Ph/CS 219a, Fall 2022
Quantum Computation **Solutions to problem set #5**
Prof. Alexei Kitaev

1. The $O(\cdots)$ notation. [5 points] *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., "$\lesssim$" stands for the O-relation and "$\sim$" 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: $n! \approx \sqrt{2\pi n}(n/e)^n$.

Let us transform some of the functions in question:

- $n\log n = cn\ln n$ (more precisely, $n\log_a n = n\frac{\ln n}{\ln a}$). Using the "$\sim$" sign for the bidirectional O-relation, we may write "$n\log n \sim n\ln n$". One must be careful, though – see the next item.
- Note that $\log_2 n = (\log_2 e)\ln n$, therefore $(\log_2 n)^n = c^n(\ln n)^n$, where and $c = \log_2 e > 1$. So we have this asymptotic inequality: $(\ln n)^n \lesssim (\log_2 n)^n$.
- $(\ln n)^n = e^{(\ln\ln n)n}$. This function grows faster than $a^n = e^{(\ln a)n}$ for any constant a.
- $n^{\ln n} = e^{(\ln n)^2}$. This function grows faster than any constant power of n, yet not as fast as a^n (where $a > 0$).
- $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ using Stirling's formula, where the "$\approx$" sign means that the ratio of the two functions tends to 1 as n goes to infinity. Therefore, $n! \gtrsim (\log n)^n$.
- $\ln(n!) \approx \ln\sqrt{2\pi n} + n(\ln n - 1) \approx n\ln n$.
- $\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \approx \frac{4^n}{\sqrt{\pi n}}$. This number is asymptotically greater than 2^n but smaller than 4^n.
- $\ln\binom{2n}{n} \approx -\ln\sqrt{\pi n} + (\ln 4)n \approx (\ln 4)n \sim n$.

Putting all these together, the ordering of the functions is

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

2. The threshold function. [15 points] *To decrease the chance of error in a probabilistic computation, 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*. Now, let us define a more general function:*

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

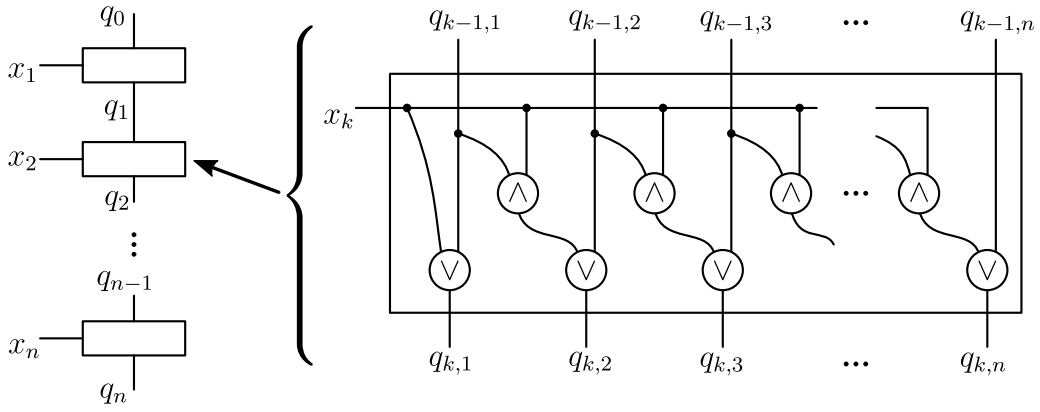
The problem is to implement it by a Boolean circuit.

