fingerprints and the various checks are done incrementally, and only require one to store each element $pos\#b\#c$ as they are received.*

Question 6 Deduce that $\mathbf{NP} \subseteq \mathbf{IP}_1$ and that $\mathbf{NP} \subseteq \mathbf{IP}_2$.

> *3-COL is* $\mathbf{NP}$*-complete under log space reductions, as we have said early in the question. Polynomial time reductions are not enough! Indeed, there is no (simple) way to guarantee that* $\mathbf{IP}_1$ *or* $\mathbf{IP}_2$ *is closed under polynomial time reductions.*
>
> *We then check that* $\mathbf{IP}_1$ *and* $\mathbf{IP}_2$ *are closed under log space reductions. We use the usual trick of composing log space functions: instead of storing and reading the intermediate tape, we simulate the whole computation of the first function, keeping only the letter of its output tape that the second function wishes to read.*

2 Strategies

A *strategy* is just what we called a Merlin map in the lectures. It maps (public) histories $x\#q_1\#y_1\#q_2\#y_2\#\cdots\#q_i$ to an answer y_i by Merlin. We require all

questions q_i and all answers y_i to be of size bounded by a fixed polynomial in the size n of x.

IP(logspace, lograndbits) is the class of languages decided by interactive proofs where Arthur uses only $O(\log n)$ space and $O(\log n)$ random bits. The acceptance conditions are: if $x \in L$, then the interactive proof will accept with probability at least 3/4 (if Merlin plays honestly); if $x \notin L$, then it will accept with probabliity at most 1/4, whichever strategy Merlin uses.

Erratum. IP(logspace, lograndbits) is the class of languages decided by interactive proofs where Arthur uses only $O(\log n)$ space and $O(\log n)$ random bits, *and* working in a polynomial number of rounds.

Question 7 We are given an **IP** protocol between Arthur and Merlin, with polynomially many rounds, say $p(n)$. Show that, if Arthur uses only $O(\log n)$ random bits in that **IP** protocol, then there is a concise way of representing any of Merlin's strategies M on input x. By *concise*, we mean of polynomial size. By a representation, we mean a data structure $S(x)$, depending on the input x, and a polynomial-time computable function f such that $f(S(x), x\#q_1\#y_1\#q_2\#y_2\#\cdots\#q_i) = M(x\#q_1\#y_1\#q_2\#y_2\#\cdots\#q_i)$ for all possible public histories $x\#q_1\#y_1\#q_2\#y_2\#\cdots\#q_i$ that may result from the interaction between Arthur and Merlin on input x.

> *On input x, any two plays played with the same strategy from Merlin and with the same sequences of random bits must be equal. Assume Arthur only uses $k \log n$ random bits. This makes at most $2^{k \log n} = n^k$ possible sequences of random bits, hence at most n^k different possible histories. Merlin only has to store up to $p(n)$ answers to the $p(n)$ questions asked in the run obtained by fixing the random bits.*
>
> *Hence $S(x)$ is a table, or rather an association list, which maps any of the at most $p(n)n^k$ possible public histories $x\#q_1\#y_1\#q_2\#y_2\#\cdots\#q_i$ that may occur during the interaction between Arthur and Merlin to the answer y_i.*
>
> *The size is $S(x)$ is at most $p(n)n^k$ times the length of the largest history, which is also a polynomial.*
>
> *The function f simply looks up the public history in the association list $S(x)$, in polynomial time.*

Question 8 Show that any language L in **IP**(logspace, lograndbits) has an **MA** protocol (an Arthur-Merlin protocol where Merlin plays first, Arthur plays second and there is no other round), with resource constraints as in the lectures, except that Arthur uses only logarithmically many random bits; and where the error probability is at most 1/4.

Given an $\mathbf{IP}(\textit{logspace}, \textit{lograndbits})$ *protocol* π *for* L*, we build the claimed* **MA** *protocol by letting Merlin output a polynomial-size strategy first, or rather the table* $S(x)$ *found in* ***Question 7****. Then Arthur will play* π *alone, simulating Merlin's moves in* π *by consulting* $S(x)$ *instead.*

If $x \in L$*, then Merlin had a way of winning* π *with probability at least* $3/4$*. Playing the corresponding table* $S(x)$*, Arthur will necessarily accept with probability at least* $3/4$ *as well.*

If $x \notin L$*, then whatever the polynomial-sized table* S *that Merlin plays in the* **MA** *protocol, playing* π *by letting Merlin use the associated strategy (obtained by looking* S *up) gets him no better chance of winning than by any strategy, i.e., Merlin wins with probability at most* $1/4$*. (If* S *is not even correctly formatted as a table, Arthur simply rejects.)*

Notice finally that it does not matter whether Arthur's coins are public (as required here) or private (as given in the original protocol π*), because Merlin will never play after Arthur.*

Question 9 Show that $\mathbf{IP}(\text{logspace}, \text{lograndbits}) \subseteq \mathbf{NP}$.

