

LECTURE 05 — DIVIDE AND CONQUER (PART 1)

COMPSCI 308 — DESIGN AND ANALYSIS OF ALGORITHMS

Xing Shi Cai <https://newptcai.gitlab.io>

Duke Kunshan University, Kunshan, China

Jan-Mar 2026

SUMMARY

ITA 2.3 Merge Sort

ITA 4.1 The Maximum-Subarray Problem

ITA 4.2 Strassen's Algorithm for Matrix Multiplication

ASSIGNMENTS¹



Practice makes perfect!

Required Readings:

- Section 2.3
- Section 4.1
- Section 4.2

Required Exercises:

- Exercises 2.3: 1–3.
- Exercises 4.1: 1–4.
- Exercises 4.2: 1–5.

¹ ● Assignments will not be collected; however, quiz problems will be selected from them. (This includes both Readings and Exercises.)

ITA 2.3 MERGE SORT

DIVIDE AND CONQUER

Divide and Conquer is a common technique in designing algorithms:

- Knife icon: **Divide**: Break the problem into smaller sub-problems.
- Sabers icon: **Conquer**: Solve the smaller sub-problems.
- Plate icon: **Combine**: Merge the solutions to create a solution for the original problem.



Figure 1: How to divide and conquer a big  pizza

MERGE SORT

Merge Sort is a classic example of a divide and conquer algorithm.

 **Divide:** Split the unsorted array into two subarrays, each containing half of the elements.

 **Conquer:** Sort each subarray.

 **Combine:** Merge the two sorted subarrays.

```
1: Merge-Sort( $A, p, r$ )
2: if  $p < r$  then
3:    $q \leftarrow \lfloor(p + r)/2\rfloor$ 
4:   MERGE-SORT( $A, p, q$ )
5:   MERGE-SORT( $A, q + 1, r$ )
6:   MERGE( $A, p, q, r$ )
```