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) A better strategy is this: we compute $\sum_{j=1}^{n} x_j - k$ *in binary and compare the result with* 0*. (Negative numbers are represented using the two's complement, e.g.,* $-2 = \overline{1 \ldots 110}$*.) Note that the addition of* $-k$ *and the comparison are relatively cheap; the main work is to compute the sum* $\sum_{j=1}^{n} x_j$*. Estimate the size of the resulting circuit in two cases: (i) when the numbers* $x_1, \ldots, x_n$ *are added sequentially and (ii) when the addition is organized in a binary tree.*

c) In the last case, apply the parallelization tricks discussed in class (namely, the logarithmic depth adder and the parallel 3-2 adder) and estimate the total depth. You are not required to recalculate the size, but please make sure that it only changes by a constant factor.

a) There are a few ways of doing this part, one of which is to think of this as sorting the input string of $x_1, \ldots, x_n$ into a string with all the 1's in front of all the 0's. Another way of doing this is to start with a string of n 0's, and shift 1's rightwards when the input is a 1. The circuit that does this is shown below, where $q_0 = 00 \ldots 0$ is a string of n 0's, and $q_k = q_{k,1} q_{k,2} \ldots q_{k,n}$ is the encoding of $y_k = \sum_{j=1}^{k} x_j$ as $\underbrace{1 \ldots 1}_{y_k}\underbrace{0 \ldots 0}_{n-y_k}$. In particular, q_n is the string representing $\sum_{j=1}^{n} x_j$ as required. The k-th bit of q_n gives the value of the THRESHOLD function.

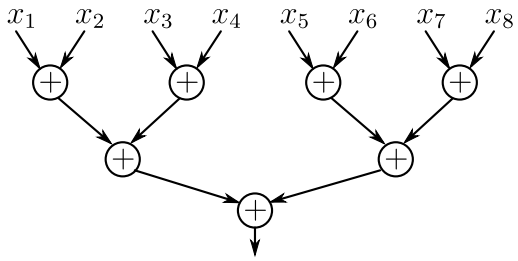


Each box in this circuit is of size $O(n)$ and constant depth, and we have n boxes in sequence. Therefore, the total circuit size is $O(n^2)$ and depth $O(n)$.

b) Let us first compute the numbers $y_l = \sum_{j=1}^{l} x_j$ sequentially. We have the obvious upper bound $y_l \le l$, therefore y_l can be represented by $\lceil \log_2(l+1) \rceil$ bits. The circuit that computes y_l has size $O(\log l)$. Summing over l, we get

$$\text{total size} = \sum_{l=1}^{n} O(\log l) \le O(n \log n). \tag{2}$$

Alternatively, we can organize the addition into a binary tree like this:



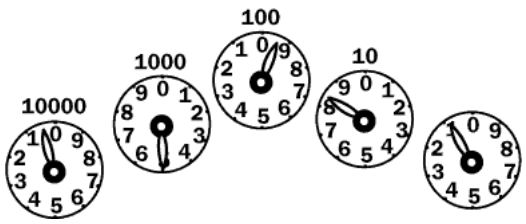
(In general, we pad the string $x_1 \dots x_n$ with zeros to make a total of 2^s input bits, where $s = \lceil \log_2 n \rceil$.) At level l, we have 2^{s-l} addition circuits, each taking a pair of l-bit numbers and producing an $(l+1)$-bit number. Thus,

$$\text{total size} = \sum_{l=1}^{s} 2^{s-l} O(l) \le \text{const} \cdot 2^s \sum_{l=1}^{\infty} \frac{l}{2^l} = O(2^s) = O(n). \tag{3}$$

c) When using the binary tree scheme, we can make the following adjustments to reduce the depth. First, we represent any intermediate number y by a pair of numbers, y' and y'' such that $y' + y'' = y$. To add two numbers represented like that, $y = y' + y''$ and $z = z' + z''$, we reduce the sum of four numbers to a sum of two numbers by using the parallel 3-2 adder twice. This is a constant depth circuit, therefore each layer of the binary tree contributes $O(1)$ to the total depth. The tree has $O(\log n)$ layers.

At the very end, we actually add the last pair of numbers (of length $l = O(\log n)$) together. The standard addition circuit and the logarithmic depth adder have depth $O(l) = O(\log n)$ and $O(\log l) = O(\log \log n)$, respectively. In this case, the differece is not critical: either way, the total depth is $O(\log n)$.

