Divide-and-Conquer Paradigm

CS313E - Elements of Software Design

Kia Teymourian

05/11/2022

Agenda

1. Divide and Conquer
2. Merge Sort
3. Recurrences
4. Strassen's algorithm for Matrix Multiplication

Divide-and-Conquer Paradigm

The divide-and-conquer paradigm has the following three steps at each level of the recursion:

- First, **divide** the problem into a number of subproblems that are smaller instances of the same problem.
- Then, **conquer** the subproblems by solving them recursively.
  If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
- Finally, **combine** the solutions to the subproblems into the solution for the original problem.

Merge Sort

We have an array A with n elements to sort.

125	44	780	128	816	743	594	170
0	1	2	3	4	5	6	7

Merge Sort

We have an array A with n elements to sort.

125	44	780	128	816	743	594	170
0	1	2	3	4	5	6	7

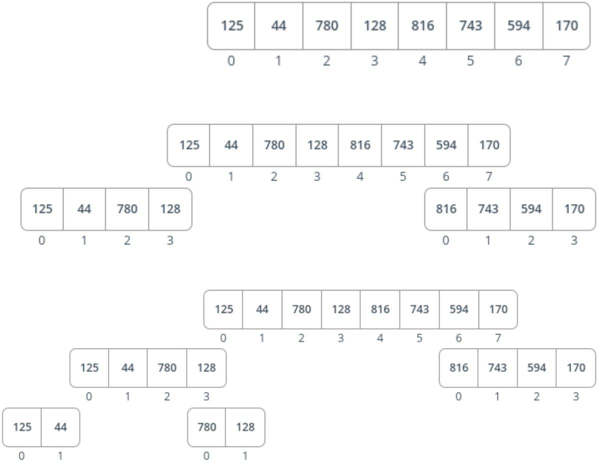
125	44	780	128	816	743	594	170
0	1	2	3	4	5	6	7

125	44	780	128
0	1	2	3

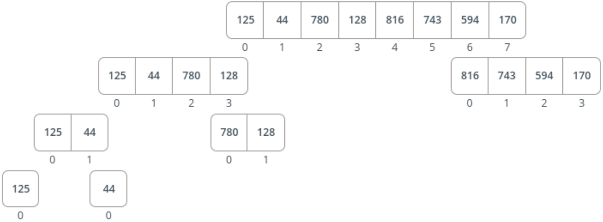
816	743	594	170
0	1	2	3

Merge Sort

We have an array A with n elements to sort.

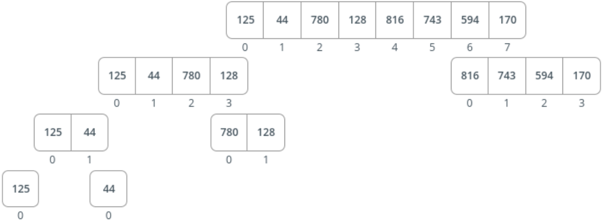


Merge Sort

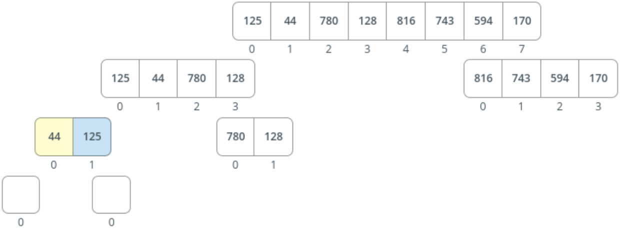


125 44 780 128 816 743 594 170
0 1 2 3 4 5 6 7
125 44 780 128
0 1 2 3
816 743 594 170
0 1 2 3
125 44
0 1
780 128
0 1
125
0
44
0

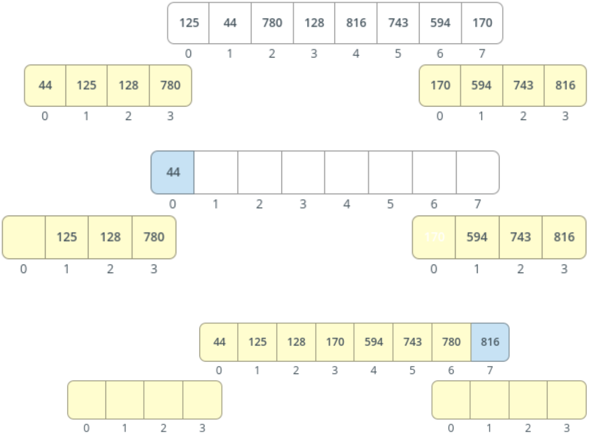
Merge Sort



Start merge the sorted array.



