Figure 1: [Task 1] When the partial differential equation describing 2D heat diffusion is discretized in time and space, the approximate heat value at any grid cell \((x, y)\) at time step \(t\) can be computed from the heat value at the four of its neighboring cells \((x, y - 1), (x, y + 1), (x - 1, y)\) and \((x + 1, y)\) at time step \(t - 1\).

Task 1. [60 Points] 2D Heat Diffusion

In this task we will consider the following partial differential equation that describes heat diffusion in two-dimensional space:

\[
\frac{\partial h_t(x, y)}{\partial t} = \alpha \left( \frac{\partial^2 h_t(x, y)}{\partial x^2} + \frac{\partial^2 h_t(x, y)}{\partial y^2} \right),
\]

where, \(h_t(x, y)\) is the heat at a point \((x, y)\) at time \(t\) and \(\alpha\) is the thermal diffusivity.

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

\[
h_{t+1}(x, y) = h_t(x, y) + \frac{\alpha \Delta t}{\Delta x^2} (h_t(x - 1, y) + h_t(x + 1, y) - 2h_t(x, y)) + \frac{\alpha \Delta t}{\Delta y^2} (h_t(x, y - 1) + h_t(x, y + 1) - 2h_t(x, y)).
\]

Given an \(N_x \times N_y\) grid \(h\) with heat values at time step 0, Figure 2 shows how to update \(h\) to heat values at time step \(T\) using periodic boundary conditions. Clearly, the algorithm runs in \(\Theta(N_x N_y T)\) time, i.e., \(\Theta(N^3)\) time when \(N_x = N_y = T = N\).

In this task you are required to do the following: given an \(N \times N\) grid with heat values at time step 0, show how to update the grid with heat values at time step \(N\) under periodic boundary conditions in \(\mathcal{O}(N^2 \log N)\) time by reducing the problem into a problem of multiplying polynomials. You can assume that \(N\) is a power of 2.
2D-Heat-Diffusion( \( h[0..N_x-1,0..N_y-1] \), \( T \)) \{ Given an \( N_x \times N_y \) grid \( h \) with heat values at time step 0, update \( h \) to heat values at time step \( T \) using periodic boundary conditions. \}

1. allocate new grid \( g[0..N_x-1,0..N_y-1] \)

2. \textbf{for} \( t = 1 \) to \( T \) \textbf{do}
3. \hspace{1em} \textbf{for} \( x = 0 \) to \( N_x - 1 \) \textbf{do}
4. \hspace{2em} \textbf{for} \( y = 0 \) to \( N_y - 1 \) \textbf{do}
5. \hspace{3em} \( g[x,y] \leftarrow h[x,y] + c_x (h[(x+1) \mod N_x,y] - 2h[x,y] + h[(x-1) \mod N_x,y]) \)
6. \hspace{3em} + \( c_y (h[x,(y+1) \mod N_y] - 2h[x,y] + h[x,(y-1) \mod N_y]) \)

7. \hspace{1em} \textbf{for} \( x = 0 \) to \( N_x - 1 \) \textbf{do}
8. \hspace{2em} \textbf{for} \( y = 0 \) to \( N_y - 1 \) \textbf{do}
9. \hspace{3em} \( h[x,y] \leftarrow g[x,y] \)

10. deallocate \( g \)

Figure 2: [Task 1] Implementation of a stencil computation for the 2D heat equation with periodic boundary conditions. The constants \( c_x = \alpha \Delta t/\Delta x^2 \) and \( c_y = \alpha \Delta t/\Delta y^2 \) are precomputed.

Task 2. [30 Points] Traceless In-place Selection

You are given an array \( A[1..n] \) of length \( n \) with each cell containing a \( \langle \text{height, weight} \rangle \) pair. All height values are distinct, and so are all weight values. The array is sorted in increasing order of the height values.

Your task is to design a recursive divide-and-conquer algorithm that given an integer \( k \in [1,n] \), finds the entry with the \( k^{th} \) smallest weight value. You are allowed to use only \( \mathcal{O}(1) \) extra space in every level of recursion. Though your algorithm is permitted to reorder the entries of \( A \) if required, it must restore the original order of the entries before termination. Your algorithm must run in \( \Theta(n) \) time.

Task 3. [50 Points] Blocking in Recursive Selection

Figure 3 shows a slightly generalized version of the selection algorithm we saw in the class. Instead of using a single block size (e.g., 5) at all levels of recursion, it uses block size \( s_{\text{rare}} \) at levels that are divisible by 3 (levels start from 1), and \( s_{\text{freq}} \) at all other levels. Now the base case size \( b \) is also a parameter to the algorithm. Observe that when \( b = 140 \) and \( s_{\text{rare}} = s_{\text{freq}} = 5 \), the algorithm reduces to the one we saw in the class.

