21120491 Advanced Data Structure and Algorithm Analysis **Spring-Summer 2022**

Lecture 7: Divide-and-Conquer & Master Theorem

Lecturer: Deshi Ye *Scribe: D. Ye*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

7.1 Overview

When we face a new problem, what shall we do?

- Can you avoid solving the problem from scratch? Is it a disguised version, variant, or special case of a problem that you already know how to solve?
- Can you find some known problem similar to the new problem?
- If you must design a new algorithm from scratch, what tools shall we use?
- Can you make your algorithm simpler or faster?

To analyze the running time, we usually focus on **asymptotic** case. In asymptotic analysis, we estimate the running time of the algorithm when it is run on sufficiently large input. In the asymptotic case, we suppress constants and lower-order terms.

Random Access Machine (RAM):

- Has a controller with registers and a memory
- Each register and memory location holds an integer
- Do arithmetic on registers, compare registers, and load or store a value from a memory location addressed by a particular register all as a single step (*i.e.*, 1 time step).
- Loops and subroutines are not considered simple operations. Instead, they are the composition of many single-step operations.

The running time of an algorithm (time complexity) is based on RAM model.

In this lecture, we first present the divide-and-conquer method that will speed up the running time of a problem. Next, a general master theorem will be provided to achieve an asymptotic running time analyis.

The crucial steps in divide and conquer method is

- **Divide:** Divide instance into many smaller instances
- **Conquer:** Solve each of smaller instances recursively and separately

- **Combine:** Combine solutions to small instances to obtain a solution for the original big instance

Divide-and-Conquer usually aims to give a more efficient algorithm, *i.e.*, to provide an algorithm with less running time.

7.2 Closest Points Problem

Problem statement: Given n points $P = \{p_1 = (x_1, y_2), p_2 = (x_2, y_2), \ldots, p_n = (x_n, y_n)\}$ in a plane. Find the closest pair of points p_i and p_j with the smallest Euclidean distance $d(p_i, p_j)$. (If two points have the same position, then that pair is the closest with a distance 0.)

To measure the distance between two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ in the plane, we use the Euclidean distance $d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Clearly, we can use the brute force method to check each pair, and it costs $O(n^2)$ time. Divide-and-conquer method can speed it up to $O(n \log n)$.

Let P_x be the set of points in P sorted by x-coordinate, and P_y be the set of points in P sorted by y-coordinate.

We denote Closestpoint(P_x, P_y) to be the function to get the closest pair of points in P.

Divide: W.L.O.G, we assume $P_x = \{p_1 = (x_1, y_2), p_2 = (x_2, y_2), \ldots, p_n = (x_n, y_n)\}$. Let $m = \lfloor n/2 \rfloor$.

We divide the instance P_x into two subproblems according to the x-coordinate. Let L_x be the first half part of P_x, and R_x be the second half part of P_x. Let $L_x = \{p_1 = (x_1, y_2), p_2 = (x_2, y_2), \ldots, p_m = (x_m, y_m)\}$, and $R_x = \{p_{m+1}, \ldots, p_n\}$. Let L_y be the first half of P_x, sorted by y-coordinate, which can be obtained by scanning through P_y and removing the points with x-coordinate larger than x_m. Let R_y be the second half of P_x, sorted by y-coordinate, which can be obtained by scanning through P_y and keeping the points with x-coordinate larger than x_m. It costs $O(n)$ time to construct L_y and R_y.

Conquer: Solve the subproblems Closestpoint(L_x, L_y), and Closestpoint(R_x, R_y), separately.

Combine: Let $\delta = \min(\text{Closestpoint}(L_x,\ L_y),\ \text{Closestpoint}(R_x,\ R_y))$.