Merge Sort


125 44 780 128 816 743 594 170
0 1 2 3 4 5 6 7
44 125 128 780
0 1 2 3
170 594 743 816
0 1 2 3
44
0 1 2 3 4 5 6 7
125 128 780
0 1 2 3
170 594 743 816
0 1 2 3
44 125 128 170 594 743 780 816
0 1 2 3 4 5 6 7
0 1 2 3
0 1 2 3

Merge Sort

The merge sort procedure works as follows:

1. First we compute length n_1 of the subarray $A[p..q]$, and n_2 the length of subarray $A[q+1..r]$.
2. Then we create arrays L and R (left side and right side arrays) of the length n_1+1 and n_2+1 respectively
3. The we do two for-loops and copy the subarray $A[p..q]$ into $L[1..n_1]$, and subarray $A[q+1..r]$ into $R[1..n_2]$
4. Then we put the sentinels (infinity values) at the end of the subarrays $L(n_1+1)$ and $R[n_1+1]$
5. Then we start for-loop over the subarrays, divide, compare and merge the results.

Algorithm 1 Merge(A, p, q ,r) , Merge Procedure

```
n1 = q - p + 1,    n2 = r - q
let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
for i = 1 to n1 do
    L[i] = A[p + i - 1]
end for
for j = 1 to n2 do
    R[j] = A[q + j]
end for
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1,   j = 1
for k = p to r do
    if L[i] ≤ R[j] then
        A[k] = L[i]
        i = i + 1
    else
        A[k] = R[j]
        j = j + 1
    end if
end for
```

Merge Sort Algorith

Algorithm 2 Merge-Sort(A, p, r) , Merge Sort Algorithm

```
if p ≤ r] then
    q = ⌊(p + r)/2⌋
    Merge-Sort(A,p,q)
    Merge-Sort(A,q+1,r)
    Merge(A, p, q, r)
end if
```

Recurrences

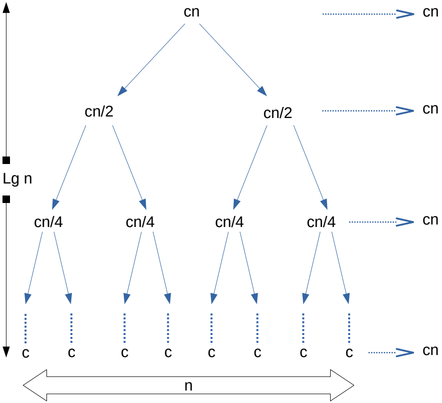
Recurrences - Running time of divide-and-conquer algorithms

- Recurrences provide us a good way to characterize the running time of divide-and-conquer algorithms.
- A recurrence is an equation or inequality to provide a function in terms of its value on smaller input sizes.

$$T(n) = C1 + 2T(n/2) + C2n$$

- $C1$: cost of divide
- $2T(n/2)$ cost of recursion
- $C2n$: cost of merge

Recurrence Tree Visualization - Merge Sort



- We stat with cn because it dominates the costs
- Number of leaves: n
- Number of levels: $1 + lg(n)$

$T(n) = (1 + lg(n)) \times cn = \Theta(nlgn)$

Recurrences - merge sort

$$T(n) = C1 + 2T(n/2) + C2n$$

$T(n) = (1 + lg(n)) \times cn = \Theta(nlgn)$
We can write this recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Recurrences - Merge Sort

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Parameters are:

- Number of recursive calls, or the number of sub-problems that we solve in our recurse algorithm
- Factor of the sub-problems or the factor that we divide the n size of the main problem into smaller problems
- The exponent of the running time outside of the recursive calls.

Solving the above recurrence:

- After solving the above recurrence, we have the running time for merge sort to be $\boldsymbol{O(n\ log(n))}$.
- **We will learn how to sovle recurrences** (Next lecture)

Strassen's algorithm for Matrix Multiplication

Strassen's algorithm

Simple divide and Conquer for Matrix Multiplication

- Use divide and conquer algorithms to do matrix multiplication
- Goal is to compute $C = A \cdot B$, n is exact power of 2 in each $n \times n$ matrix

$$C = A \cdot B$$

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Strassen's algorithm

Simple divide and Conquer for Matrix Multiplication

- Use divide and conquer algorithms to do matrix multiplication
- Goal is to compute $C = A \cdot B$, n is exact power of 2 in each $n \times n$ matrix

$$C = A \cdot B$$

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

The straightforward matrix multiplication has $\boldsymbol{T(n) = \Theta(n^3)}$.