We derandomize Arthur's computations in the protocol of Question ***Question 8****. Since Arthur has only* $k \log n$ *random bits, we can instead simulate all* $2^{k \log n} = n^k$ *possible runs of Arthur, by enumerating all* $k \log n$*-bit random bit strings, and counting how many lead Arthur to accept. Either at least* $3/4$ *of them will lead to acceptance, and we accept; or at most* $1/4$ *lead to acceptance, and we reject.*

This computation only takes polynomial time by Arthur. So we obtain an **MA** *protocol where Merlin plays first, and Arthur decides in polynomial time. This is an* $\mathbf{M} = \mathbf{NP}$ *protocol.*

Question 10 Conclude that all the classes $\mathbf{IP}(\text{logspace}, \text{lograndbits})$, $\mathbf{NP}$, $\mathbf{IP}_1$ and $\mathbf{IP}_2$ are equal.

We have just proved $\mathbf{IP}(\textit{logspace}, \textit{lograndbits}) \subseteq \mathbf{NP}$*. By* ***Question 6****,* $\mathbf{NP}$ *is included in* $\mathbf{IP}_1$ *and in* $\mathbf{IP}_2$*.* $\mathbf{IP}_2$ *is clearly included in* $\mathbf{IP}(\textit{logspace}, \textit{lograndbits})$*, so* $\mathbf{IP}(\textit{logspace}, \textit{lograndbits}) = \mathbf{IP}_2 = \mathbf{NP} \subseteq \mathbf{IP}_1$*.*

It remains to show that $\mathbf{IP}_1 \subseteq \mathbf{IP}(\textit{logspace}, \textit{lograndbits})$*, but that is obvious.*

A Primality tests

Let p be a positive natural number. Its size is $|p| = \lceil \log_2 p \rceil + 1$.

1. One can test whether p is prime in deterministic polynomial time by an algorithm due to Kayal, Agrawal, and Saxena. The space used is polynomial.

2. Primality is in **coNP**. One can check that p is prime by verifying that for every x with $2 \leq x \leq p-1$, x does not divide p.

3. Primality is in **NP**, by a result of Pratt, using a recursive version of Lucas' primality test. The idea is that a *proof of primality* of p is a tree whose root is labeled by p and by a natural number x in $[1, p-1]$, and whose immediate subtrees are proofs of primality of numbers $p_1, \ldots, p_k$ in $[1, p-2]$. In order to verify such a proof, one checks that: either $p = 2$ and $k = 0$; or $p \geq 3$ is odd, x is between 1 and $p-1$, the immediate subtrees are valid proofs of primality of $p_1, \ldots, p_k$, $\prod_{i=1}^{k} p_i = p - 1$, x is prime with p, $x^{p-1} = 1 \bmod p$, and $x^{(p-1)/p_i} \neq 1 \bmod p$ for every i with $1 \leq i \leq k$.

4. The Rabin-Miller test. We assume p odd: the even case is trivial. Then one can write p as $1 + 2^k q$ where q is odd. We draw x uniformly at random between 1 and $p-1$, we verify that $x^q = 1 \bmod p$ or that $x^{2^j q} = -1 \bmod p$ fr some j with $1 \leq j < k$. If p is prime, the test always succeeds. Otherwise, it fails with probabilty at least $3/4$.

5. The Solovay-Strassen test. We again assume p odd. We draw x at random as above, and we check that $x^{(p-1)/2} = \left(\frac{x}{p}\right) \bmod p$, where the Jacobi symbol $\left(\frac{x}{p}\right)$ is computed recursively by the following formulae (where n is odd):

   - $\left(\frac{m}{n}\right) = \left(\frac{m \bmod n}{n}\right)$, if $m \geq n$;
   - $\left(\frac{1}{n}\right) = 1$;
   - $\left(\frac{2m}{n}\right) = \left(\frac{m}{n}\right)$ if n is equal to 1 or 7 mod 8, $\left(\frac{2m}{n}\right) = -\left(\frac{m}{n}\right)$ if n is equal to 3 or 5 mod 8;
   - when m is odd and $m < n$, $\left(\frac{m}{n}\right) = \left(\frac{n}{m}\right)$ if n or m is equal to 1 mod 4, $\left(\frac{m}{n}\right) = -\left(\frac{n}{m}\right)$ if $n = m = 3 \bmod 4$.

   If p is prime, the test always succeeds. Otherwise, it fails with probabilty at least $1/2$.