

CSE 317: Design and Analysis of Algorithms

Divide and Conquer Algorithms

Shahid Hussain

Week 2: Fall 2025

SMCS, IBA

Divide and Conquer Algorithms

Divide and Conquer Algorithms

- We start with an *original problem*
- **Divide** it into two or more *subproblems* (ideally of approximately equal sizes)
- **Solve** each subproblem *recursively*
- And **combine** the solutions of subproblems to form a solution for the original problem

1

Mergesort Algorithm

- We consider **MERGESORT** as a first example of divide and conquer algorithms
- Let $\langle a_1, \dots, a_n \rangle$ be the input sequence to be sorted
- **MERGESORT** divides the array into two (almost) equal parts as:
$$\langle a_1, \dots, a_{\lfloor n/2 \rfloor} \rangle \quad \text{and} \quad \langle a_{\lfloor n/2 \rfloor + 1}, \dots, a_n \rangle$$
- We use **MERGESORT** to sort these two subsequences (subproblems)
- Let α and β be the two sorted sequences we receive after recursive calls
- Now we combine (merge) these lists to form a new list
- We compare first element of α with first element of β and transfer the smaller element to the new sequence and move the pointer where we take element from
- If at any point if one of the sequences α or β becomes empty we concatenate the other sequence to the new sequence.

2

Mergesort Algorithm

Algorithm: MERGESORT

Input: A sequence $A = \langle a_1, a_2, \dots, a_n \rangle$

Output: The sorted sequence $A' = \langle a_{i_1}, a_{i_2}, \dots, a_{i_n} \rangle$ s.t. $a_{i_1} \leq a_{i_2} \leq \dots \leq a_{i_n}$

1. **if** $n > 1$ **then**
2. $\alpha \leftarrow \text{MERGESORT}(\langle a_1, \dots, a_{\lfloor n/2 \rfloor} \rangle)$
3. $\beta \leftarrow \text{MERGESORT}(\langle a_{\lfloor n/2 \rfloor + 1}, \dots, a_n \rangle)$
4. $\text{MERGE}(\alpha, \beta)$
5. **return** A

3

Mergesort Algorithm: Merge

Algorithm: MERGE

Input: Two sorted lists $A = \langle a_1, \dots, a_k \rangle$ and $B = \langle b_1, \dots, b_l \rangle$

Output: Combined sorted sequences of A and B

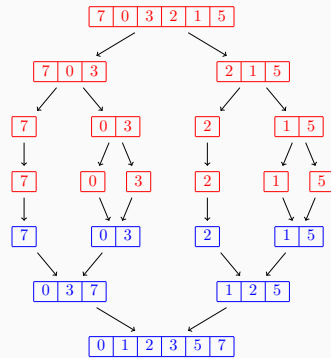
1. **if** $k = 0$ **then return** $\langle b_1, \dots, b_l \rangle$
2. **if** $l = 0$ **then return** $\langle a_1, \dots, a_k \rangle$
3. **if** $a_1 \leq b_1$ **then return** $\langle a_1 \rangle \circ \text{MERGE}(\langle a_2, \dots, a_k \rangle, \langle b_1, \dots, b_l \rangle)$
4. **else return** $\langle b_1 \rangle \circ \text{MERGE}(\langle a_1, \dots, a_k \rangle, \langle b_2, \dots, b_l \rangle)$

[Here \circ denotes concatenation.]

4

Mergesort Algorithm: Visualization

Sorting the sequence $\langle 7, 0, 3, 2, 1, 5 \rangle$ using MERGESORT.



5

Mergesort Algorithm: Time Complexity

- Let the two sorted lists A and B are of sizes k and l , respectively
- We can show that the procedure MERGE can merge the lists A and B in time $O(k + l)$
- So if $|A| + |B| = n$ then running time of MERGE is $O(n)$
- Let $T(n)$ represents the running time of MERGESORT then:

$$T(n) = 2T\left(\frac{n}{2}\right) + n \quad \text{with } T(1) = 1$$

- This is a *divide and conquer recurrence* and we show that

$$T(n) = O(n \log n)$$

6

Mergesort recurrence

- Solving the recurrence $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$ with $T(1) = 1$ and $n = 2^k$

$$\begin{aligned}
 T(n) &= 2T\left(\frac{n}{2}\right) + n \\
 &= 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n = 2^2T\left(\frac{n}{2^2}\right) + n + n \\
 &= 2^2\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + n + n = 2^3T\left(\frac{n}{2^3}\right) + n + n + n \\
 &\vdots \\
 &= 2^kT\left(\frac{n}{2^k}\right) + kn \quad \text{since } T(1) = 1 \\
 &= 2^kT(1) + kn = n + kn = n(1 + k) \\
 &= n(1 + \log_2 n) = O(n \log n)
 \end{aligned}$$

