Algorithm Design and Analysis (Spring 2023)

Assignment 1

Deadline: Mar 19, 2023

1. (20 Points) Prove the following generalization of the master theorem. Given constants $a \geq 1, b > 1, d \geq 0$, and $w \geq 0$, if $T(n) = 1$ for $n < b$ and $T(n) = aT(n/b) + n^d \log^w n$, we have

$$T(n) = \begin{cases} O(n^d \log^w n) & \text{if } a < b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \\ O(n^d \log^{w+1} n) & \text{if } a = b^d \end{cases}.$$

By the recurrence relation, we have

$$\begin{aligned} T(n) &= aT(n/b) + n^d \log^w n \\ &= a^2 T(n/b^2) + n^d \log^w n + a(n/b)^d \log^w (n/b) \\ &= a^3 T(n/b^3) + n^d \log^w n + a(n/b)^d \log^w (n/b) + a^2 (n/b^2)^d \log^w (n/b^2) \\ &\quad \vdots \\ &= n^d \log^w n + a(n/b)^d \log^w (n/b) + a^2 (n/b^2)^d \log^w (n/b^2) + \\ &\quad \cdots + a^{\log_b n} (n/b^{\log_b n})^d \log^w (n/b^{\log_b n}) \\ &< n^d \log^w n (1 + (a/b^d) + (a/b^d)^2 + \cdots + (a/b^d)^{\log_b n}) \end{aligned}$$

If $a < b^d$, then we have $a/b^d < 1$. Thus,

$$T(n) < n^d \log^w n \frac{1 - (a/b^d)^{\log_b n}}{1 - a/b^d} = O(n^d \log^w n).$$

If $a > b^d$, choose $\varepsilon > 0$ such that $\log_b a > d + \varepsilon$. Since $n^d \log^w n = O(n^{d+\varepsilon})$, applying the original master theorem on the following recurrence will yield $T(n) = O(n^{\log_b a})$:

$$U(n) = aU(n/b) + O(n^{d+\varepsilon}).$$

If $a = b^d$, we have $e/b^d = 1$. Therefore,

$$T(n) < n^d \log^w n \cdot \log_b n = O(n^d \log^{w+1} n).$$

2. (20 points) Recall the median-of-the-medians algorithm we learned in the lecture. It groups the numbers by 5. What happens if we group them by $3, 7, 9 \ldots$? Please analyze those different choices and discuss which one is the best. Note that in this problem, we may drop the big-O notation and discuss the constants.

Let k be the number of elements in a group, where k is an odd constant. Let $T(n)$ be the time complexity for running the median-of-the-medians algorithm on n elements. For the median-of-the-medians algorithm, we need to find the median for each group, which takes constant time. Thus, finding the medians over all groups takes $O(n)$ time. Next, we need to find the median for those n/k medians, which takes $T(n/k)$ time. After finding the median a of those medians, at least $\frac{n}{2k} \cdot \frac{k+1}{2}$ elements can be removed from our consideration. We need to find the median over the remaining $\frac{3k-1}{4k}n$ elements, which takes time $T(\frac{3k-1}{4k}n)$. Therefore, the time complexity can be written by the following recurrence relation:

$$T(n) \leq T\left(\frac{n}{k}\right) + T\left(\frac{3k-1}{4k}n\right) + O(n).$$

Let c be a constant such that

$$T(n) \leq T\left(\frac{n}{k}\right) + T\left(\frac{3k-1}{4k}n\right) + cn.$$

For $k \geq 5$ (and k is an odd number), we have $T(n) = O(n)$ which can be proved by induction.

For the base step, $T(n) \leq Bn$ for some constant B and sufficiently small n. For the inductive step, suppose $T(n) \leq Bn$ holds for $n = 1, \ldots, i$. For $n = i + 1$, we have

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{k}\right) + T\left(\frac{3k-1}{4k}n\right) + cn \\ &\leq B \cdot \frac{n}{k} + B \cdot \frac{3k-1}{4k}n + cn && \text{(induction hypothesis)} \\ &= Bn - \left(B \cdot \frac{k-3}{4k} - c\right) n, \end{aligned}$$

which implies $T(n) \leq Bn$ if we choose B such that $B \cdot \frac{k-3}{4k} - c \geq 0$. Therefore, choosing any constant odd $k \geq 5$ yields the asymptotic time complexity $T(n) = O(n)$.

However, the induction above fails for $k = 3$, as $B \cdot \frac{k-3}{4k} - c \geq 0$ no longer holds.

For $k = 3$, we have

$$T(n) \leq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n).$$

We will prove $T(n) = O(n \log n)$ by induction.

