Divide & Conquer

Julius February 9, 2024

1 Definition

- Divide the problem into n subproblems.
- Conquer: solve the subproblems recursively. If trivial, solve the subproblems in a straightforward manner.
- Combine the solutions to the subproblems into the solution for the original problem. (Most challening)

Merge Sort: the divide step is to divide the array into two halves. The conquer step is to sort the two halves recursively. The combine step is to merge the two sorted halves into one sorted array. There are total log(n) levels, and at each level, the total work is $2^i * (c * n/2^i)$. Thus, the total work is $\Theta(nlogn)$.
By the way, we can actually use the merge-insertion sort hybrid to improve the performance, because insertion sort is faster than merge sort when the array size is small.

Proof of correctness. Let P(n) be the proposition that merge sort correctly sorts any array of length n.
Base case: P(1) is an array of 1 element, this array is already sorted. True.
Inductive step: Assume P(k) is true for all $1 \leq k < n$. We need to show that merge sort also works correctly on any array of size n. Suppose we call merge sort on an array of size n. It will then recursively call merge sort on two arrays of size n/2. By the inductive hypothesis, merge sort will correctly sort the two arrays of size n/2. Then, the merge step will correctly merge the two sorted arrays into one sorted array of size n.
*Note: In this case, the POC for the combine step is trivial. If the combine step is not trivial, it has to be proven separately. □

Runtime Analysis by Divide and Conquer:
Divide: $\Theta(1)$
Conquer: If the original problem takes T(n), the two subproblems take $2T(n/2)$
Combine: $\Theta(n)$.
Thus, the total runtime is $T(n) = 2T(n/2) + \Theta(n) + \Theta(1)$.
Our recurrence equation for a Divide and Conquer solution usually looks like:

$$T(n) = \begin{cases} \Theta(1), n \leq c \\ a * T(n/b) + D(n) + C(n), n > c \end{cases}$$

Where a is the number of subproblems, n/b is the size of each subproblem, D(n) is the time to divide the problem, and C(n) is the time to combine the solutions.

2 Master Theorem

The Master Theorem is a general method for solving recurrences of the form: $T(n) = a * T(n/b) + f(n)$, where $a \geq 1, b \geq 1$ are constants, and $f(n)$ is a asymptotically positive function. Then T(n) can be bounded as follows:

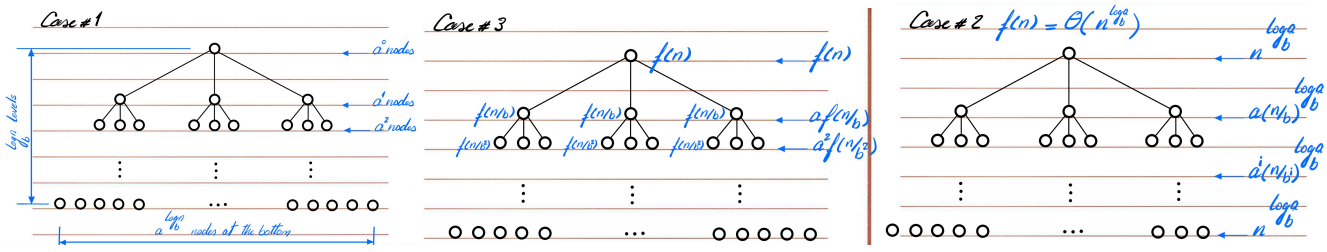
$$T(n) = \begin{cases} \Theta(n^{\log_b a}), \quad f(n) = O(n^{\log_b a - \epsilon}) \\ \Theta(n^{\log_b a} * \log n), \quad f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)), \quad f(n) = \Omega(n^{\log_b a + \epsilon}), \quad \exists c < 1 : a * f(n/b) \leq c * f(n) \end{cases}$$

Generalizing the second case: if $f(n) = \Theta(n^{\log_b a} * \log^k n)$, where $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} * \log^{k+1} n)$.
Intuition behind the Master Theorem Case 1: there are total $a^{log_b n} = n^{log_b a}$ nodes on the bottom level, then the total number of subproblems is:

$$n^{log_b a} + \frac{n^{log_b a}}{a} + \frac{n^{log_b a}}{a^2} + \cdots = \frac{a}{a-1} n^{log_b a} = \Theta(n^{log_b a})$$