EXAMPLE OF MERGE SORT

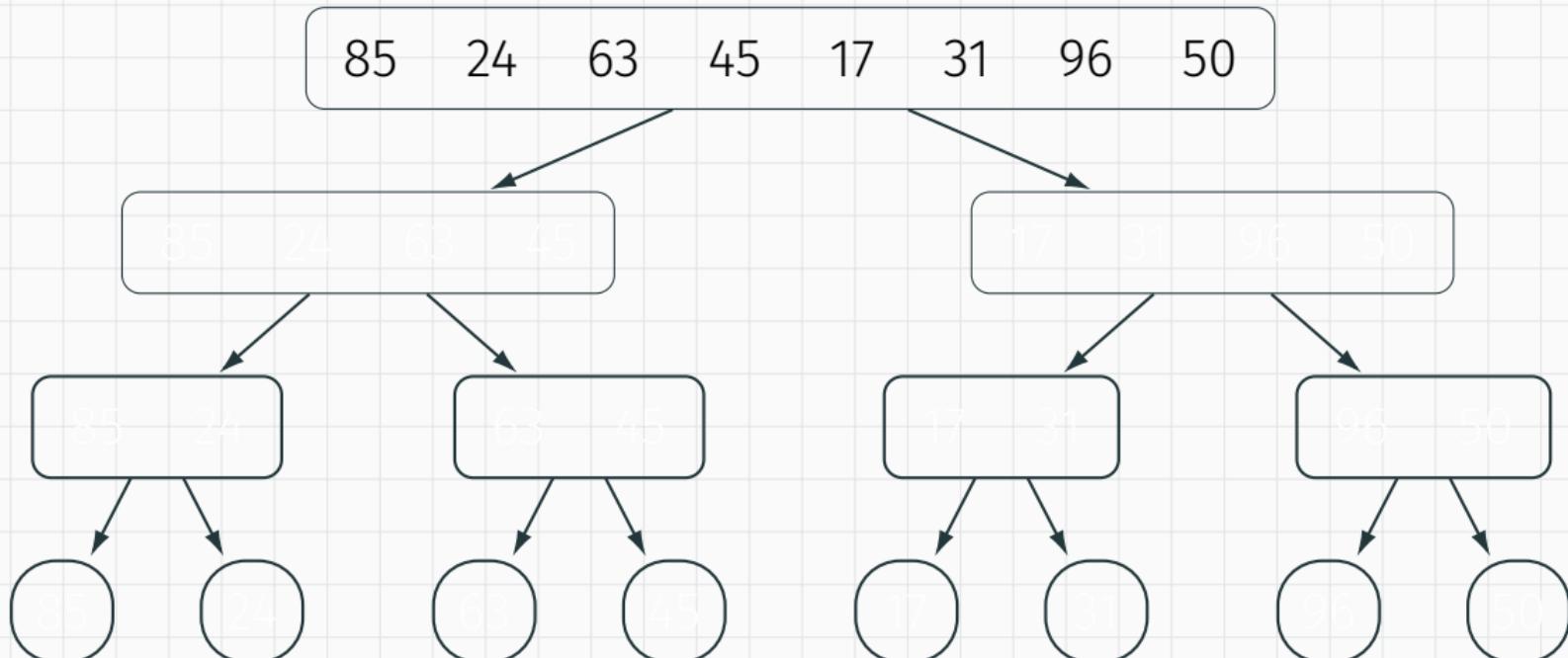


Figure 2: Divide

EXAMPLE OF MERGE SORT

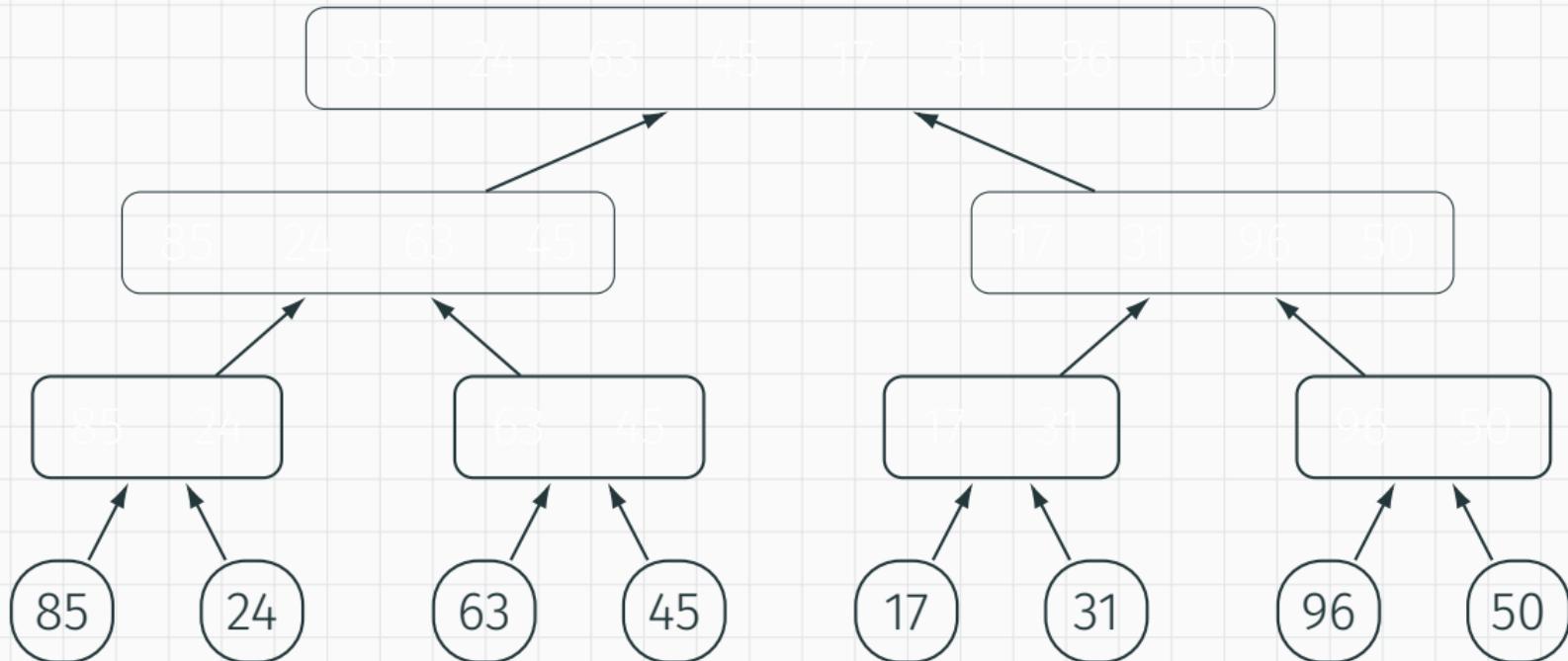


Figure 2: Combine

WHAT IS A SENTINEL?