For the base step, $T(n) \leq Bn \log n$ for some constant B and sufficiently small $n \geq 2$. For the inductive step, suppose $T(n) \leq Bn$ holds for $n = 2, \ldots, i$. For $n = i + 1$, we have

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn \\ &\leq B \cdot \frac{n}{3} \log \frac{n}{3} + B \cdot \frac{2}{3} n \log \frac{2n}{3} + cn && \text{(induction hypothesis)} \\ &= Bn \log n - B \cdot \frac{n}{3} \log 3 - B \cdot \frac{2n}{3} \log \frac{3}{2} + cn \\ &= Bn \log n - n \cdot \left(\frac{B}{3} \log 3 + \frac{2B}{3} \log \frac{3}{2} - c \right), \end{aligned}$$

which can imply $T(n) \leq Bn \log n$ if we choose B to be sufficiently large.

3. (20 points) Given n integers, where n is even. Can you find both the maximum and minimum within $3n/2 - 2$ comparisons? Note that we also do not use big-O notations here.

   The algorithm is described as follows.

   1. Let $a_1, \ldots, a_{n/2}$ be the first $n/2$ integers, and let $b_1, \ldots, b_{n/2}$ be the last $n/2$ integers. For each $i = 1, \ldots, n/2$, compare a_i with b_i. Let ℓ_i be the larger one and s_i be the smaller one. Break tie arbitrarily.
   2. Find the maximum among $\ell_1, \ldots, \ell_{n/2}$ using the naïve algorithm, and output this maximum as the global maximum.
   3. Find the minimum among $s_1, \ldots, s_{n/2}$ using the naïve algorithm, and output this minimum as the global minimum.

   For the correctness of the algorithm, we will only show that the algorithm outputs the maximum number, as the analysis for the minimum number is similar. It suffices to show that there exists a maximum number among $\ell_1, \ldots, \ell_{n/2}$. If $s_1, \ldots, s_{n/2}$ do not contain a maximum, we are done. Otherwise, if certain s_i is the maximum, then we must have ℓ_i is also a maximum with $\ell_i = s_i$. It is still true that there exists a maximum among $\ell_1, \ldots, \ell_{n/2}$.

   The algorithm uses a total of $\frac{3n}{2} - 2$ comparisons: $n/2$ comparisons for Step 1, $n/2 - 1$ comparisons for Step 2, and $n/2 - 1$ comparisons for Step 3.

4. (20 points) Recall the Word RAM model we learn in the lecture. Let w be the number of bits in a machine word. We know that those basic operations on each word (like addition, right shift, and/or, ...) can be viewed as unit operations. We will design algorithms to compute the transpose A^T of a binary matrix $A \in \{0,1\}^{h \times h}$.

   (a) (5 points) If we want to store the matrix in one word, how large h can be? How to store it in one word?

   Since $h^2 \leq w$, h can be at most $\lfloor \sqrt{w} \rfloor$.

   (b) (15 points) Consider an one-word size matrix A. Design an $O(\log w)$ algorithm to compute its transpose A^T.

   The algorithm is described as follows.

   1. Divide matrix A to 4 blocks, each is $\frac{h}{2} \times \frac{h}{2}$.

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}.$$

   2. Recursively transpose each block (simultaneously), we get

$$\begin{bmatrix} A_{11}^T & A_{12}^T \\ A_{21}^T & A_{22}^T \end{bmatrix}.$$

   3. Swap A_{12}^T and A_{21}^T, we get

$$\begin{bmatrix} A_{11}^T & A_{21}^T \\ A_{12}^T & A_{22}^T \end{bmatrix},$$

      which is exactly A^T.

   By blocking we reduce the problem from w to $w/4$,

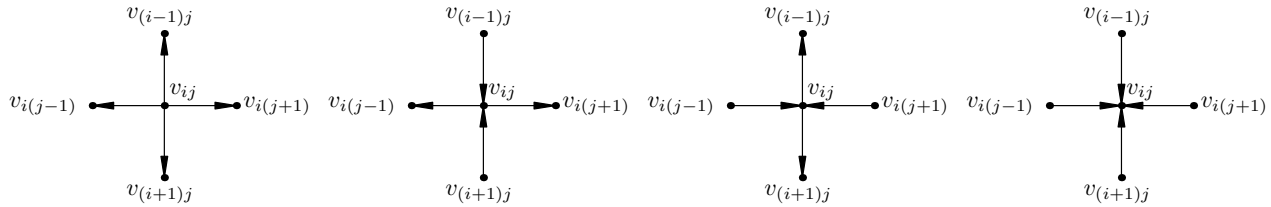
$$T(w) = T\left(\frac{w}{4}\right) + O(1).$$

   By Master Theorem, $T(w) = O(\log w)$.

