| | |
|---|---|
| **Ph/CS 219, Fall 2006**<br>**Quantum Computation**<br>**Prof. Alexei Kitaev** | **Problem set #3**<br>**Due Friday, November 17, 2006** |

**3.1 The $O(\cdots)$ notation.** We have used this notation in class: $f(n) \le O(g(n))$ if there are some constants $c$ and $n_0$ such that $f(n) \le cg(n)$ for all $n \ge n_0$. The $O$ notation allows one to compare the asymptotic growth of different functions: $f \le O(g)$ 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 $f \le O(g)$ nor $g \le O(f)$, 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., "$\le$" 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: $x_0 x_1 \ldots x_{n-1}$ should be transformed into $x_0 x_0 x_1 x_1 \ldots x_{n-1} x_{n-1}$, where $x_j = 0, 1$. Again, the internal alphabet $S = \{0, 1, \sqcup\}$ will suffice.

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

**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:

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

The problem is to implement it as a Boolean circuit.