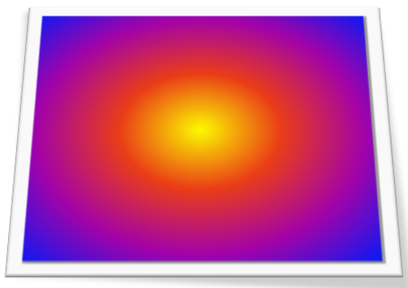
Homework #1

(Due: Oct 20)



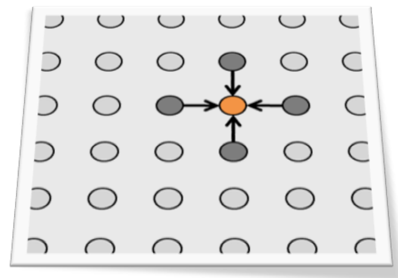


Figure 1: [Task 1] When the partial differential equation describing 2D heat diffusion is discretized in time and space, the approximate heat value at any grid cell (x, y) at time step t can be computed from the heat value at the four of its neighboring cells $(x, y-1)$, $(x, y+1)$, $(x-1, y)$ and $(x+1,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

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

$$\begin{aligned} h_{t+1}(x,y) \quad = \quad & h_t(x,y) \\ & + \frac{\alpha \Delta t}{\Delta x^2} \left(h_t(x-1,y) + h_t(x+1,y) - 2h_t(x,y) \right) \\ & + \frac{\alpha \Delta t}{\Delta y^2} \left(h_t(x,y-1) + h_t(x,y+1) - 2h_t(x,y) \right). \end{aligned}$$

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 polyonomials. 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 × N_y grid h with heat values at time
                                                        step 0, update h to heat values at time step T
                                                        using periodic boundary conditions.}

allocate new grid g[0..N_x − 1, 0..N_y − 1]

for t = 1 to T do
    for x = 0 to N_x − 1 do
        for y = 0 to N_y − 1 do
            g[x, y] ← h[x, y] + c_x (h[(x + 1) mod N_x, y] − 2h[x, y] + h[(x − 1) mod N_x, y])
                              + c_y (h[x, (y + 1) mod N_y] − 2h[x, y] + h[x, (y − 1) mod N_y])

    for x = 0 to N_x − 1 do
        for y = 0 to N_y − 1 do
            h[x, y] ← g[x, y]

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 $\langle height, weight\rangle$ 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_{rare} at levels that are divisible by 3 (levels start from 1), and s_{freq} at all other levels. Now the base case size b is also a parameter to the algorithm. Observe that when $b = 140$ and $s_{rare} = s_{freq} = 5$, the algorithm reduces to the one we saw in the class.

(*a*) [**10 Points**] Write a recurrence relation describing the running time of Select on an array of size n assuming $s_{rare} = s_{freq} = 3$. 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 of $A[q:r]$ such that $rank(x, A[q,r]) = k$.

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

recurrence? What is the smallest value of b you get?

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

(c) [**15 Points**] Suppose we replace line 11 of 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 $s_{rare} = s_{freq} = 3$?

11.1   divide $M[1 : \lceil \frac{n}{s} \rceil]$ into blocks B'_i's each containing s consecutive elements
        (last block may contain fewer than s elements)
11.2   ***for*** $i \leftarrow 1$ ***to*** $\lceil \lceil \frac{n}{s} \rceil / s \rceil$ ***do***
11.3 $M'[i] \leftarrow$ median of B'_i
11.4 $x \leftarrow \text{SELECT}\left(M'\left[1 : \lceil \lceil \frac{n}{s} \rceil / s \rceil\right],\ \left\lfloor \frac{\lceil \lceil \frac{n}{s} \rceil / s \rceil + 1}{2} \right\rfloor,\ d+1,\ s_{freq},\ s_{rare},\ b \right)$