5. (20 points) Let $G = (V, E)$ be a $n \times n$ *grid* which is an undirected graph with n^2 vertices labeled $V = \{v_{ij}\}_{i=0,1,\ldots,n-1;j=0,1,\ldots,n-1}$. Vertex v_{ij} is adjacent to $v_{(i-1)j}$ (unless $i = 0$), $v_{(i+1)j}$ (unless $i = n-1$), $v_{i(j-1)}$ (unless $j = 0$), and $v_{i(j+1)}$ (unless $j = n-1$).

   Let H be a directed graph sharing the same vertex set V as G. The edge set of H is the same as the edge set of G, except that a direction of each edge in G is specified for the corresponding edge in H.

   (a) (10 points) Suppose the directions of the edges in H satisfy the followings. The directions of the edges between v_{ij} and the "left" and the "right" neighbors of v_{ij} are the same, i.e., either both edges point towards v_{ij} or both edges point outwards v_{ij}, and the directions of the edges between v_{ij} and the "upper" and the "lower" neighbors of v_{ij} are the same. Specifically, the four edges incident to v_{ij} (where $0 < i < n-1$ and $0 < j < n-1$) must be in one of the following four configurations.



   Design an algorithm that finds *one vertex with out-degree* 0 or *one directed cycle*. Your algorithm must run in $O(\log n)$ time.

   It is straightforward to check that, for the four vertices v_{ij}, $v_{(i+1)j}$, $v_{(i+1)(j+1)}$, $v_{i(j+1)}$, either that one of them has out-degree 0 or that they form a cycle. It takes $O(1)$ time to check the four vertices.

   (b) (10 points) Suppose the directions of the edges in H are defined as follows. There is a function (given as input) $f : V \to \mathbb{Z}$ that assigns an integer *value* to each vertex, and assume $f(u) \neq f(v)$ for any $u \neq v \in V$. The direction of the edge between the two vertices u, v is from the vertex with a higher value to the vertex with a lower value.

   Design an algorithm that finds *one vertex with out-degree* 0 or *one directed cycle*. Your algorithm must run in $O(n)$ time.

   First of all, it is obvious that H does not contain directed cycles. It remains to find a vertex with out-degree 0.

   We divide the graph into 4 quadrants using the middle row $(v_{1,\lceil n/2\rceil}, v_{2,\lceil n/2\rceil}, \ldots, v_{n,\lceil n/2\rceil})$ and the middle column $(v_{\lceil n/2\rceil,1}, v_{\lceil n/2\rceil,2}, \ldots, v_{\lceil n/2\rceil,n})$. We find the vertex v on the boundary rows, boundary columns, the middle row and the middle column with minimum $f(v)$. If the vertex is the center vertex $v_{\lceil n/2\rceil,\lceil n/2\rceil}$, we have found a vertex with out-degree 0, and we are done. Otherwise, suppose v is on the center row, i.e., $v_{\lceil n/2\rceil,j}$. If v is a "local minimum", we are done. Otherwise, one of its neighbors

that is not on the center row, for example, $v_{(\lceil n/2 \rceil+1),j}$, must have a smaller value of f than $v_{\lceil n/2 \rceil,j}$. Then "water" flowing from $v_{\lceil n/2 \rceil,j}$ to the quadrant containing $v_{(\lceil n/2 \rceil+1),j}$ cannot leave the quadrant since $v_{\lceil n/2 \rceil,j}$ is the lowest on the quadrant boundary. Therefore, we can recurse on the quadrant. If v is on the boundary but not any of the center row or center column, we know that it is either a local minimum or the unique neighbor of v in the interior of H must have a smaller f value. In this case, we can recurse on the quadrant containing this unique neighbor. Note that recursive call must include the part of the middle row, the middle column, and the original boundary incident to the quadrant.

The correctness of the algorithm follows from the discussion above. The time complexity is given by the recurrence relation

$$T(n) = T(n/2) + O(n),$$

which yields $T(n) = O(n)$ by master theorem.

(c) (Bonus 5 points) If no further assumptions were made for directions of the edges in H, prove that any algorithm that finds *one vertex with out-degree* 0 or *one directed cycle* requires $\Omega(n^2)$ time.

It is possible to construct H such that H contains a single directed cycle with length n^2 (and no vertex with out-degree 0 in particular). For this H as input, it takes time $\Theta(n^2)$ to even output the solution.