The number of subproblems dominates the complexity, here the tree is bottom heavy.
Intuition behind the Master Theorem Case 3: if **regularity condition** $\exists c < 1, s.t.\ a * f(n/b) \leq c * f(n)$, then the cost of the combine and divide steps at each level must be a fraction of that in the previous level.

$$f(n) + cf(n) + c^2 f(n) + \cdots = \frac{f(n)}{1-c} = \Theta(f(n))$$



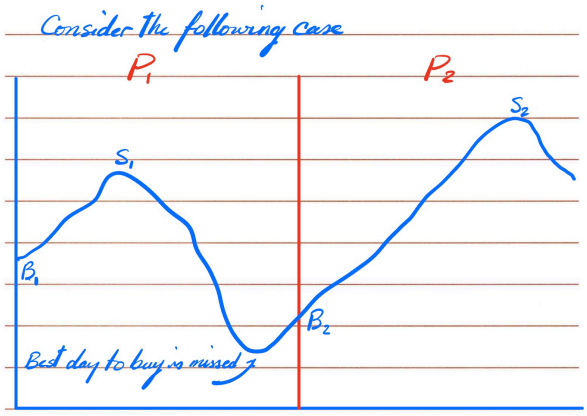
The cost of combine and divide steps dominates the complexity, here the tree is top heavy.
Intuition behind the Master Theorem Case 2: if $f(n) = \Theta(n^{\log_b a} * \log^k n)$, then at level i, the total work is:

$$a^i(n/b^i)^{log_b a} = n^{log_b a}(a^i b^{-i log_b a}) = n^{log_b a}(a^i(b^{log_b a*(-i)})) = n^{log_b a}(a^i a^{-i}) = n^{log_b a}$$

Since we have log n levels, the total work is $\Theta(n^{\log_b a} * \log n)$, here the cost of operations at each level is the same, the tree levels are equally weighted.

3 Stock Market Example

Given the price chart for a stock over a period of n day, which buy and sell days give us the maximum profit? (must buy before selling)
Brute Force: for each pair of days, calculate the profit, and return the pair with the maximum profit. The runtime is $\Theta(n^2)$.
Divide and Conquer: divide the array into two halves, and find the maximum profit in the left half $S_1 - B_1$ and right half $S_2 - B_2$, as well as the minimum and maximum price in the two halves $Min_1, Max_1, Min_2, Max_2$. Then the combine step is to find $max(S_1 - B_1, S_2 - B_2, Max_2 - Min_1)$. **Complexity Analysis by Master Theorem:**



a = b = 2, so $n^{\log_b a} = n$. The cost of the divide and combine steps $\Theta(1)$, so f(n) = $\Theta(1)$. Fall into case 1, $T(n) = \Theta(n)$.

4 Dense Matrix Multiplication

Given two n*n matrices A_{n*n} and B_{n*n} , we want to compute the product $C_{n*n} = A_{n*n} * B_{n*n}$.
Brute Force: for each element in the result matrix, we need to calculate the dot product of the corresponding row and column, so the runtime is $\Theta(n^3)$.
Divide and Conquer: divide the matrix into four n/2 * n/2 submatrices, and recursively compute the product of the submatrices. Then the combine step is to combine the four submatrices into the result matrix.

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$C_{11} = A_{11}B_{11} + A_{12}B_{21} \quad C_{12} = A_{11}B_{12} + A_{12}B_{22} \quad C_{21} = A_{21}B_{11} + A_{22}B_{21} \quad C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

Complexity Analysis by Master Theorem: a = 8, b = 2, so $n^{\log_b a} = n^3$. The cost of the divide and combine steps $\Theta(1) + \Theta(n^2)$, so f(n) = $\Theta(n^2)$. Fall into case 1, $T(n) = \Theta(n^3)$.
Strassen's Algorithm: Strassen's algorithm is a divide and conquer algorithm that multiplies two n*n matrices in $O(n^2.81)$ time. The idea is to reduce the number of subproblems from 8 to 7 by using the following 7 products:

$$Q = (A_{21} + A_{22}) * B_{11} \quad R = A_{11} * (B_{12} - B_{22}) \quad S = A_{22} * (B_{21} - B_{11}) \quad T = (A_{11} + A_{12}) * B_{22}$$

$$P = (A_{11} + A_{22}) * (B_{11} + B_{22}) \quad U = (A_{21} - A_{11}) * (B_{11} + B_{12}) \quad V = (A_{12} - A_{22}) * (B_{21} + B_{22})$$