3. Measurement decoding. [Extra credit 15 points] *Some old 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*

$$2^j x \in \Big(\frac{s_j - 1}{8},\, \frac{s_j + 1}{8}\Big) \pmod 1 \qquad \text{for } j \in \{0, \ldots, n-1\}, \tag{4}$$

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*

$$x - y \in \big(-2^{-(n+2)},\, 2^{-(n+2)}\big) \pmod 1. \tag{5}$$

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-carry *values in the addition algorithm), elementary blocks, and the way they are grouped in levels.*

We are given n readings $s_0, s_1, \ldots, s_{n-1} \in \{0, 1, \ldots, 7\}$ from n meters differing in rotation speeds by a factor of 2, with the s_0 meter being the slowest. We are promised that these readings are consistent estimates of a number x between 0 and 1 such that $2^j x \pmod 1 \in \left(\frac{s_j - 1}{8}, \frac{s_j + 1}{8}\right)$. Let us define $\beta_j := \frac{s_j}{8}$, then,

$$\big|(2^j x - \beta_j)_{\text{mod } 1}\big| < \frac{1}{8} \qquad \text{for } j = 0, 1, \ldots, n-1.$$

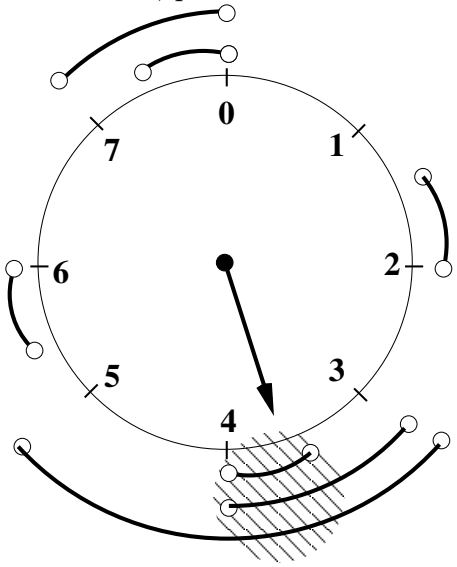
The problem is to combine the overlapping information from the n readings to get an estimate y of x such that $|(x - y)_{\text{mod } 1}| < 2^{-(n+2)}$.

First, a simple fact that we will use: suppose $|(2a - \overline{.b_1 b_2 b_3 \ldots})_{\text{mod } 1}| < \delta < \frac{1}{2}$, then exactly one of the following is true:

$$\begin{aligned} \text{either} \qquad & \big|\big(a - \overline{.0 b_1 b_2 \ldots}\big)_{\text{mod } 1}\big| < \delta/2 \\ \text{or} \qquad & \big|\big(a - \overline{.1 b_1 b_2 \ldots}\big)_{\text{mod } 1}\big| < \delta/2. \end{aligned} \tag{6}$$

For each j, let us write $\beta_j := \overline{.b_{j1} b_{j2} b_{j3}}$ (note that we only need three digits since $\beta_j = \text{integer}/8$). Given $|(2^j x - \beta_j)_{\text{mod } 1}| < 1/8$, using (6), we also know that either $|(2^{j-1} x - \overline{.0 b_{j1} b_{j2} b_{j3}})_{\text{mod } 1}| < 1/16$ or $|(2^{j-1} x - \overline{.1 b_{j1} b_{j2} b_{j3}})_{\text{mod } 1}| < 1/16$. Repeating this procedure, we eventually get 2^j possible regions, each of length $2^{-(j+2)}$, in which x can lie which are consistent with the given value of β_j. Since the data are consistent, there will be a finite region contained in the intersection

Figure 1: s_0 meter, pointer indicates value of x.



of the sets of regions from the different j values. This region is of length $2^{-(n+1)}$ and contains x. If we take y as the midpoint of this region, then $|(x-y)_{\text{mod }1}| < 2^{-(n+2)}$.