Form a list Q of the points (sorted by increasing y) that are within δ of the x coordinate of p_m (i.e., x_m is the largest x-coordinate in left half). $Q = \{q_1, \ldots, q_l\}$ with x-coordinate between $x_m - \delta$ and $x_m + \delta$ and sorted by increasing y-coordinate. The set Q can be computed in linear time $O(n)$ by scanning through P_y (i.e., L_y and R_y) and removing any points with an x-coordinate outside the range of interest.

Now, we only need to compute the minimum distance among points in Q, in which we can show that it costs $O(n)$ time according to Lemma 7.1. In sum, it costs $O(n)$ to combine.

Lemma 7.1 *For any i, the distance of $d(q_i, q_{i+8}) \geq \delta$. In other words, to find any point below q_i within the distance δ, we only need to consider at most 7 points below q_i.*

Proof: In Fig. 7.1, there is a four (across) by two (down) grid of squares of siz $\delta/2$. The top of the grid goes through q_i.

Note that the farthest apart that two points in a box can be is at opposite corners of the box, which is $\delta/\sqrt{2} < \delta$. Recall that δ is the smallest distance between a left pair or a right pair. Thus, each square box contains at most one point of Q. On the other hand, any point below the eight boxes to the point q_i has a distance larger than δ. The Lemma follows immediately. ■

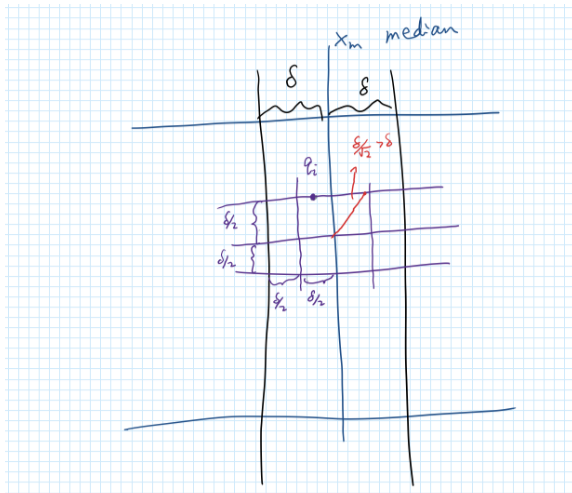

Figure 7.1: There is at most one point in each box

Running time The initiation phase, sorts by x-coordinate and also by y-coordinate, it costs $O(n \log n)$. Let $T(n)$ be the running time of Closestpoint(P_x, P_y). Then, we have

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

which solves to $O(n \log n)$.

7.3 Master Theorem

We would like to get asymptotic running time of the general form

$$T(n) = aT(n/b) + f(n).$$

There are several forms of master theorems, see for example in [1, 2].

Theorem 7.2 (Master's Theorem) *The recurrence relations of the form $T(n) = aT(n/b) + f(n)$, where a and b are constants, $a \geq 1$ and $b > 1$.*

- *Case 1: $f(n) = O(n^{log_b a - \epsilon})$ for some constant $\epsilon > 0$, which says that f grows more slowly than the number of leaves. In this case, the total work is dominated by the work done at the leaves, so $T(n) = \Theta(n^{\log_b a})$.*
- *Case 2: $f(n) = \Theta(n^{log_b a})$, which says that f grows at the same rate as the number of leaves. In this case, $T(n) = \Theta(n^{\log_b a} \log n)$.*

- *Case 3: $f(n) = O(n^{log_b a+\epsilon})$ for some constant $\epsilon > 0$, which says that f grows faster than the number of leaves. For the upper bound, we also need an extra smoothness condition on f, namely $af(n/b) \le cf(n)$ for some constant $c < 1$ and large n. In this case, the total work is dominated by the work done at the root, so $T(n) = \Theta(f(n))$.*

Proof: The solution of the recurrence is

$$T(n) = aT(n/b) + f(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i) + O(n^{\log_b a}) \tag{7.1}$$

The Equation (7.1) is drawing by the tree generated by the recurrence, which is illustrated in Fig. 7.2.