7

Divide and Conquer Recurrences

- Divide and conquer recurrences are of the form:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

where $f(n)$ is the time taken to divide the problem into subproblems and combine the solutions of subproblems

Theorem (Master Theorem)

Let $T(n) = aT(n/b) + f(n)$ be a divide and conquer recurrence with $a \geq 1$, $b > 1$ and $f(n)$ is asymptotically positive. Then $T(n)$ has the following asymptotic bounds:

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } f(n) = O(n^{\log_b a - \epsilon}) \text{ for some } \epsilon > 0 \\ \Theta(n^{\log_b a} \log n) & \text{if } f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & \text{if } f(n) = \Omega(n^{\log_b a + \epsilon}) \text{ for some } \epsilon > 0 \end{cases}$$

8

Maximum Subarray Sum

- For a given array $A[1..n]$ of n numbers, the *maximum subarray sum*, denoted as \mathcal{M} is the largest sum of contiguous subarray of $A[i..j]$ where $1 \leq i \leq j \leq n$, i.e.,

$$\mathcal{M}(A) = \sum_{x=i}^j A[x]$$

- For example, for $A = [-2, 3, 4, -1, 5, -3, 2, -1]$, $\mathcal{M}(A) = 11$ for the subarray $[3, 4, -1, 5]$

9

Maximum Subarray Sum: Trivial Solution

Algorithm: BRUTEFORCE-MAX-SUBARRAY-SUM

Input: An array $A[1..n]$ of n numbers

Output: The maximum subarray sum $\mathcal{M}(A)$

1. $\mathcal{M} \leftarrow -\infty$
2. **for** $i \leftarrow 1$ **to** n
3. **for** $j \leftarrow i$ **to** n
4. $s \leftarrow 0$
5. **for** $k \leftarrow i$ **to** j
6. $s \leftarrow s + A[k]$
7. **if** $s > \mathcal{M}$ **then** $\mathcal{M} = s$
8. **return** \mathcal{M}

The time complexity of above algorithms is $\Theta(n^3)$

10

Maximum Subarray Sum: Divide and Conquer

- Let us design a divide and conquer algorithm
- We divide the array into two almost equal parts and find the maximum subarray sum in the left and right parts recursively
- At this point we have three cases:
 1. The maximum subarray sum is in the left part
 2. The maximum subarray sum is in the right part
 3. The maximum subarray sum crosses the middle point
- Thus, the algorithm returns the maximum of these three cases
- The crossover sum can be computed in $O(n)$ time by starting from the middle point and going left and right to find the maximum sum
- If $T(n)$ represents the time complexity of the algorithm then we have the following recurrence:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

11

Maximum Subarray Sum: Divide and Conquer

Algorithm: DC-MAX-SUBARRAY-SUM

Input: An array $A[l..h]$ of $h - l + 1$ numbers

Output: The maximum subarray sum $\mathcal{M}(A)$

1. **if** $l = h$ **then return** $A[l]$
2. $m \leftarrow \lfloor (l + h) / 2 \rfloor$
3. $\mathcal{M}_L \leftarrow \text{DC-MAX-SUBARRAY-SUM}(A[l..m])$
4. $\mathcal{M}_R \leftarrow \text{DC-MAX-SUBARRAY-SUM}(A[m + 1..h])$
5. $\mathcal{M}_C \leftarrow \text{MAX-CROSSING-SUM}(A[l..h])$
6. **return** $\max\{\mathcal{M}_L, \mathcal{M}_R, \mathcal{M}_C\}$

