Homework #1

(Due: Oct 6)

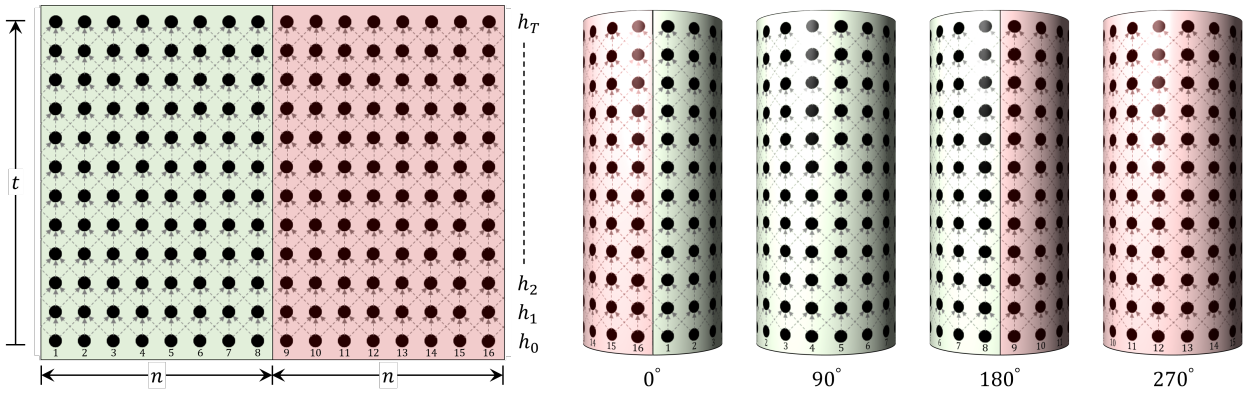

Figure 1: [Task 1] Our $2n \times (T+1)$ space-time grid can be viewed as wrapped around a cylinder making the first and the last columns of the grid neighbors.

Task 1. [50 Points] 1D Heat Diffusion in a Segmented Ring

Consider the following partial differential equation that describes heat diffusion in 1D space:

$$\frac{\partial h_t(x)}{\partial t} = \alpha \left(\frac{\partial^2 h_t(x)}{\partial x^2} \right),$$

where, $h_t(x)$ is the heat at a point x at time t and α is the thermal diffusivity of the material.

By discretizing space and time, the equation above can be solved approximately by using the following update equation:

$$h_{t+1}(x) \quad = \quad h_t(x) + \frac{\alpha \Delta t}{\Delta x^2} \left(h_t(x-1) + h_t(x+1) - 2h_t(x) \right)$$

Given a 1D grid h of length n with heat values at time step 0, Figure 2 shows how to update h to heat values at time step T under periodic boundary conditions, that is, assuming that the spatial dimension of the grid wraps around and thus forms a ring. Clearly, the algorithm runs in $\Theta(nT)$ time, that is, in $\Theta\left(n^2\right)$ time when $T = n$.

In this task, however, we are interested in heat diffusion in a ring that is composed of two segments of equal length connected end to end. The two segments are made of two different materials and thus have different thermal diffusivity values. Figure 3 shows how to update such a grid h of length $2n$ to compute heat values at time step T under periodic boundary conditions. We assume that the first and second halves ($h[1..n]$ and $h[n+1..2n]$, respectively) of the grid are made of different

```
1D-Heat-Diffusion-1-Seg( h, n, T, α̅ )   {Given a 1D grid h of length n with heat values at
                                          time step 0 and α̅ = αΔt/Δx², where α is the thermal
                                          diffusivity of the material, update h to the heat values
                                          at time step T under periodic boundary conditions.}
 1. allocate new grid g[1..n]
 2. for t = 1 to T do
 3.     g[1] ← h[1] + α̅ · (h[n] − 2h[1] + h[2])
 4.     g[n] ← h[n] + α̅ · (h[n − 1] − 2h[n] + h[1])
 5.     for x = 2 to n − 1 do
 6.         g[x] ← h[x] + α̅ · (h[x − 1] − 2h[x] + h[x + 1])
 7.     for x = 1 to n do
 8.         h[x] ← g[x]
 9. deallocate g
```