(a) [10 Points] Write a recurrence relation describing the running time of \textsc{Select} on an array of size \( n \) assuming \( s_{\text{rare}} = s_{\text{freq}} = 3 \). What is the best running time you get by solving the
**Select** \((A[q:r], k, d, s_{freq}, s_{rare}, b)\)

**Input:** An array of distinct elements, and an integer \(k \in [1, r - q + 1]\). The parameter \(d\) is the depth of recursion (initially 1) with \(s_{rare}\) being the block size to be used at depths that are divisible by 3, and \(s_{freq}\) at all other depths. Also \(b\) is an upper bound on the size of the base case.

**Output:** An element \(x \in A[q:r]\) such that \(\text{rank}(x, A[q:r]) = k\).

1. \(n \leftarrow r - q + 1\)
2. **if** \(n \leq b\) **then**
   3. sort \(A[q:r]\)
   4. return \(A[q+k-1]\)
3. **else**
   6. **if** \(d \mod 3 = 0\) **then**
      7. \(s \leftarrow s_{rare}\)
   8. **else** \(s \leftarrow s_{freq}\)
   9. divide \(A[q:r]\) into blocks \(B_i\)'s each containing \(s\) consecutive elements (last block may contain fewer than \(s\) elements)
10. **for** \(i \leftarrow 1\) **to** \(\left\lceil \frac{n}{s} \right\rceil\) **do**
11. \(M[i] \leftarrow \text{median of } B_i\)
12. \(x \leftarrow \text{Select}\left(M[1: \left\lceil \frac{n}{s} \right\rceil], \left\lceil \frac{\left\lceil \frac{n}{s} \right\rceil}{2} \right\rceil, d + 1, s_{freq}, s_{rare}, b\right)\) \(\text{ \{median of medians\}}\)
13. \(t \leftarrow \text{Partition}(A[q:r], x)\) \(\text{ \{partition around } x \text{ which ends up at } A[t]\}}\)
14. **if** \(k = t - q + 1\) **then return** \(A[t]\)
15. **else if** \(k < t - q + 1\) **then return** \(\text{Select}(A[q:t-1], k, d+1, s_{freq}, s_{rare}, b)\)
16. **else return** \(\text{Select}(A[t+1:r], k, d+1, s_{freq}, s_{rare}, b)\)

Figure 3: Selection with two potentially different block sizes. Initial call to the function uses \(d = 1\).

---

(b) [ **25 Points** ] Repeat part (a) with \(s_{freq} = 3\) and \(s_{rare} = 5\).

(c) [ **15 Points** ] Suppose we replace line 11 of \textit{Select} with lines 11.1–11.4 as shown below. Then what will be its running time on an input of size \(n\) if we call it with \(s_{rare} = s_{freq} = 3\)?

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.1</td>
<td>divide (M[1: \left\lceil \frac{n}{s} \right\rceil]) into blocks (B_i)'s each containing (s) consecutive elements (last block may contain fewer than (s) elements)</td>
</tr>
<tr>
<td>11.2</td>
<td><strong>for</strong> (i \leftarrow 1) <strong>to</strong> (\left\lceil \frac{n}{s} \right\rceil/s) <strong>do</strong></td>
</tr>
<tr>
<td>11.3</td>
<td>(M'[i] \leftarrow \text{median of } B'_i)</td>
</tr>
<tr>
<td>11.4</td>
<td>(x \leftarrow \text{Select}\left(M'[1: \left\lceil \frac{n}{s} \right\rceil/s], \left\lceil \frac{\left\lceil \frac{n}{s} \right\rceil}{2s} \right\rceil, d + 1, s_{freq}, s_{rare}, b\right))</td>
</tr>
</tbody>
</table>
Figure 4: [Task 4] Sometimes rotating a photo clockwise by 90° can help you understand the real story behind the photo. For example, in the photo above you won’t realize that the guy in the black t-shirt is dangerously hanging from above until you rotate the photo clockwise. In the original photo they seem like relaxing on the sidewalk like everyone does. Photo credit: Steph Goralnick.

Task 4. [40 Points] Image Rotation

Sometimes rotating an image clockwise by 90° can be very useful as Figure 4 shows. This task asks you to do something simpler: rotate an $n \times n$ clockwise by 90°, where $n$ is a power of 2. However, you must design recursive divide-and-conquer algorithms for solving the problem and must not use more than $\Theta(1)$ extra space in any level of recursion.

Figure 5 shows a recursive divide-and-conquer approach for solving our problem. You first divide the image $Q$ into four quadrants $Q_1$, $Q_2$, $Q_3$ and $Q_4$. Then move each quadrant clockwise to occupy the position of the next quadrant in the sequence, that is, move $Q_1$ to the location of $Q_2$, $Q_2$ to the location of $Q_4$, $Q_4$ to the location of $Q_3$, and $Q_3$ to the location of $Q_1$. Finally, rotate each quadrant recursively. The result is a version of $Q$ rotated in the clockwise direction by 90°.

(a) [10 Points] Show that the algorithm described above based on Figure 5 runs in $\Theta(n^2 \log n)$ time.

(b) [30 Points] Extend the idea shown in Figure 5 to design a recursive divide-and-conquer algorithm that runs in $\Theta(n^2)$ time.
Figure 5: [Task 4] A recursive algorithm for rotating $n \times n$ images, where $n$ is a power of 2.