A **sentinel** is a soldier whose job is to guard something.

— Oxford Learner's Dictionary

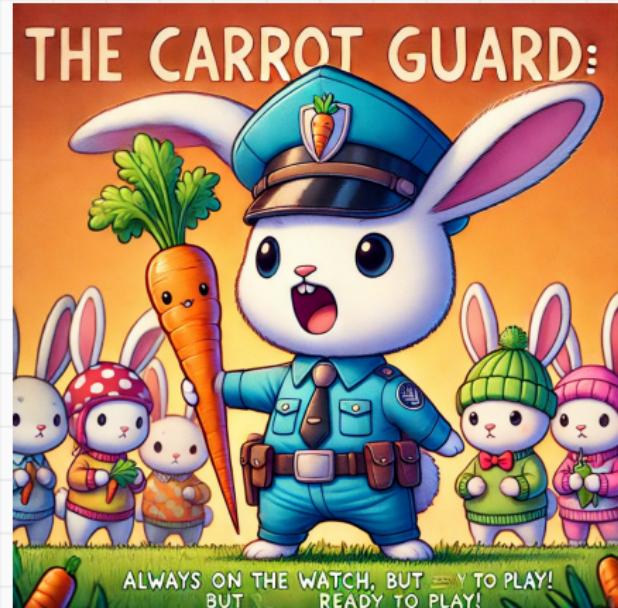
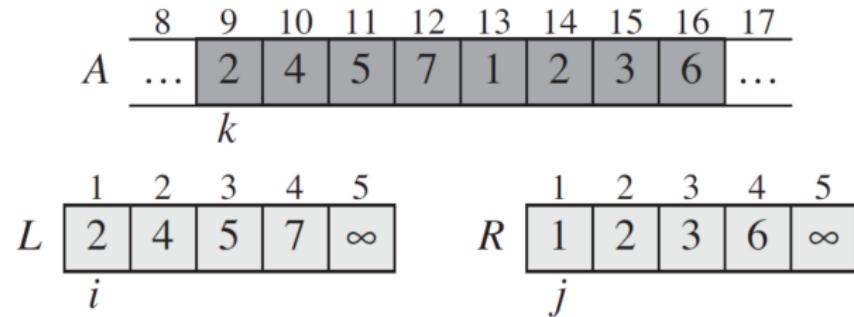


Figure 3: The Carrot Sentinel!

 EXAMPLE OF MERGE

Let's run $\text{Merge}(A, 9, 12, 16)$ given the following array:



PSEUDOCODE OF MERGE FUNCTION

```
1: Merge( $A, p, q, r$ )
2:  $n_1 \leftarrow q - p + 1$ 
3:  $n_2 \leftarrow r - q$ 
4: Let  $L[1 \dots n_1 + 1]$  and  $R[1 \dots n_2 + 1]$  be
   new arrays
5: for  $i \leftarrow 1$  to  $n_1$  do
6:    $L[i] \leftarrow A[p + i - 1]$ 
7: for  $j \leftarrow 1$  to  $n_2$  do
8:    $R[j] \leftarrow A[q + j]$ 
9:  $L[n_1 + 1] \leftarrow \infty$            ▷ A sentinel
10:  $R[n_2 + 1] \leftarrow \infty$       ▷ Another sentinel
```

PSEUDOCODE OF MERGE FUNCTION

```
1: Merge( $A, p, q, r$ )
2:  $n_1 \leftarrow q - p + 1$ 
3:  $n_2 \leftarrow r - q$ 
4: Let  $L[1 \dots n_1 + 1]$  and  $R[1 \dots n_2 + 1]$  be
   new arrays
5: for  $i \leftarrow 1$  to  $n_1$  do
6:    $L[i] \leftarrow A[p + i - 1]$ 
7: for  $j \leftarrow 1$  to  $n_2$  do
8:    $R[j] \leftarrow A[q + j]$ 
9:  $L[n_1 + 1] \leftarrow \infty$                                 ▷ A sentinel
10:  $R[n_2 + 1] \leftarrow \infty$     ▷ Another sentinel
11:  $i \leftarrow 1$ 
12:  $j \leftarrow 1$ 
13: for  $k \leftarrow p$  to  $r$  do
14:   if  $L[i] \leq R[j]$  then
15:      $A[k] \leftarrow L[i]$ 
16:      $i \leftarrow i + 1$ 
17:   else
18:      $A[k] \leftarrow R[j]$ 
19:      $j \leftarrow j + 1$ 
```



What is the running time of the above algorithm?