Figure 2: [Task 1] Heat diffusion in a 1D grid under periodic boundary conditions assuming that the entire grid is made of the same material.

```
1D-Heat-Diffusion-2-Seg( h, n, T, α̅₁, α̅₂ )   {Given a 1D grid h of length 2n with heat values at time
                                                step 0, α̅₁ = α₁Δt/Δx² and α̅₂ = α₂Δt/Δx², update h to
                                                heat values at time step T under periodic boundary conditions.
                                                We assume that h[1..n] and h[n + 1..2n] are made of different
                                                materials with thermal diffusivity α₁ and α₂, respectively.}
 1. allocate new grid g[1..2n]
 2. for t = 1 to T do
 3.     g[1] ← h[1] + α̅₁ · (h[2n] − 2h[1] + h[2])
 4.     g[2n] ← h[2n] + α̅₂ · (h[2n − 1] − 2h[2n] + h[1])
 5.     for x = 2 to n do
 6.         g[x] ← h[x] + α̅₁ · (h[x − 1] − 2h[x] + h[x + 1])
 7.     for x = n + 1 to 2n − 1 do
 8.         g[x] ← h[x] + α̅₂ · (h[x − 1] − 2h[x] + h[x + 1])
 9.     for x = 1 to 2n do
10.         h[x] ← g[x]
11. deallocate g
```

Figure 3: [Task 1] Heat diffusion in a 1D grid under periodic boundary conditions assuming that the grid is composed of two segments made of two different materials.

materials and so will have to be updated differently. Since the boundary condition is periodic, the entire space-time grid can be viewed as wrapped around a cylinder as shown in Figure 1. This algorithm, too, runs in $\Theta(nT)$ time, that is, in $\Theta\left(n^2\right)$ time when $T = n$.

Now, given a 1D grid $h[1..2n]$ composed of two segments of equal length as explained above and initialized with heat values at time step 0, design a recursive divide-and-conquer algorithm to calculate the heat values at time step $T = n$ in $\mathcal{O}\left(n \log^2 n\right)$ time. You can assume that n is a power of 2.

Task 2. [50 Points] Beyond 2-way Divide and Conquer for Computing DFT

The FFT algorithm we have seen in the class uses 2-way recursive divide and conquer to compute the Discrete Fourier Transform (DFT) of an input vector. This task asks you to extend the algorithm to decompose the problem into more than two sub-problems in each recursive call.

(*a*) [**20 Points**] Assuming that the length n of the input vector is a power of 3, that is, $n = 3^k$ for some nonnegative integer k, extend the FFT algorithm we have seen in the class to use 3-way recursive divide and conquer. Analyze its running time.

(*b*) [**20 Points**] Assuming that $n = b^k$ for some integer $k \geq 0$ and a fixed integer $b \geq 2$, extend the FFT algorithm to use b-way recursive divide and conquer. Express its asymptotic run time in terms of both n and b, that is, do not eliminate b treating it as a constant.

(*c*) [**10 Points**] Is it possible to reduce the run time of the algorithm to $o(n \log n)$ by choosing an appropriate fixed value for b? Explain.

Task 3. [40 Points] Multiplying 3D Matrices

Let's define the product of three $n \times n \times n$ matrices X, Y, and Z as another $n \times n \times n$ matrix W such that for all integers $i, j, k \in [1, n]$,

$$\boxed{w_{i,j,k} = \sum_{1 \leq p,q \leq n} x_{i,p,q} \cdot y_{q,j,p} \cdot z_{p,q,k}}$$

Thus, as shown in Figure 4, each entry of W is obtained as the sum of the pointwise products of three planes of X, Y, and Z. Two $n \times n \times n$ matrices can be added in $\Theta\left(n^3\right)$ time using standard pointwise additions of cell values the way we do it for 2D matrices.

(*a*) [**20 Points**] Give a recursive divide-and-conquer algorithm for computing matrix W in $\Theta\left(n^5\right)$ time.