Let us look at a specific example. Consider $x = \overline{.011101}$, which we know just to check our results. Take $n = 3$. Then, we can have the data $s_0 = 4, s_1 = 7, s_2 = 7$. We can check that this set of data satisfies the required conditions:

$$\left|\left(x - \frac{s_0}{8}\right)_{\text{mod }1}\right| = \left|\overline{.011101} - \overline{.100}\right| = \left|-\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{64}\right| = \frac{3}{64} < \frac{1}{8},$$
$$\left|\left(2x - \frac{s_1}{8}\right)_{\text{mod }1}\right| = \left|\overline{.11101} - \overline{.111}\right| = \frac{1}{64} < \frac{1}{8},$$
$$\left|\left(2^2x - \frac{s_2}{8}\right)_{\text{mod }1}\right| = \left|\overline{.1101} - \overline{.111}\right| = \left|-\frac{1}{8} + \frac{1}{16}\right| = \frac{1}{16} < \frac{1}{8}.$$

Using (6) repeatedly, from s_2, we get four possible regions in which x can lie, each of length $1/16$ (see Fig. 1), from s_1, two regions, each of length $1/8$, and from s_0, one region of length $1/4$. The shaded region (of length $1/16$ as constrained by the region coming from s_2) corresponds to the overlap between all the different s_j values. As can be seen from the diagram, x indeed lies in this region, and we get the desired precision if we take y as the midpoint of the shaded region.

(a) How do we get the correct value of y with circuit size $O(n)$? The problem is, of course, that there are exponentially many regions. Fortunately, we do not need to list all of them.[1] Let us go back to our original argument. The inequalities $|(2^{j-1}x - \overline{.0b_{j1}b_{j2}b_{j3}})_{\text{mod }1}| < 1/16$

[1] A subtle point to note is that there is more than one possible set of values for $\{s_j\}$ that is consistent with a given x. For instance, in the example above, s_0 could have been 3 instead of 4, and we would still get the same shaded region. This causes the following seemingly plausible strategy to *fail* to give the correct y: write each β_j in the binary fraction representation $\beta_j = \overline{.b_{j1}b_{j2}b_{j3}}$ and take $y = \overline{.b_{01}b_{11}b_{21}\ldots b_{n-2,1}b_{n-1,1}b_{n-1,2}b_{n-1,3}}$, i.e. start with $\beta_{n-1} = \overline{.b_{n-1,1}b_{n-1,2}b_{n-1,3}}$, and prepend with the first digit of $\beta_{n-2}, \beta_{n-3}, \ldots, \beta_0$. In our example,

and $|(2^{j-1}x - \overline{.1b_{j1}b_{j2}b_{j3}})_{\mathrm{mod}\ 1}| < 1/16$ define two arcs of length $1/8$ on opposite sides of the circle. Only one of the arcs can overlap with the region defined by the inequality $|(2^{j-1}x - \beta_{j-1})_{\mathrm{mod}\ 1}| < 1/8$ because the length of this region $(1/4)$ is smaller than the gaps between the arcs $(3/8)$. Therefore, we can select the right arc before further multiplying the number of choices. This observation suggest the following algorithm, in which we work out the bits of y *right to left*.

Let us start with the binary representation of $\beta_{n-1} := \overline{.y_n y_{n+1} y_{n+2}}$. Now, sequentially pick y_k, $k = n-1, n-2, \ldots, 1$ such that

$$y_k = \begin{cases} 0 & \text{if } \left(\overline{.0y_{k+1}} - \beta_{k-1}\right)_{\mathrm{mod}\ 1} \in \left\{-\frac{2}{8}, -\frac{1}{8}, \frac{0}{8}, \frac{1}{8}\right\}, \\ 1 & \text{if } \left(\overline{.1y_{k+1}} - \beta_{k-1}\right)_{\mathrm{mod}\ 1} \in \left\{-\frac{2}{8}, -\frac{1}{8}, \frac{0}{8}, \frac{1}{8}\right\}. \end{cases} \tag{7}$$

(Note that exactly one of these two conditions holds.) Then, take $y = \overline{.y_1 y_2 \ldots y_n y_{n+1} y_{n+2}}$.

