Lecture 3

Leftover comment from last time: highlight some important keywords to be familiar with

1. two-qubit gate, (as opposed to one-qubit gate, two-qubit gate count is the more important metric),
2. two-qubit gate error, f: this is the probability of failure when one two-qubit gate is applied; conversion to number of gates that can be performed $1/f$. Comment: then did probability calculation.
3. physical qubits, or just qubits = number of "elementary quantum units" (electrons/photons/superconducting circuits)
4. logical qubits: conversion is number of logical qubits = number of physical qubits times $100 - 1000$. (The factor arises from a concept of error correction, that we'll discuss towards the end of this course.)

Power of randomized computation. Quantum computation can be seen as a generalization of randomized computation (complex generalizes non-negative numbers.) It is important to distinguish between when a problem's speedup is due to randomness vs due to quantumness. Can do interesting things with randomness alone.

Consider the following two problems:

1. Given a string of n bits that's either all zeros or half zero and half one but you don't know where they're placed: decide which is the case. Deterministic $\Omega(n)$ in the worst case (with respect to the worst-case input). Randomized $O(1)$ (for *very* high probability of success). Comment: then did the randomized analysis. Comment: there was a question about whether this is an "apples-to-apples" comparison. I answered that you can also get separations in a fully apples-to-apples case. This is the wrong answer, sorry. (I was thinking about "Las Vegas" algorithms: while they are always correct, their runtime bound is only "in expectation", so the complexity comparison there with deterministic classical is also not apples-to-apples.) In fact, the answer for a *fully* apples-to-apples comparison is "no, randomized algorithms cannot be faster if you demand both *always* correct and *and* a runtime bound that *always* holds". Interestingly, it is true that "quantum algorithms *can* be faster even if you demand both *always* correct and *and* a runtime bound that *always* holds" – in fact, a relatively simple one that we'll see: Deutsch-Jozsa.

2. NAND tree on $n := 2^h$ variables. Randomized: consider the following recursively-defined randomized algorithm $\mathcal{A}_h$: choose the left or right branch uniformly at random. Then compute the value of that branch using $\mathcal{A}_{h-1}$. If get 0, just outputs 1 — this will give the correct answer by the truth-table of NAND. If get 1, also compute the value of the other remaining branch using $\mathcal{A}_{h-1}$. Comment: $\mathcal{A}_h$ is computing the NAND tree on 2^h variables.

   For $b \in \{0,1\}$, let $\alpha_b(h)$ denote the algorithm's expected complexity (expectation is over the randomness of the algorithm *not* the input, the input is assumed to be worst-case) when run on inputs $x \in \{0,1\}^{2^h}$ that map to b. Then,

$$\alpha_0(h) \leq 2\alpha_1(h-1), \tag{6}$$

$$\alpha_1(h) \leq \frac{1}{2}(\alpha_1(h-1) + \alpha_0(h-1)) + \frac{1}{2}\alpha_0(h-1) \leq \alpha_0(h-1) + \alpha_1(h-1)/2; \tag{7}$$

   which solves to

$$\left((1+\sqrt{33})/4\right)^h = O(n^{0.754}). \tag{8}$$

   In fact this is the optimal randomized complexity – see Saks and Wigderson '86. Deterministic: can show it's $\Omega(n)$. It turns out that the answer is $\Theta(\sqrt{n})$ for quantum. So this is an interesting problem with a "three-way" complexity separation between deterministic, randomized, and quantum.