(*b*) [**20 Points**] Suppose $z_{i,j,k} = 1$ for all $i, j, k \in [1, n]$. Show that in that case, W can be computed in $\mathcal{O}\left(n^{\log_2 14}\right)$ time.

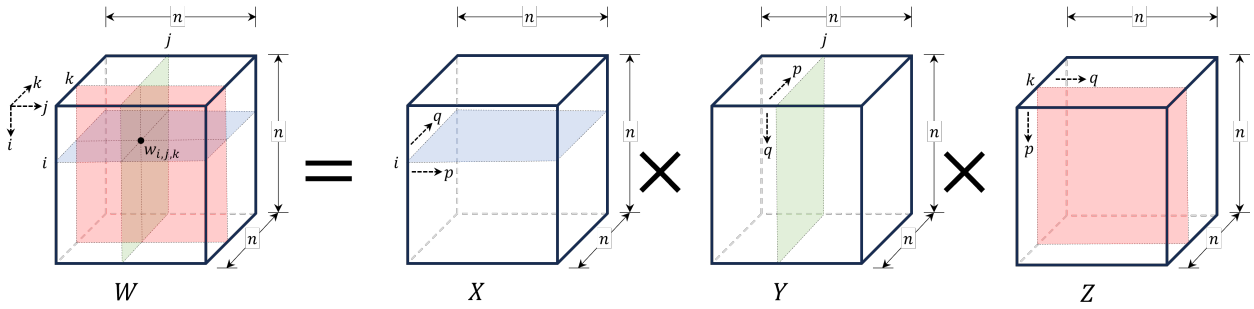

Figure 4: [Task 3] Entry $w_{i,j,k}$ of the 3D matrix W is obtained as the sum of the pointwise products of three planes of the 3D matrices X, Y, and Z.

Task 4. [40 Points] Integer Multiplication

Suppose that you are given two n-bit integers x and y that contain n_x and n_y nonzero bits, respectively. This task asks you to explore whether you can take advantage of this additional information to asymptotically improve the run times of integer multiplication algorithms.

(a) [**20 Points**] Design a nonrecursive algorithm to multiply x and y in $\mathcal{O}\left((n + n_x n_y) \log n\right)$ time. Observe that this algorithm asymptotically improves over the standard $\Theta\left(n^2\right)$ time algorithm when $n_x = o\left(\frac{n}{\log n}\right)$ and/or $n_y = o\left(\frac{n}{\log n}\right)$.

(b) [**10 Points**] Can you asymptotically improve the run-time complexity of the $\Theta\left(n^2\right)$-time recursive divide-and-conquer algorithm we saw in the class for the problem in our current task? If yes, what bound do you get? You may use your result from part (a) as a subroutine if that helps.

(c) [**10 Points**] Repeat part (b) for Karatsuba's algorithm.

Task 5. [20 Points] Fun with Programming Christmas Lights! (Idea: Tianchi (Maverick) Mo)

This task asks you to do recursive divide and conquer on a string of decorative light bulbs!

You are given a string of $n = 2^m - 1$ fancy bulbs, where m is a positive integer. If you number the light bulbs by consecutive integers from left to right starting with 1, every even-numbered bulb is red and the odd-numbered ones are a mix of green and yellow (see Figure 5).

The bulbs are fancy because each has a simple processing chip. Each chip has a small constant number (independent of n) of storage bits. It also has an internal clock that ticks at a fixed rate. With every tick of the clock, the chip reads its own bits and the storage bits of its immediate neighbors. Based on what it has read, it can change its own storage bits and turn its bulb on or off right before the next tick.

All chips in the string are synchronized, and so the clocks tick at the same time and at the same rate. All chips read the storage bits at exactly the same time with every clock tick before the bits are updated before the next clock tick.