Algorithm: MAX-CROSSING-SUM

Input: An array $A[l..h]$ of $h - l + 1$ numbers

Output: The maximum crossing sum

1. $m \leftarrow \lfloor (l + h) / 2 \rfloor$, $s \leftarrow 0$, $\text{left_sum} \leftarrow -\infty$
2. **for** $i \leftarrow m$ **downto** l
3. $s \leftarrow s + A[i]$
4. **if** $s > \text{left_sum}$ **then** $\text{left_sum} \leftarrow s$
5. $s \leftarrow 0$, $\text{right_sum} \leftarrow -\infty$
6. **for** $i \leftarrow m + 1$ **to** h
7. $s \leftarrow s + A[i]$
8. **if** $s > \text{right_sum}$ **then** $\text{right_sum} \leftarrow s$
9. **return** $\text{left_sum} + \text{right_sum}$

12

Closest-Pair Problem

- Given a set of n points $P = \{p_1 = (x_1, y_1), \dots, p_n = (x_n, y_n)\}$
- The *closest-pair* problem asks to find the pair of points p_i and p_j that are closest to each other i.e., $d(p_i, p_j)$ is minimum, where

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

- A brute force algorithm will find the distance between all pairs of points and return the minimum distance
- This algorithm has a time complexity of $O(n^2)$ as there are $n(n - 1)/2$ different pairs of points
- We will see a divide and conquer algorithm that solves this problem in $O(n \log n)$ time

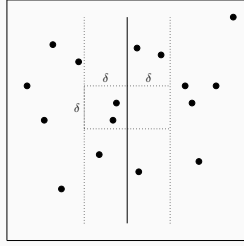
13

Closest-Pair Problem: Divide and Conquer

- The first step is to sort the points according to their x -coordinates
- We divide the points into two almost equal parts P_L and P_R and find the closest pair in the left and right parts recursively
- Let l be the vertical line that separates P_L and P_R
- Let δ_L and δ_R be the minimum distance between two points in P_L and P_R , respectively
- Let δ' be the minimum distance between a point in P_L and a point in P_R
- The closest pair is either in P_L or P_R or it crosses the line l , therefore the closest pair is the minimum of δ_L , δ_R and δ'

14

Closest-Pair Problem: Divide and Conquer



- We set $\delta = \min\{\delta_L, \delta_R\}$
- If there is a pair of points $p_i \in P_L$ and $p_j \in P_R$ such that $d(p_i, p_j) < \delta$ then this is the closest pair
- If the closest pair consists of some point in P_L and P_R then we need to check only those points that are within δ distance from the line l (both sides)
- We can now run a linear scan to find the closest pair in this region from top to bottom (see the figure)

15

Closest-Pair Problem: Divide and Conquer

- Let $T(n)$ be the time complexity of the algorithm
- We have the following recurrence:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n \log n)$$

16

Integer Multiplication

- Let X and Y be two n -bit integers
- Adding x and y takes $O(n)$ bit operations (linear time)
- Multiplying x and y takes $O(n^2)$ bit operations (quadratic time)
- Product of two n -bit numbers is at most a $2n$ -bit number

$$\begin{array}{r} 1001 \\ \times 1010 \\ \hline 0000 \\ 1001 \\ 0000 \\ 1001 \\ \hline 1011010 \end{array}$$

17

Integer Multiplication: Divide and Conquer

- Let $X = \langle x_1, x_2, \dots, x_n \rangle_2$ and $Y = \langle y_1, y_2, \dots, y_n \rangle_2$ be two n -bit integers, assuming $n = 2^k$, $k \in \mathbb{N}$
- We can write $X = X_L X_R$ and $Y = Y_L Y_R$, where
- $X_L = \langle x_1, x_2, \dots, x_{n/2} \rangle_2$, $X_R = \langle x_{n/2+1}, x_{n/2+2}, \dots, x_n \rangle_2$
- $Y_L = \langle y_1, y_2, \dots, y_{n/2} \rangle_2$, $Y_R = \langle y_{n/2+1}, y_{n/2+2}, \dots, y_n \rangle_2$
- So, we can write $X = 2^{n/2} X_L + X_R$ and $Y = 2^{n/2} Y_L + Y_R$
- The product of X and Y is:

$$XY = (2^{n/2} X_L + X_R)(2^{n/2} Y_L + Y_R) = 2^n X_L Y_L + 2^{n/2} (X_L Y_R + X_R Y_L) + X_R Y_R$$

- So we can compute product of two n -bit integers by computing four products of $n/2$ -bit integers