Matrix Multiplication

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{aligned}$$

- We divide $n \times n$ into 4 $n/2 \times n/2$ matrices
- We assume n is an exact power of 2

Simple divide and Conquer for Matrix Multiplication

Algorithm 3 Simple divide and Conquer for Matrix Multiplication

SquareMatrixMultiply(A, B)
$n = A.rows$
let C be a new $n \times n$ matrix
if $n == 1$ **then**
    $c_{11} = a_{11} \cdot b_{11}$
else
    Partition A, B, and C
    $C_{11} = SquareMatrixMultiply(A_{11}, B_{11}) + SquareMatrixMultiply(A_{12}, B_{21})$
    $C_{12} = SquareMatrixMultiply A_{11}, B_{12}) + SquareMatrixMultiply(A_{12}, B_{22})$
    $C_{21} = SquareMatrixMultiply(A_{21}, B_{11}) + SquareMatrixMultiply(A_{22}, B_{21})$
    $C_{22} = SquareMatrixMultiply(A_{21}, B_{12}) + SquareMatrixMultiply(A_{22}, B_{22})$
end if
    return C
EndFunction

Running time of Simple divide and Conquer for Matrix Multiplication

$$
\begin{aligned}
T(1) &= \Theta(1) \\
T(n) &= \Theta(1) + 8T(n/2) + \Theta(n^2) \\
&= 8T(n/2) + \Theta(n^2)
\end{aligned}
$$

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

- Based on recurrence 1, the recursive matrix multiplication has $T(n) = \Theta(n^3)$

Running time of Simple divide and Conquer for Matrix Multiplication

$$\begin{aligned} T(1) &= \Theta(1) \\ T(n) &= \Theta(1) + 8T(n/2) + \Theta(n^2) \\ &= 8T(n/2) + \Theta(n^2) \end{aligned}$$

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

- Based on recurrence 1, the recursive matrix multiplication has $T(n) = \Theta(n^3)$
- The simple divide-and-conquer approach is no faster than the straightforward matrix multiplication

Can we do better than this?

Strassen's Method

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Let us the fine the following 10 helper matrices:

$$\begin{aligned}
S_1 &= B_{12} - B_{22} \\
S_2 &= A_{11} + A_{12} \\
S_3 &= A_{21} + A_{22} \\
S_4 &= B_{21} - B_{11} \\
S_5 &= A_{11} + A_{22} \\
S_6 &= B_{11} + B_{22} \\
S_7 &= A_{12} - A_{22} \\
S_8 &= B_{21} + B_{22} \\
S_9 &= A_{11} - A_{21} \\
S_{10} &= B_{11} + B_{12}
\end{aligned}$$

Strassen's Method

Let us define the following 7 matrices:

$$P_1 = A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22}$$
$$P_2 = S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}$$
$$P_3 = S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11}$$
$$P_4 = A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11}$$
$$P_5 = S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}$$
$$P_6 = S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}$$
$$P_7 = S_9 \cdot S_10 = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}$$

Strassen's Method

Let us define the following 7 matrices:

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\
P_2 &= S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\
P_3 &= S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\
P_4 &= A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11} \\
P_5 &= S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\
P_6 &= S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22} \\
P_7 &= S_9 \cdot S_1 0 = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}
\end{aligned}
$$

We can proof that the elements of the C matrix can be computed as follows:

$$
\begin{aligned}
\boldsymbol{C_{11}} &= \boldsymbol{P_5 + P_4 - P_2 + P_6} \\
\boldsymbol{C_{12}} &= \boldsymbol{P_1 + P_2} \\
\boldsymbol{C_{21}} &= \boldsymbol{P_3 + P_4} \\
\boldsymbol{C_{22}} &= \boldsymbol{P_5 + P_1 - P_3 - P_7}
\end{aligned}
$$

$$
\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6 \\
C_{12} &= P_1 + P_2 \\
C_{21} &= P_3 + P_4 \\
C_{22} &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

1. Divide the input matrices A and B and output matrix C into $\frac{n}{2} \times \frac{n}{2}$ sub-matrices. This takes $\Theta(1)$
2. Create the above 10 helper matrices $S_1, S_2, ..., S_{10}$, each of which is $\frac{n}{2} \times \frac{n}{2}$ in size, and is the sum of the differences of two matrices of step-1. This step takes $\Theta(n^2)$ to create all 10 matrices.
3. Use the matrices created in the above 2 steps, recursively compute 7 product matrices $P_1, P_2, \ldots, P_7$ each of which $\frac{n}{2} \times \frac{n}{2}$
4. Compute the desired submatrices $C_{11}, C_{12}, C_{21}, C_{22}$ of the result matrix C by adding and subtracting various combinations of the P_i matrices. We can compute this for all 4 submatrices in $\Theta(n^2)$ time.

Recurrence for the Strassen's Algorithm

We can setup the recurrence for the running time of Strassen's method as follows:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1 \end{cases}$$

Readings from CLRS Book (Introduction to Algorithms, 3rd Edition)

- Section 2.3.1 The divide-and-conquer approach
- Section 2.3.2 Analyzing divide-and-conquer algorithms
- Section 3.2. Standard notations and common functions
- Section 4.2. Strassen's algorithm for matrix multiplication
- Section 4.3 The substitution method for solving recurrences
- Section 4.5 The master method for solving recurrences