We need to check that y found this way gives the required precision for x, i.e., $|(x-y)_{\mathrm{mod}\ 1}| < 2^{-(n+2)}$. We prove this using induction on the statement:

$$\left|\left(\overline{.y_k y_{k+1} \ldots y_{n+2}} - 2^{k-1}x\right)_{\mathrm{mod}\ 1}\right| < \delta_k, \qquad \text{where } \delta_k = 2^{-(n+3-k)}. \tag{8}$$

Base case. If $k = n$ then

$$\left|\left(\overline{.y_n y_{n+1} y_{n+2}} - 2^{n-1}x\right)_{\mathrm{mod}\ 1}\right| = |(\beta_{n-1} - 2^{n-1}x)_{\mathrm{mod}\ 1}| < 2^{-3} = \delta_n.$$

Induction step. Assume that equation (8) is true for some k; we need to prove it for $\tilde{k} = k-1$. The inductive assumption implies that

$$\left|\left(\overline{.uy_k y_{k+1} \ldots y_{n+2}} - 2^{k-2}x\right)_{\mathrm{mod}\ 1}\right| < \frac{\delta_k}{2} = \delta_{k-1} \tag{9}$$

for $u = 0$ or $u = 1$. We just need to check that (7) gives the correct value for y_{k-1}, i.e., $y_{k-1} = u$. To see this, notice that

$$0 \le \overline{.uy_k y_{k+1} \ldots y_{n+2}} - \overline{.uy_k} \le \sum_{j=3}^{n+4-k} 2^{-j} = 1/4 - \delta_{k-1}. \tag{10}$$

Using (9) and (10) together with the data consistency condition, $|(2^{k-2}x - \beta_{k-2})_{\mathrm{mod}\ 1}| \le 1/8$, we get:

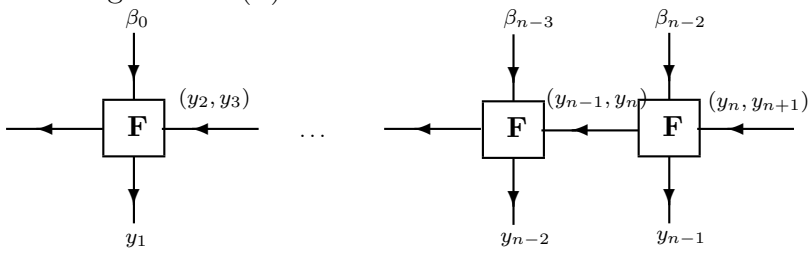
$$-\delta_{k-1} - (1/4 - \delta_{k-1}) - 1/8 \ < \ (\overline{.uy_k} - \beta_{k-2})_{\mathrm{mod}\ 1} \ < \ \delta_{k-1} - 0 + 1/8.$$

Hence

$$(\overline{.uy_k} - \beta_{k-2})_{\mathrm{mod}\ 1} \in \{-2/8, -1/8, 0, 1/8\} \tag{11}$$

if the data is given as $s_0 = 3, s_1 = 7, s_2 = 7$, this strategy gives $y = \overline{.01111}$ which is within $1/16$ of the value of x. However, if the data is given as $s_0 = 4, s_1 = 7, s_2 = 7$, we get $y = \overline{.11111}$, which is *not* within the desired precision. It is easy to see that this strategy only works if we are given s_j values that are the nearest tick marks counterclockwise from the pointer for $2^j x$ on the respective meters.