18

Integer Multiplication: Divide and Conquer

- If time to multiply X and Y is $T(n)$ then we have the following recurrence:

$$T(n) = 4T\left(\frac{n}{2}\right) + O(1) = \Theta(n^2) \quad \text{with} \quad T(1) = 0$$

- If we use *Karatsuba's algorithm* then we have the following recurrence:

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n) = O(n^{\log_2 3}) \approx O(n^{1.5849}) \quad \text{with} \quad T(1) = 1$$

- That is if we can somehow reduce the number of multiplications from four to three we can improve the running time

19

Integer Multiplication: Divide and Conquer

- $XY = 2^n X_L Y_L + 2^{n/2}(X_L Y_R + X_R Y_L) + X_R Y_R$
- We observe that:

$$(X_L Y_R + X_R Y_L) = (X_L + X_R)(Y_L + Y_R) - X_L Y_L - X_R Y_R$$

- Since $X_L Y_L$ and $X_R Y_R$ is already computed to evaluate XY we get:

$$\begin{aligned} XY &= 2^n X_L Y_L + 2^{n/2}(X_L Y_R + X_R Y_L) + X_R Y_R \\ &= 2^n X_L Y_L + 2^{n/2}((X_L + X_R)(Y_L + Y_R) - X_L Y_L - X_R Y_R) + X_R Y_R \end{aligned}$$

- So we can compute XY using three multiplications of $n/2$ -bit integers instead of four to get:

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n) = O(n^{\log_2 3}) \approx O(n^{1.5849}) \quad \text{with} \quad T(1) = 1$$

20

Integer Multiplication: Example

- Let $X = 1101_2$ and $Y = 1010_2$, $X_L = 11$, $X_R = 01$, $Y_L = 10$ and $Y_R = 10$
- Here $n = 2^2 = 4$ and $n/2 = 2^1 = 2$

$$\begin{aligned} XY &= 1101_2 \times 1010_2 \\ &= 2^2(11)(01) + 2^1((11 + 01)(10 + 10) - (11)(10) - (01)(10)) + (01)(10) \end{aligned}$$

- Each multiplication in above will be carried out recursively

21

Matrix Multiplication

- If X and Y are two matrices of sizes $m \times n$ and $n \times r$ then the product $Z = AB$ is a matrix of size $m \times r$

$$\begin{array}{c} \begin{array}{c} \uparrow \\ m \\ \downarrow \end{array} \left(\begin{array}{c} \xleftarrow{n} \\ \bullet \quad \dots \quad \bullet \\ \xrightarrow{n} \end{array} \right) \times \begin{array}{c} \begin{array}{c} \xleftarrow{r} \\ \bullet \\ \vdots \\ \bullet \\ \xrightarrow{r} \end{array} \\ n \end{array} = \begin{array}{c} \begin{array}{c} \xleftarrow{r} \\ \bullet \\ \xrightarrow{r} \end{array} \\ m \end{array} \\ \text{row } i \quad \times \quad \text{column } j \quad = \quad (i, j)\text{-cell} \end{array}$$

- In general AB is not the same as BA .

22

Matrix Multiplication

- Let X and Y be two matrices of size $n \times n$ such that $n = 2^k$ for some $k > 0$.
- We can break the problem into subproblems e.g.,

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

- Now XY can be computed as

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

- To compute the size n product XY we now compute eight size $n/2$ products, AE, BG, \dots, DH
- $T(n) = 8T(n/2) + O(n^2) = O(n^3)$

23

Strassen's Algorithm

- Strassen observed that XY can be computed from just seven $n/2 \times n/2$ subproblems i.e.,

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

- where

$$\begin{aligned} P_1 &= A(F - H) & P_5 &= (A + D)(E + H) \\ P_2 &= (A + B)H & P_6 &= (B - D)(G + H) \\ P_3 &= (C + D)E & P_7 &= (A - C)(E + F) \\ P_4 &= D(G - E) \end{aligned}$$

- The new running time is

$$T(n) = 7T(n/2) + O(n^2) = O(n^{\log_2 7}) \approx O(n^{2.81})$$

24