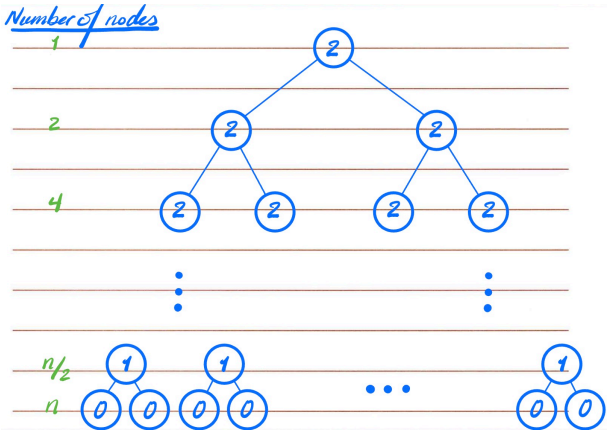
Then we can compute the result matrix using the 7 products:

$$C_{11} = P + S - T + V \quad C_{12} = R + T$$
$$C_{21} = Q + S \quad C_{22} = P + R - Q + U$$

Complexity Analysis by Master Theorem: a = 7, b = 2, so $n^{\log_b a} = n^{log_2 7} \approx n^{2.81}$. The cost of the divide and combine steps $\Theta(1) + \Theta(n^2)$, so f(n) = $\Theta(n^2)$. Fall into case 1, $T(n) = \Theta(n^{2.81})$.

5 Finding the Minimum and Maximum in an unsorted array

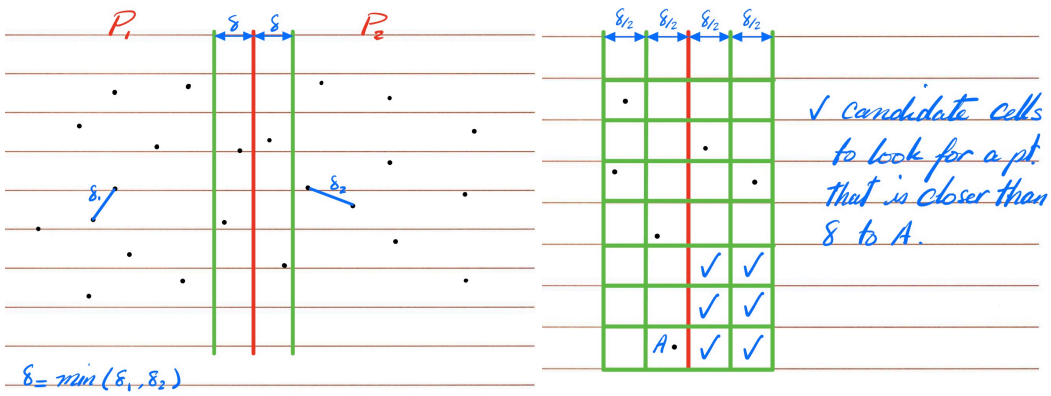
Brute Force: for each element in the array, compare it with the current minimum and maximum, and update the minimum and maximum if necessary. The number of comparison is $(n-1) + (n-1) = 2n - 2 = \Theta(n)$.
Divide and Conquer: divide the array into two halves, and find the minimum and maximum in the left half and right half. Then the combine step needs two comparisons for the minimum and maximum in the two halves. So our recursion tree is a full binary tree, where the nodes represent the number of comparisons. Remember



for full binary tree, if the nodes at the bottom level is n, then all nodes at the upper levels are n-1, so the total comparisons are $2 * (n-1) - n/2 = 3n/2 - 2$

6 Closest Pair of Points

Given n points in the Euclidean plane, find the pair of points with the smallest distance between them.
Brute Force: for each pair of points, calculate the distance, and return the pair with the minimum distance. The runtime is $(n-1) + (n-2) + \cdots + 1 = \Theta(n^2)$.
Divide and Conquer: divide the points into two halves P_1, P_2, and find the closest pair of points in two halves δ_1, δ_2. Then the combine step also needs to consider the case where one point in P_1 the other in P_2 and they are close to the dividing line. Let $\delta = \min(\delta_1, \delta_2)$, we can consider a $\frac{\delta}{2} * \frac{\delta}{2}$ square, then the farthest two points



within this square are $\sqrt{2}\delta/2 < \delta$, so there are at most one point in each $\delta/2$ square. For every point in this 2δ width band, we can start search by the y-coordinate, and only need to consider at most 11 other points above it to check if two points are in two different halves with distance less than δ.

