Algorithm Design and Analysis (Fall 2023)
Assignment 1
Deadline: Nov 1, 2023

1. (25 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

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

Proof: The running time of solving all size < b problem is: $a^{log_b n}O(1) = O(n^{log_b a})$.
The total running time for combining is: $n^d log^\omega n + a(\frac{n}{b})^d log^\omega(\frac{n}{b}) + ... + a^{log_b n}(\frac{n}{b^{log_b n}})^d log^\omega(\frac{n}{b^{log_b n}})$.
Simplification: $n^d(log^\omega n + \frac{a}{b^d} log^\omega(\frac{n}{b}) + ... + (\frac{a}{b^d})^{log_b n} log^\omega \frac{n}{b^{log_b n}})$. Since $n \geq b^k$ when $k \leq log_b n$, so the expression is less than $n^d log^\omega n(1 + (\frac{a}{b^d}) + ... + (\frac{a}{b^d})^{log_b n})$.
Case 1: $a < b^d$
$\frac{a}{b^d} < 1$, so $T(n) < n^d log^\omega n \frac{1(1-(\frac{a}{b^d})^{log_b n})}{1-\frac{a}{b^d}} = (\frac{1-(\frac{a}{b^d})^{log_b n}}{1-\frac{a}{b^d}}) n^d log^\omega n = O(n^d log^\omega n)$.
Case 2: $a > b^d$
$\frac{a}{b^d} > 1$, so the last term dominates the sum. $T(n) = O(n^d log^\omega n (\frac{a}{b^d})^{log_b n}) = O(log^\omega n \cdot a^{log_b n}) = O(log^\omega n \cdot n^{log_b a}) = O(n^{log_b a})$.
Case 3: $a = b^d$
$\frac{a}{b^d} = 1$, so $T(n) = O(n^d log^\omega n \cdot log_b n) = O(n^d log^{\omega+1} n)$.
Therefore, we have

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

2. (25 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.

The problem is to find the k-th smallest integer in a set S of n integers $x_1, x_2, ..., x_n$. First, let's group them by 3. Partition S into subsets with size 3 and finding the medians of them: $v_1, v_2, ...$ take O(n). Fixing v to be the median of $v_1, v_2, ...$ takes $T(\frac{n}{3})$. Now there are $\frac{n}{3}$ groups, so $\frac{n}{3}$ medians. v is no greater than $\frac{n}{6}$ medians, no less than $\frac{n}{6}$ medians. Each median is no greater than 2 integers, no less than 2 integers. So v is no greater than $\frac{n}{3}$, no less than $\frac{n}{3}$ integers. $T(|L|) \leq T(n - \frac{n}{3}), T(|R|) \leq T(n - \frac{n}{3})$, thus $T(n) = T(\frac{n}{3}) + T(\frac{4n}{6}) + O(n)$. Similarly we have $T(n) = T(\frac{n}{5}) + T(\frac{7n}{10}) + O(n)$ when we group them by 5. Now let's group them by k, $k = 2N + 1, N$ is an integer: Partition and fixing v to be the median of $\frac{n}{k}$ medians takes $O(n) + T(\frac{n}{k})$. v is no greater and no less than $\frac{n}{2k}$ medians. Each median is no greater and no less than $\frac{k+1}{2}$ integers. So v is no greater and no less than $\frac{(k+1)n}{4k}$ integers. So $T(n) = T(\frac{n}{k}) + T(\frac{(3k-1)n}{4k}) + O(n)$ when we group them by k.

For grouping them by 3: Suppose $T(n) \leq Bn$.
Basic step: $T(1) = 1$.
Inductive step: Suppose $T(i) \leq Bi$ for each $i = 1, ..., n-1$.
$T(n) = T(\frac{n}{3}) + T(\frac{4n}{6}) + Cn \leq \frac{1}{3}Bn + \frac{4}{6}Bn + Cn = Bn + Cn$. Obviously, it's greater than Cn. So the hypothesis is false. So the time complexity is greater than when we group them by 5.
Now let's discuss when we group them by $k, k \geq 5$.
Suppose $T(n) \leq Bn$.
Basic step: $T(1) = 1$.
Inductive step: Suppose $T(i) \leq Bi$ for each $i = 1, ..., n-1$.
$T(n) = T(\frac{n}{k}) + T(\frac{(3k-1)n}{4k}) + Cn \leq \frac{1}{k}Bn + \frac{3k-1}{4k}Bn + Cn = \frac{3k+3}{4k}Bn + Cn = (\frac{3}{4} + \frac{3}{4k})Bn + Cn$.
We notice that when $k \geq 5$, we always have $T(n) = O(n)$. However, we notice that as k increases, C also increases, which means that it takes more time on finding the medians of $\frac{n}{k}$ groups and sorting them because k is getting larger and larger. So as k increases, the time complexity also increases when $k > 5$. Therefore, grouping them by 5 is overall best.

3. (25 points) Let X and Y be two sets of integers. Write $X \succ Y$ if $x \geq y$ for all $x \in X$ and $y \in Y$. Given a set of m integers, design an $O(m \log(m/n))$ time algorithm that partition these m integers to k groups $X_1, \ldots, X_k$ such that $X_i \succ X_j$ for any $i > j$ and $|X_1|, \ldots, |X_k| \leq n$. Notice that k is not specified as an input; you can decide the number of the groups in the partition, as long as the partition satisfies the given conditions. You need to show that your algorithm runs in $O(m \log(m/n))$ time.