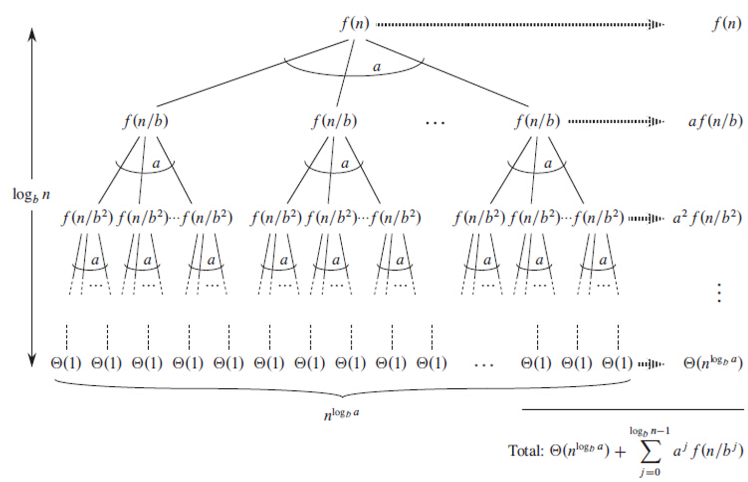

Figure 7.2: The recursive tree for $T(n) = aT(n/b) + f(n)$

Now, we prove this theorem case by case.

Case 1:

$$T(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i) + O(n^{\log_b a}) \le \sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a-\epsilon} + O(n^{\log_b a})$$

and

$$\begin{aligned}
\sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a-\epsilon} &= n^{\log_b a-\epsilon} \sum_{i=0}^{\log_b n} a^i b^{-i\log_b a} b^{i\epsilon} \\
&= n^{\log_b a-\epsilon} \sum_{i=0}^{\log_b n} a^i a^{-i} b^{i\epsilon} \\
&= n^{\log_b a-\epsilon} \frac{b^{\epsilon(\log_b n+1)-1}}{b^\epsilon - 1} \\
&= n^{\log_b a-\epsilon} \frac{n^\epsilon b^\epsilon - 1}{b^\epsilon - 1} \\
&\leq n^{\log_b a} \frac{b^\epsilon}{b^\epsilon - 1} \\
&= O(n^{\log_b a})
\end{aligned}$$

In all, we have $T(n) = O(n^{\log_b a})$.

Case 2.

$$\begin{aligned}
\sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a} = n^{\log_b a} \sum_{i=0}^{\log_b n} a^i b^{-i\log_b a} &= n^{\log_b a} \sum_{i=0}^{\log_b n} a^i a^{-i} \\
&= n^{\log_b a}(\log_b n + 1) = \Theta(n^{\log_b a} \log_b n).
\end{aligned}$$

Easy to check that $T(n) = \Theta(n^{\log_b a} \log_b n)$

Case 3.

$$af(n/b) = a(n/b)^{\log_b a+\epsilon} = an^{\log_b a+\epsilon} b^{-\log_b a} b^{-\epsilon} = f(n)b^{-\epsilon}.$$

Let $c = b^{-\epsilon} < 1$. In this case, we have $a^i f(n/b^i) \leq c^i f(n)$ (one can prove it by induction). Thus,

$$\begin{aligned}
T(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i) + O(n^{\log_b a}) &\leq \sum_{i=0}^{\log_b n} c^i f(n) + O(n^{\log_b a}) \\
\leq f(n) \sum_{i=0}^{\infty} c^i + O(n^{\log_b a}) = f(n) \frac{1}{1-c} &+ +O(n^{\log_b a}) = O(f(n)).
\end{aligned}$$

■

Theorem 7.3 (Master's Theorem) *The recurrence relations of the form $T(n) = aT(n/b) + f(n)$, where a and b are constants, $a \geq 1$ and $b > 1$.*