Implementation of Divide and Conquer:

$closestPair(P)$:
Construct lists of points sorted by x and y coordinates P_x, P_y. $\Theta(nlgn)$
return $(p_0, p_1) = closestPairRec(P_x, P_y)$
$closestPairRec(P_x, P_y)$:
if $|P| \leq 3$, return the closest pair of points in P by brute force.
else
construct $P_{xL}, P_{xR} = P_x[: |P|/2], P_x[|P|/2 :]$, i.e. the left and right half of the P_x, and sort the points in P_{xL}, P_{xR} by their y-coordinate, denoted by P_{xLy}, P_{xRy}. $\Theta(1) + \Theta(n)$
$(q_0, q_1) = closestPairRec(P_{xL}, P_{xLy})$; $(r_0, r_1) = closestPairRec(P_{xR}, P_{xRy})$;
$\delta = \min(dist(q_0, q_1), dist(r_0, r_1))$; $\Theta(1)$
Let S be the set of points in P with x-coordinate in $[P_x[|P|/2].x - \delta, P_x[|P|/2].x + \delta]$, and sort the points in S by their y-coordinate, denoted by S_y. $\Theta(n)$
For each point p_i in S_y, compare it with the next 11 points in S_y to find the closest pair of points (s_0, s_1) in S. $\Theta(n)$
If $dist(s_0, s_1) < \delta$, return (s_0, s_1);
else if $dist(q_0, q_1) < dist(r_0, r_1)$, return (q_0, q_1);
else return (r_0, r_1). $\Theta(1)$

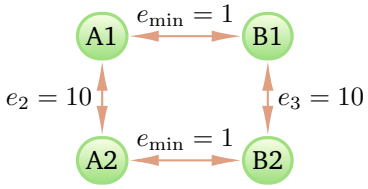
Complexity Analysis by Master Theorem: a = 2, b = 2, so $n^{\log_b a} = n$. The cost of the divide and combine steps $\Theta(1) + \Theta(n)$, so f(n) = $\Theta(n)$. Fall into case 2, $T(n) = \Theta(nlgn)$. And the cost of preparation step is also $\Theta(nlgn)$, so the worst-case runtime is $\Theta(nlgn)$.

7 Nearest neighbor search (NNS)

given a set S of points in a space M and a query point $q \in M$, find the closest point in S to q.
A direct generalization of this problem is a k-NN search, where we need to find the k closest points.

8 Discussion Questions

1. Suppose we have two graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, along with their MST T_1, T_2. Now consider a new graph $G = (V, E)$ such that $V = V_1 \cup V_2$ and $E = E_1 \cup E_2 \cup E_3$, where E_3 is the set of edges that connect the vertices in V_1 and V_2. Let $e_{\min} = \min_{i \in E_3}\{e_i\}, T = T_1 \cup T_2 \cup \{e_{\min}\}$. Is T a MST of G?



Consider the graph above, where $G_1 = (V_1, E_1)$ is a complete graph with vertices $A1, A1$, and $G_2 = (V_2, E_2)$ is a complete graph with vertices $B2, B2$. Then $T_1 \cup T_2 \cup \{e_{\min}\} = 10 + 10 + 1$, while the MST of G is 1 + 1 + 10. So T is not a MST of G.