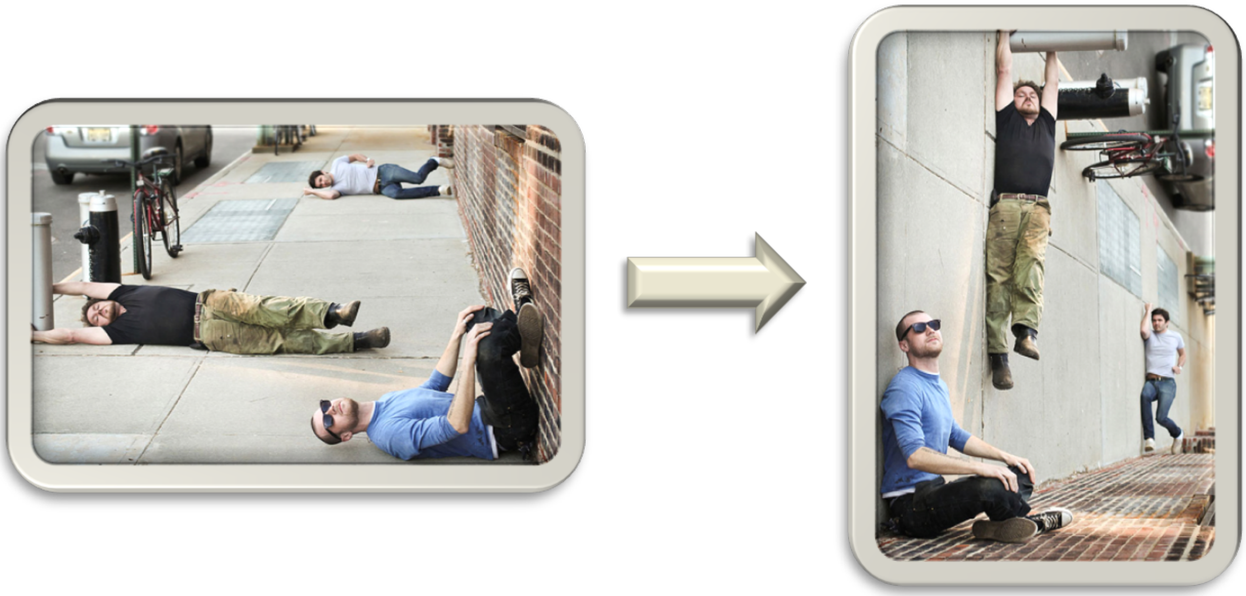


Figure 4: [Task 4] Sometimes rotating a photo clockwise by 90° can help you understand the real story behind the photo. For example, in the photo above you won't realize that the guy in the black t-shirt is dangerously hanging from above until you rotate the photo clockwise. In the original photo they 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 $\Theta\left(1\right)$ 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 $\Theta\left(n^2 \log n\right)$ time.

(b) [**30 Points**] Extend the idea shown in Figure 5 to design a recursive divide-and-coqnuer algorithm that runs in $\Theta\left(n^2\right)$ time.

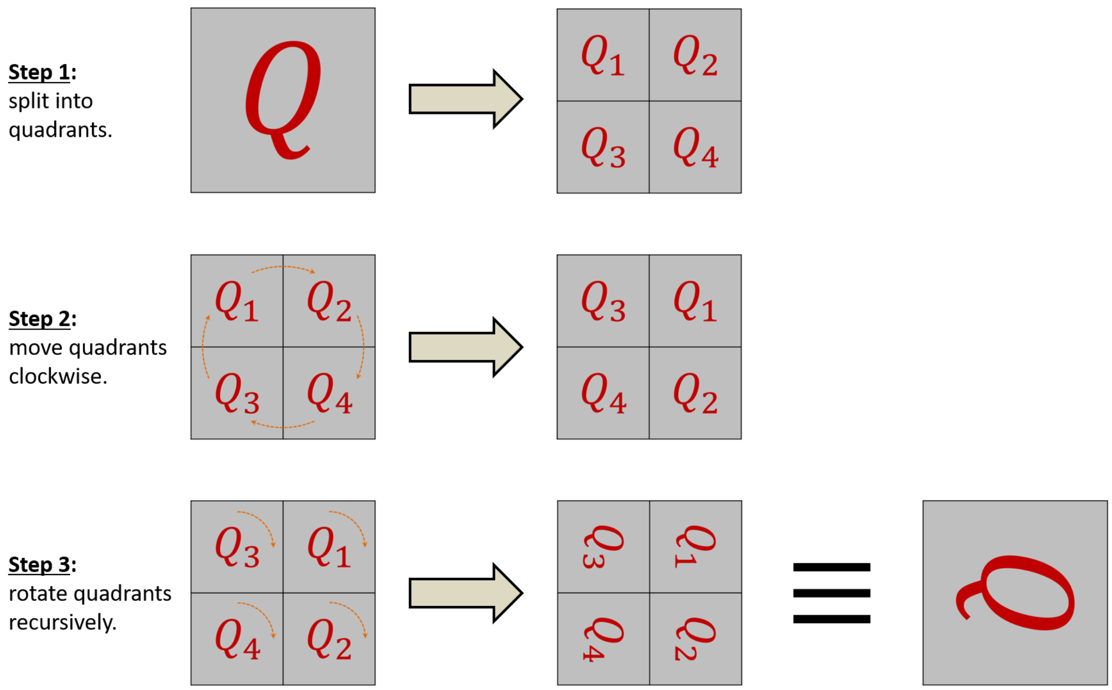


Figure 5: [Task 4] A recursive algorithm for rotating $n \times n$ images, where n is a power of 2.