- *Case 1: If $af(n/b) = Kf(n)$ for some constant $K > 1$, then $T(n) = \Theta(n^{\log_b a})$.*
- *Case 2: If $af(n/b) = f(n)$, then $T(n) = \Theta(n^{\log_b a} \log n)$*
- *Case 3: If $af(n/b) = \tau f(n)$ for some constant $\tau < 1$, then $T(n) = \Theta(f(n))$*

Theorem 7.4 (Master's Theorem) *The recurrence relations of the form $T(n) = aT(n/b) + O(n^k \log^p n)$, where a and b are constants, $a \geq 1$ and $b > 1$, $p \geq 0$.*

- *Case 1: If $a > b^k$ for some constant $K > 1$, then $T(n) = \Theta(n^{\log_b a})$.*
- *Case 2: If $a = b^k$, then $T(n) = \Theta(n^k \log^{p+1} n)$*
- *Case 3: If $a < b^k$, then $T(n) = \Theta(n^k \log^p n)$*

7.4 Practice Problems

For each of the following recurrences, solve the running time of $T(n)$ with Master Theorems or indicate that it does not apply.

P1:

$$T(n) = 3T(n/2) + n^2$$

Sol. Here, $a = 3, b = 2$. Check that $f(n) = n^2 = \Omega(n^{\log_2 3 + (2 - \log_2 3)})$, then $\epsilon = 2 - \log_2 3$, and $3(n/2)^2 = 3n^2/4$, we can check that the Case 3 of Theorem 7.2 valids. Thus $T(n) = \Theta(n)^2$.

P2:

$$T(n) = 3T(n/3) + \sqrt{n}$$

Sol. Here, $a = b = 3$. Note that $f(n) = n^{1/2} = O(n^{\log_b a - \epsilon}) = O(n^{1-\epsilon})$, for any $\epsilon < 1/2$. Then Case 1 of Theorem 7.2 valids. Thus, $T(n) = \Theta(n^{\log_b a}) = \Theta(n)$.

P3:

$$T(n) = 4T(n/2) + n/\log n$$

Sol. Note that $a = 4, b = 2$. Easy to check that $f(n) = n/\log n = O(n^{\log_b a - \epsilon}) = O(n^{2-\epsilon})$, if $\epsilon < 1$. Then Case 1 of Theorem 7.2 valids. Thus, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$.

P4:

$$T(n) = 2T(n/2) + n/\log n$$

Sol. Here, $a = 2, b = 2$. One cannot find any constant $\epsilon > 0$ such that $f(n) = n/\log n = O(n^{\log_b a - \epsilon}) = O(n^{1-\epsilon})$. Thus, Master Theorem 7.2 does not apply.

We use recursive three method to solve it. Each level i has $n/(\log_2 n/2^i)$, and there are $\log_2 n$ levles.

$$T(n) = \sum_{i=0}^{\log_2 n} \frac{n}{\log_2 n/2^i} = \sum_{i=0}^{\log_2 n} \frac{n}{\log_2 n - i} = \sum_{j=1}^{\log_2 n} \frac{n}{j} = O(n \log \log n).$$

P5: Given an integer $M > 1$.

$$T(n) = \begin{cases} 8T(n/2) + 1 & \text{if } n^2 > M \\ M & \text{otherwise} \end{cases} \tag{7.2}$$

Sol. The master theorem does not apply. We use the recursive tree method to solve it. If $n/\sqrt{M} > 1$, each level is 1. When $n/\sqrt{M} = 1$, we have leaves with each of size M. There are total $\log_2 \frac{n}{\sqrt{M}}$ levels, and a total of $8^{\log_2 \frac{n}{\sqrt{M}}}$ leaves. The total running time is $O(M8^{\log_2 \frac{n}{\sqrt{M}}} + \log_2 n/\sqrt{M}) = O(n^3/\sqrt{M})$.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

[2] M. A. Weiss. *Data Structures and Algorithm Analysis in C (2nd Ed.).* Addison-Wesley Longman Publishing Co., Inc., USA, 1996.