TIME COMPLEXITY OF MERGE SORT

Let $T(n)$ denote the running time of the merge sort algorithm. Then $T(n)$ satisfies

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ D(n) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + \Theta(n) & \text{otherwise.} \end{cases}$$

TIME COMPLEXITY OF MERGE SORT

Let $T(n)$ denote the running time of the merge sort algorithm. Then $T(n)$ satisfies

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ D(n) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + \Theta(n) & \text{otherwise.} \end{cases}$$

Usually the order of growth of such an algorithm does not depend on

- the floor and ceiling functions,
- and the boundary conditions.

This leads to the simplified equation

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

Later we will see

$$T(n) = \Theta(n \log n).$$

SOLVE AN EXACT RECURSIVE EQUATION

Suppose instead of asymptotic notations in the recursion, we have

$$T(n) = \begin{cases} c & \text{if } n = 1, \\ 2T\left(\frac{n}{2}\right) + cn & \text{otherwise.} \end{cases}$$

Moreover, assume that $n = 2^k$ for some integer $k \geq 0$.

Solving this recursive equation, we get

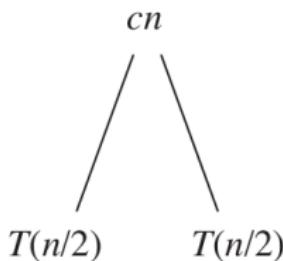
$$T(n) = cn \log_2 n + cn = cn \lg n + cn.$$

The argument uses a .

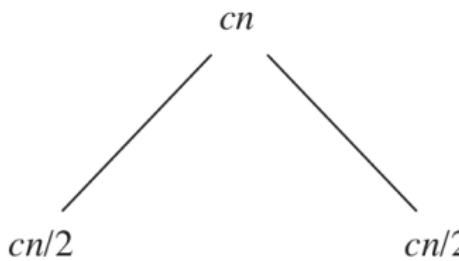
💣 This is not a real proof since the recursion is not exact.

SOLVE AN EXACT RECURSIVE EQUATION BY TREE

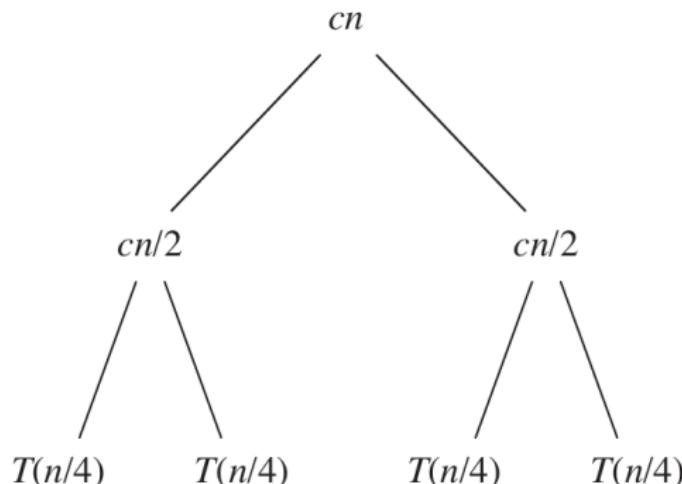
$T(n)$



(a)