   Remark: We have not formally define the asymptotic notation for multi-variable functions in the class. For f and g be functions that maps $\mathbb{R}^k_{>0}$ to $\mathbb{R}_{>0}$, we say $f(\mathbf{x}) = O(g(\mathbf{x}))$ if there exist constants $M, C > 0$ such that $f(\mathbf{x}) \leq C \cdot g(\mathbf{x})$ for all $\mathbf{x}$ with $x_i \geq M$ for some i. The most rigorously running time should be written as $O(m \cdot \max\{\log(m/n), 1\})$, although it is commonly just written as $O(m \log(m/n))$ for this kind of scenarios.

   Algorithm: First, select a pivot from the set of m integers by median of medians. Then, compare each integers with the pivot to partition the remaining integers into two groups, denoted as A and B. A's elements are all greater or equal to the pivot and B's elements are all less than the pivot. Next, if the size of A is greater than n, recursively apply the algorithm to the group A until each small group's size is smaller than n. Then apply the same method to the group B until each small group's size is smaller than n. Finally, we can get a group of sorted groups as the question request.

   Proof: show that the algorithm runs in $O(mlog(m/n))$ time.
   Because we apply the median of medians algorithm, suppose each recursion reduces the size of the problem by at least $1/k$, k is a constant that is greater than 1. Then the depth of the recursion is at most $log(m/n)$. Below show the proof:
   Suppose the depth is d, than we have $m(1/k)^d < n \rightarrow d < log_k(m/n) = log(m/n)$.
   Each level we have at most m integers to apply the median of medians algorithm, which takes $O(m)$ time, so the overall time complexity is $O(mlog(m/n))$.

4. (25 points) Given an array $A[1, \ldots, n]$ of integers sorted in ascending order, design an algorithm to **decide** if there exists an index i such that $A[i] = i$ for each of the following scenarios. Your algorithm only needs to decide the existence of i; you do not need to find it if it exists.

   (a) The n integers are positive and distinct.

   (b) The n integers are distinct.

   (c) The n integers are positive.

   (d) The n integers are positive and are less than or equal to n.

   (e) No further information is known for the n integers.

   Prove the correctness of your algorithms. For each part, try to design the algorithm with running time as low as possible.

(a) If n integers are positive and distinct, we just need to check whether $A[1]$ is equal to 1. If so, then there exists an index i. If not, there is no index i.
Proof: The first part is clear: if $A[1] = 1$, then $i = 1$. For the last part: Suppose that $A[1] > 1$ and there exists an index i that satisfies $A[i] = i$. There are $i - 2$ spaces between $A[1]$ and $A[i]$, like $A[2], A[3], ..., A[i-1]$. Because A is sorted in ascending order and the n integers are positive and distinct, there are less than $i-2$ distinct integers to be fit in these spaces, which makes it impossible to place i in $A[i]$. Therefore, the last part is also correct.
(b)Use binary search. If $A[mid] = mid$, then there exists an index i. Else if $A[mid] < mid$, binary search $A[mid+1]$ to $A[r]$. Else, binary search $A[l]$ to $A[mid-1]$.
Proof: The n integers are distinct and are in ascending order. If $A[mid] < mid$, then i can't be in $[l, mid-1]$. The reason is the same as (a). Similarly, if $A[mid] > mid$, then i can't be in $[mid+1, r]$ with the same reason as (a).
(c)The n integers are positive. First, binary search A, if $A[mid] = mid$, then there exists an index $i = mid$, else if $A[mid] < mid$, let $A[mid] = mid - k$, than search between $A[l]$ to $A[mid-k](mid-k \geq l)$ and between $A[mid+1]$ to $A[r]$ with the same method. Else if $A[mid] > mid$, let $A[mid] = mid + k$, than search between $A[l]$ to $A[mid-1]$ and between $A[mid+k]$ to $A[r](mid+k \leq r)$.
Proof: We just need to show that if $A[mid] = mid + k$, then the index i can't be in $[mid, mid+k-1]$. The n integers are positive and some integers can be the same and in ascending order, if $A[mid] = mid + k$, then $A[mid+1], A[mid+2], ..., A[mid+k-1] \geq mid + k$ because they are in ascending order. So the index i can't be in $[mid+1, mid+k-1]$. Similarly if $A[mid] = mid - k$, then the index i can't be in $[mid-k+1, mid]$. Therefore, if there exists an index i, we can always find it by applying

the above-mentioned algorithm.

(d)The n integers are positive and are less than or equal to n. There always exists an index i.

Proof: Suppose there is no such index i. Let $A[1] = k, 1 \le k \le n$, then $A[k] = k+n_1, n_1 > 0$. So $A[k+n_1] = k+n_1+n_2, n_2 > 0$...Finally, we have $A[n] = n+n_k, n_k > 0$, which contradicts to "n integers are less than or equal to n".

(e)The method is similar to (c). Additionally, if $A[mid] \le 0$, then we only need to search between $A[mid+1]$ to $A[r]$.

Proof: The proof is also similar to (c), if $A[mid] \le 0$, then $A[k] \le 0, k < mid$, which means $A[k] < k$.

5. How long does it take you to finish the assignment (including thinking and discussion)? Give a score (1,2,3,4,5) to the difficulty. Do you have any collaborators? Please write down their names here.

   About 6-7 hours(though it was divided into several days).

   1. 3
   2. 3
   3. 4
   4. 4

   No collaboration.