Problem 1: Non-binary coins

Our definition of PTM gives the machine the possibility to pick random bits (i.e., do something with probability $1/2$). But what if we want to do something with a different probability, say, $1/3$? The definition of PTM should not depend on what kind of random numbers we use!

(a) Show that there is a polynomial-time PTM that, given input $x$, outputs $1$ with probability $1/3 \pm 2^{-|x|}$ (but not more or less).

Note: This gives us a subroutine that we can use to simulate $1/3$-random bits with high enough precision not to influence the overall output distribution of the PTM too much.

**Solution.** The PTM does the following:

- Let $n := |x|$.
- Pick an integer $r \in \{0, \ldots, 2^n - 1\}$. (This can be done by taking $n$ random bits and interpreting them as an integer.)
- Return $1$ iff $3 \mid r$.

The algorithm works in polynomial time. The probability of output $1$ is $2^{-n}R$ where $R$ is the number of integers $r$ in $\{0, \ldots, 2^n - 1\}$ with $3 \mid r$. We have $R = \lceil 2^n/3 \rceil = 2^n/3 \pm 1$. Thus $2^{-n}R = 2^{-n}(2^n/3 \pm 1) = 1/3 \pm 2^{-n}$.

(b) Show that there is not polynomial-time PTM that, given input $n$, outputs $1$ with probability exactly $1/3$.

**Hint:** You can assume that the PTM runs exactly $T(n)$ steps for some function $T$ (i.e., the runtime does not depend on the random choices). Let $R$ be the number of random choices that lead to output $1$. Show that the probability of output $1$ is a fraction whose denominator is a power of two.

**Solution.** We can assume that the PTM runs exactly $T(n)$ steps because we can just make it run extra steps when needed (that do not affect the final result). There are $2^{T(n)}$ different possible random choices when running exactly $T(n)$ steps. The probability of output $1$ is thus $R/2^{T(n)}$. This is a fraction which has a power of two
in the denominator. $1/3$ cannot be written as such a fraction. Thus $R/2^{T(n)} \neq 1/3$. Hence the PTM does not output 1 with probability exactly 1/3. 

**Problem 2: Amplification**

We have define the class $\text{BPP}$ as the set of all languages that can be solved in probabilistic polynomial time with probability at least 2/3. We have seen that the arbitrary number 2/3 does not matter much: if we can decide $L$ in probabilistic polynomial time with probability 2/3, then we can decide $L$ in probabilistic polynomial time with probability $\alpha$ for any constant $\alpha < 1$.

We showed this by simply constructing a new algorithm $\hat{M}$ that runs the original algorithm $M$ $t$ times (for suitable $t$), and then outputs the most common output.

A search problem is a relation $S$ between bitstrings (i.e., $S$ is a set of pairs $(x, y)$ where $y$ is a solution for $x$). We say we can “solve $S$ in probabilistic polynomial time with probability $\alpha$” iff there is a PTM $M$ such that for all $x$, $\Pr[(x, y) \in R : y \leftarrow M(x)] \geq \alpha$.

Is amplification also possible for search problems? That is, is the following fact true?

**Lemma 1 (Wrong!)** If we can solve a search problem $S$ in probabilistic polynomial time with probability 2/3, then we can solve $S$ in probabilistic polynomial time with probability $\alpha$.

We will consider the prime search problem. Given $N$, the problem is to find a prime $p \in [N, 2N]$. Assume that we have a PTM $M_{\text{prime}}$ that solves the prime search problem with probability 2/3.

(a) Show that amplification by majority decision does not work for constructing a PTM $M_{\text{prime}}$ that solves the prime search problem in probabilistic polynomial time with probability, say, 3/4.

**Solution.** If we run $M_{\text{prime}}(N)$ $t$ times (for polynomial $t$), we will get a different number each time (with high probability), about 2/3 of them being primes. ($[N, 2N]$ contains exponentially many primes.) Since no number is repeated, a majority decision will output an arbitrary one of these numbers. With probability 1/3 it will be a non-prime. Thus we return a prime with probability less than 3/4.

(b) Show how to solve the prime search problem in probabilistic polynomial time with probability 3/4 using $M_{\text{prime}}$.

**Note:** Here you may use the fact that it can be decided in deterministic polynomial time whether a given number is a prime.

---

1For simplicity, we consider only search problems where every $x$ has a solution.
Solution. The algorithm for finding a prime in $[N,2N]$ with probability $3/4$ is:

- $p \leftarrow M_{\text{prime}}(N)$.
- Test if $p$ is prime. If so, return $p$.
- Else: repeat (max. $t$ times).
- After $t$ unsuccessful tries: return $\bot$.

The probability of outputting $\bot$ is at most $(1/3)^t$. If $\bot$ is not output, the output must be a prime. Thus we output a prime with probability $1 - (1/3)^t$. For $t = 2$ we have $1 - (1/3)^2 = 8/9 \geq 3/4$.

(c) (Bonus points) Show that Lemma 1 is not true.

**Hint:** That is, you should show that there is a search problem that can be solved in probabilistic polynomial time with probability $2/3$, but not with probability, say, $3/4$. For example, you could use a random search problem $S$ where for each $x$, a bitstring $y$ is a solution with probability $0.7$.

**Note:** A complete proof will be quite elaborate. A proof sketch is sufficient.

Solution. Sketch: Let $S$ be as in the hint. The following algorithm $M$ solves $S$ with probability $2/3$: On input $x$, output a random $y$. Since $0.7 > 2/3$ of all $y$ are correct solutions for $x$, $M(x)$ solves $S$ with probability $2/3$.

However, $S$ cannot be solved with probability $3/4$. For every $x$, the set $Y = \{y : (x,y) \in S\}$ of solutions is a random set which contains each $y$ with probability $0.7$. Thus, a PTM $M(x)$ will output a $y'$ that is stochastically independent of $Y$. (Note that $M$ is never given any information about $S$ or $Y$.) Hence $y' \in Y$ with probability $0.7 < 3/4$. Hence $M$ does not solve $S$ with probability $3/4$.

Note that this argument was for a randomized search problem. What we skip in this proof sketch is to show that if a randomized search problem is not solvable, then by picking and fixing a search problem $S$ at random, with probability 1 we get a search problem that is not solvable. (There is some additional math involved here.)