2. Solve the following recurrences using the Master Method:

a. $A(n) = 3A(n/3) + 15$
a = 3, b = 3, so $n^{\log_b a} = n$. The cost of the divide and combine steps $f(n) = 15 = \Theta(1)$.
Fall into case 1, $A(n) = \Theta(n)$.
b. $B(n) = 4B(n/2) + n^3$
a = 4, b = 2, so $n^{\log_b a} = n^2$. The cost of the divide and combine steps $f(n) = n^3 = \Theta(n^3)$.
Fall into case 3, now we need to check whether $\exists c < 1, s.t.\ af(n/b) \leq cf(n)$, $4 * (n/2)^3 = n^3/2 = 0.5f(n)$, so we can let $0.5 \leq c < 1$, the inequality checks out, $B(n) = \Theta(n^3)$.
c. $C(n) = 4C(n/2) + n^2$
a = 4, b = 2, so $n^{\log_b a} = n^2$. The cost of the divide and combine steps $f(n) = n^2 = \Theta(n^2)$.
Fall into case 2, $C(n) = \Theta(n^2 * \log n)$.
d. $D(n) = 4D(n/2) + n$
a = 4, b = 2, so $n^{\log_b a} = n^2$. The cost of the divide and combine steps $f(n) = n = \Theta(n)$.
Fall into case 1, $D(n) = \Theta(n^2)$.

3. There are 2 sorted arrays A and B of size n each. Design a D&C algorithm to find the median of the array obtained after merging the above 2 arrays (i.e. array of length 2n). Discuss its runtime complexity.

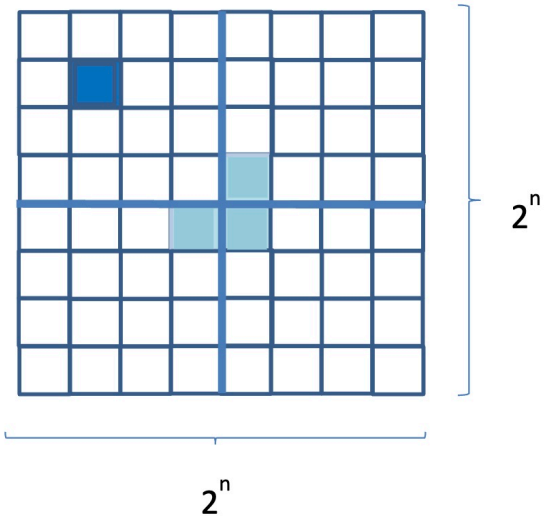
Median of Two Sorted Arrays:

Find the median of two sorted arrays m_A, m_B.
If $m_A == m_B$ return m_A.
If $m_A < m_B$, then the median is in the right half of A and the left half of B.
else the median is in the right half of B and the left half of A.
Solve the subproblem recursively.

Complexity Analysis by Master Theorem: Divide step $\Theta(1)$ including find the medians of A and B and throw away half of two arrays. There is no combine step. The number of subproblems at each step is 1, and the size of subproblems is $n/2$, so we can apply Master Method: a = 1, b = 2, so $n^{\log_b a} = n^0 = 1$. The cost of the divide and combine steps $\Theta(1)$, so f(n) = $\Theta(1)$. Fall into case 2, $T(n) = \Theta(\log n)$.

4. A tromino is a figure composed of three 1×1 squares in the shape of an L. Given a $2^n \times 2^n$ checkerboard with 1 missing square, tile it with trominoes. Design a D&C algorithm and discuss its runtime complexity.

The figure below shows a $2^n \times 2^n$ checkerboard with a hole highlighted in dark blue. Here is a divide and conquer solution.



Divide and Conquer Solution:

Divide the grid into 4 equal grids of size $2^{n-1} \times 2^{n-1}$.
Add holes to the three grids that do not have a hole in them. The position of the new holes should be at the center of the grid so that when each region is solved/tiled recursively we can cover the remaining three holes with one tromino.
Solve all 4 subproblems recursively. During recursion, when we reach 2×2 grids with a single hole, we can solve the problem directly by placing a single tromino to tile the grid.
When combining the solutions, place a tile over three holes in the center to complete tiling.

Complexity Analysis by Master Theorem: Divide step $\Theta(1)$ including divide the grid into 4 equal grids and add holes to the three grids that do not have a hole in them. Combine step just place a tromino in the center to cover the three holes $\Theta(1)$. The number of subproblems at each step is a=4, and the size of subproblems is $2^{n-1} \times 2^{n-1}$, so b=2. a = 4, b = 2, so $n^{\log_b a} = n^2$. The cost of the divide and combine steps $\Theta(1)$, so f(n) = $\Theta(1)$. Fall into case 1, $T(n) = \Theta(n^2)$.