Figure 2: $O(n)$ circuit as a finite-state automaton.



as required.

In this algorithm, y_k depends only on β_k and y_{k+1}. Proving the correctness by induction is somewhat tricky, though. The proof may be easier to carry out if we replace condition (7) by, say, this inequality:

$$\left|\left(\overline{.y_k y_{k+1} y_{k+2} 1} - \beta_{k-1}\right)_{\bmod 1}\right| < 1/4. \tag{12}$$

In this case, y_k depends on β_k, y_{k+1}, and y_{k+2}. However, the overall structure of the algorithm remains the same. It can be viewed as a finite-state automaton $F : (\beta, q) \mapsto (y, q)$ (where $\beta \in \{0, 1\}$, $y \in \{0, 1\}$, and $q = (y', y'') \in \{0, 1\}^2$ denotes the state of the automaton) defined as:

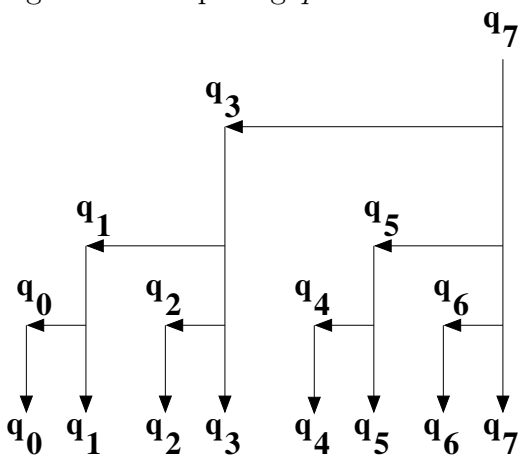
$$F(\beta_{k-1}, (y_{k+1}, y_{k+2})) = (y_k, (y_k, y_{k+1}))$$

with y_k given by (12). This is represented schematically in Fig. 2. The initial state of the automaton is computed from β_{n-1} (which can be done with a constant size circuit). Notice that F is computable by a fixed size circuit, but we need to repeat it n times in sequence, so the circuit complexity is given by

$$\begin{aligned} \text{size} &= O(n) \\ \text{depth} &= O(n). \end{aligned}$$

(b) From part (a), we know that our algorithm can be viewed as a finite-state automaton operating n times in sequence. This point of view makes it very easy to see how to parallelize the circuit, just by analogy with the parallelization of addition discussed in class. Recall that in parallelizing addition, we compute all the carry-bits with depth $O(\log n)$ by first computing the composition of functions given the input in the form of a tree of depth $O(\log n)$. Here, the automaton state $q_k := (y_{k+1}, y_{k+2})$ plays the role of the carry bit, which we need to compute the next value of y_k. We carry out the exact same procedure as in the case of parallelizing addition. We first compute all compositions of the functions $F_{\beta_k} : Q \longrightarrow Q$ in the form of a tree. This tree is of depth $O(\log n)$, with $O(n)$ nodes, each node corresponding to a composition of two fixed size functions (and hence computable by a circuit with constant size and depth). Hence this composition process has circuit size $O(n)$ and depth $O(\log n)$. Having gotten all these composition functions, we need to compute $q_k \forall k$. This can be done in $O(\log n)$ steps, just like in the

Figure 3: Computing q values for n=8.



case of computing the carry bits in the parallel addition algorithm (see Figure 3 for an example for $n = 8$). Finally, we compute in parallel, all the y_k values given the q_k values. Altogether then, the circuit complexity is:

$$\begin{aligned} \text{size} &= O(n), \\ \text{depth} &= O(\log n). \end{aligned}$$

Note that all finite-state automaton algorithms of the above form can be parallelized in the same way to reduce the depth from $O(n)$ to $O(\log n)$. For further reference, see the solution to KSV problem 2.11 (for simplicity, you can take the parameter k in that problem to be a constant).