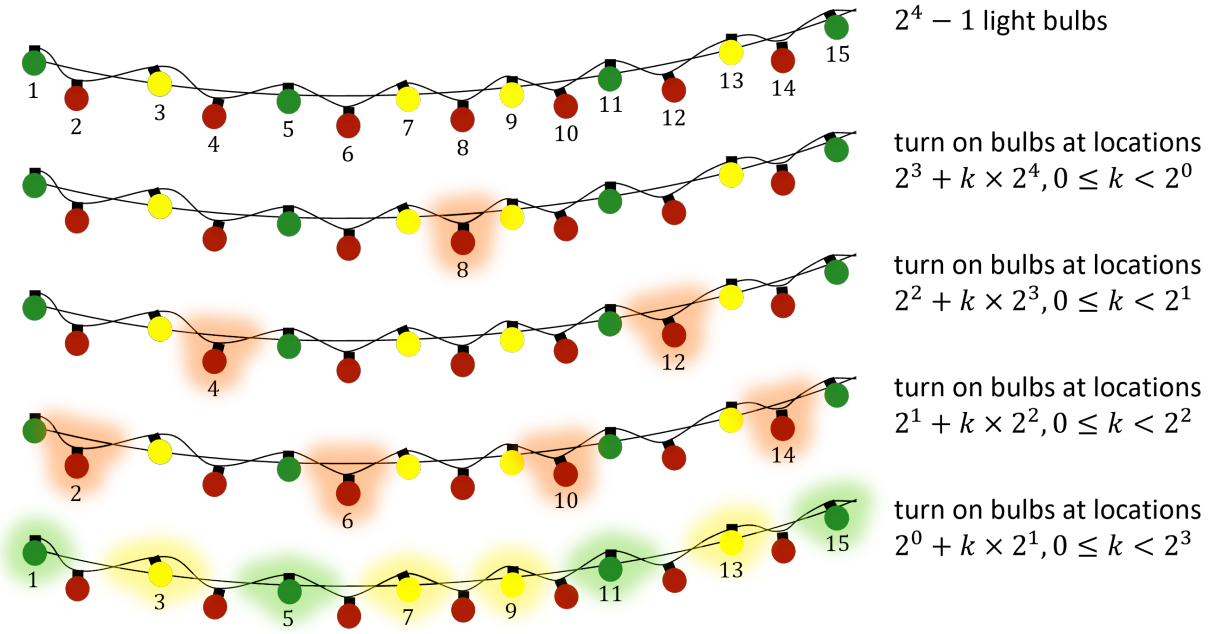

Figure 5: [Task 5] An example with $2^4 - 1 = 15$ light bulbs.

The chips are programmable. You can program what the chips do after reading the bits at every clock tick. All chips must execute the same code. You can set the storage bits of the leftmost chip (i.e., the one belonging to light bulb 1) before the execution starts, but all other chips start with their bits cleared to zero. When you turn on the light string, all the chips start executing right away.

Note that a chip knows neither n nor the index of the light bulb to which it belongs. Indeed, since it contains only a small constant number (say < 10) of storage bits, it does not have enough space to store either. Observe that storing n will require $\approx \log_2 n$ bits and an index can also be as large as n. Each chip executes the code locally, and there is no global variable accessible to every chip. A chip is only aware of its immediate neighbors, which is one or two depending on the location of the light bulb in the string. The leftmost and the rightmost light bulbs can infer that they are at those two extreme locations after trying and failing to read the storage bits of one of their neighbors.

Now answer the following questions.

(a) [**10 Points**] How do you program the chips so that after some number of clock ticks exactly one of the bulbs lights up and that bulb is exactly at the center of the string (i.e., at location 2^{m-1})? At which clock tick will that happen?

(b) [**10 Points**] Part (a) effectively divides the string into two parts of length $2^{m-1} - 1$ each. But your code must do more. Next, you must light up the two bulbs at the center of the two halves (i.e., at locations $2^{m-2} + k \times 2^{m-1}$ for $0 \le k < 2^1$), which, in turn, divides the string

into four equal parts; then you must light up the four bulbs at the center of those four parts (i.e., at locations $2^{m-3} + k \times 2^{m-2}$ for $0 \leq k < 2^2$), and so on. Observe that all these bulbs that you light up are at even-numbered locations and so are red, except in the last step, when you light up all bulbs only at odd-numbered locations, which are either green or yellow! At exactly what clock tick does your program light up all green and yellow lights?