Ph/CS 219, Fall 2006
Quantum Computation
Prof. Alexei Kitaev

Problem set #3
Due Friday, November 17, 2006

3.1 The $O(\cdots)$ notation. We have used this notation in class: $f(n) \leq O(g(n))$ if there are some constants c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$. The O notation allows one to compare the asymptotic growth of different functions: $f \leq O(g)$ means that f grows more slowly, or at least not faster than g. Note that two function can grow at the same rate, e.g., $f(n) = 2n$ and $g(n) = n + 100$. In some cases neither $f \leq O(g)$ nor $g \leq O(f)$, e.g., $f(n) = n$ and $g(n) = n^2 \sin^2(\log n)$ (of course, this example is rather contrived).

Sort the following functions in order of their asymptotic growth (i.e., "$\leq$" stands for the O-relation and "$=$" stands for the bi-directional O-relation):

$$n, \quad n \log n, \quad (\log_2 n)^n, \quad (\ln n)^n, \quad , n^{\ln n}, \quad 2^n, \quad n!, \quad \binom{2n}{n}, \quad \ln(n!), \quad \ln \binom{2n}{n}.$$

You may use the Stirling approximation: $n! \approx \sqrt{2\pi n}(n/e)^n$.

3.2 Programming a Turing machine.

a) Construct a Turing machine for the solution of this problem: if the input bits $x_0, \ldots, x_{n-1}$ represent a binary number u written backwards (i.e., $u = x_0 + 2x_1 + \cdots + 2^{n-1}x_{n-1}$), increment that number by 1. Note that this problem is defined using the input-output alphabet $A = \{0, 1\}$. The internal alphabet is your choice, but there is actually no need for additional characters except the blank symbol.

b) Construct a Turing machine that duplicates each symbol of the input word: $x_0x_1 \ldots x_{n-1}$ should be transformed into $x_0x_0x_1x_1 \ldots x_{n-1}x_{n-1}$, where $x_j = 0, 1$. Again, the internal alphabet $S = \{0, 1, \text{␣}\}$ will suffice.

c) Argue that any computation can be done on a Turing machine with the internal alphabet $\{0, 1, \text{␣}\}$.

3.3 The threshold function.

By definition, a probabilistic algorithm may produce a wrong result with some probability $p_1 < 1/2$. To decrease the chance of error, one generally needs to run the algorithm several times and choose the result that occurs most frequently. For simplicity, let us assume that the output of each run, x_j is a single bit. Then the final result is given by the majority function $\mathrm{MAJ}_n(x_1, \ldots, x_n)$, which is equal to 1 if and only if more than a half of its arguments are 1. Let us define a more general function:

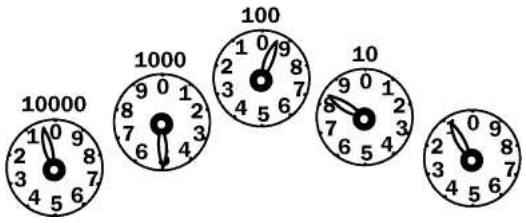
$$\mathrm{THRESHOLD}_{n,k}(x_1, \ldots, x_n) = \begin{cases} 0 & \text{if } \sum_{j=1}^n x_j < k, \\ 1 & \text{if } \sum_{j=1}^n x_j \geq k. \end{cases} \tag{1}$$

The problem is to implement it as a Boolean circuit.

a) We first try the simplest, though not the most efficient method. Let us compute $\sum_{j=1}^{n} x_j$ using this encoding: a number $y \in \{0, \ldots, n\}$ is represented as $\underbrace{1 \ldots 1}_{y} \underbrace{0 \ldots 0}_{n-y}$. Then the threshold function is just the k-th bit of the sum. Describe and/or draw the circuit corresponding to this calculation; determine its size and depth.

b) Compute $\sum_{j=1}^{n} x_j$ using the standard binary encoding and the parallelization tricks discussed in class. You need to squeeze into these bounds: size $\leq O(n \log n)$, depth $\leq O(\log n)$. It should straightforward to modify your solution so that to compute $\sum_{j=1}^{n} x_j - k$ and compare the result with 0. (As is usual, negative numbers are represented using the complementary code, e.g., $-2 = \overline{1 \ldots 110}$.)

3.4 Measurement decoding. (Strange as it sounds, this problem has important applications in quantum computation. We will use the algorithm in class!)
Some electricity meters look like this:



Apparently, it takes a special skill to read them quickly. Let us consider a mathematical version of the meter reading problem. We will assume that the dials have eight digits instead of ten and rotate at speeds that differ by powers of 2. The meter may roll over the highest value, which is equal to 1.

Let x be a real number *modulo* 1, i.e., we do not distinguish between numbers that differ by an integer. We may also think of x as a point on the circle that is obtained from the interval $[0, 1]$ by identifying the points 0 and 1. However, the actual value of x is unknown. Instead, we are given individual dial readings $s_0, \ldots, s_{n-1} \in \{0, 1, 2, 3, 4, 5, 6, 7\}$ such that

$$2^j x \in \Big(\frac{s_j - 1}{8}, \frac{s_j + 1}{8}\Big) \pmod 1 \qquad \text{for } j \in \{0, \ldots, n-1\}, \tag{2}$$

where the notation $(a, b) \pmod 1$ stands for the interval from a to b on the circle. Assuming that the data are consistent, we need to find a number $y = \overline{.y_1 \ldots y_{n+2}} = \sum_{j=1}^{n+2} 2^{-j} y_j$ (where $y_j \in \{0, 1\}$) such that

$$x - y \in \big(-2^{-(n+2)}, 2^{-(n+2)}\big) \pmod 1. \tag{3}$$

a) Construct a circuit of size $O(n)$ for the solution of this problem. (Describe the circuit in reasonable detail, e.g., as a composition of blocks operating with a constant number of bits.)

b) Parallelize the computation by analogy with the parallel addition algorithm discussed in class. Specifically, construct a circuit of size $O(n)$ and depth $O(\log n)$. You don't have to draw the whole circuit. Just describe intermediate data (like the 0-1-p values in the addition algorithm), elementary blocks, and the way they are grouped in levels.