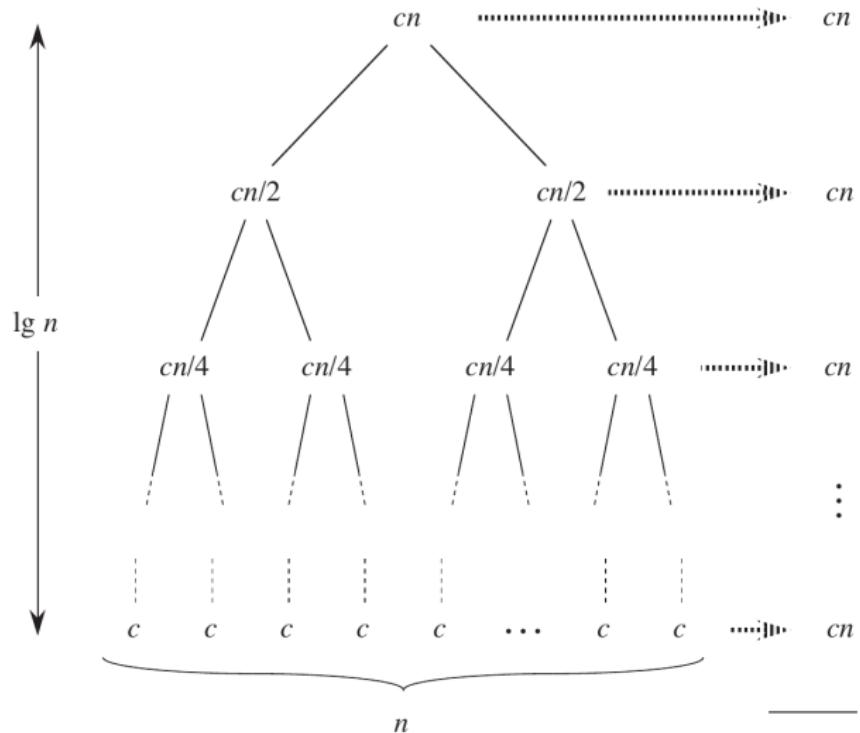


(b)



(c)

SOLVE AN EXACT RECURSIVE EQUATION BY TREE



ITA 4.1 THE MAXIMUM-SUBARRAY PROBLEM

EXAMPLE — STOCK PRICE

The prices of a stock are given in the following plot.

🍰 If you buy once and sell once, when should you do so?



Figure 4: What is the algorithm to maximize 💰!

THE MAXIMUM SUBARRAY PROBLEM

Let $S[0, \dots, n]$ be the stock prices from day 0 to day n . Let

$$A[i] = S[i] - S[i-1], \text{ for } 1 \leq i \leq n.$$

 Then if you buy on day $p - 1$ and sell on day q , the profit is

$$S[q] - S[p-1] = \underline{\hspace{10em}}$$

THE MAXIMUM SUBARRAY PROBLEM

Let $S[0, \dots, n]$ be the stock prices from day 0 to day n . Let

$$A[i] = S[i] - S[i-1], \text{ for } 1 \leq i \leq n.$$

 Then if you buy on day $p-1$ and sell on day q , the profit is

$$S[q] - S[p-1] = \sum_{i=p}^q A[i]$$

Given an array $A[1, \dots, n]$, the **maximum subarray problem** aims to find indices p and q which maximizes

$$\sum_{i=p}^q A[i]$$

THE MAXIMUM SUBARRAY PROBLEM

Given an array $A[1, \dots, n]$, the **maximum subarray problem** aims to find indices p and q which maximizes

$$\sum_{i=p}^q A[i]$$

```
1: Max-Subarray-Brute-Force( $A, n$ )
2: max_profit  $\leftarrow -\infty$ 
3: for  $p \leftarrow 1$  to  $n$  do
4:   sum  $\leftarrow 0$ 
5:   for  $q \leftarrow p$  to  $n$  do
6:     sum  $\leftarrow$  sum +  $A[q]$ 
7:     if sum > max_profit then
8:       max_profit  $\leftarrow$  sum
9:       (buy, sell)  $\leftarrow (p - 1, q)$ 
10: return max_profit
```



This runs in $O(\underline{\hspace{2cm}})$ time.

THE DIVIDE AND CONQUER ALGORITHM



Divide: Split the array into two halves.



Conquer: Recursively find the maximum subarray in each case:

- ⬅ Maximum subarray entirely in the left half.
- ➡ Maximum subarray entirely in the right half.
- ↔ Maximum subarray crossing the midpoint.



Combine: Pick the maximum subarray from each of the three possibilities.



What is the maximum crossing subarray in the following array?

THE DIVIDE AND CONQUER ALGORITHM



What is the maximum crossing subarray in the following array?

Array:	13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Sum to Mid:	<input type="text"/>															
-------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------

THE DIVIDE AND CONQUER ALGORITHM



What is the maximum crossing subarray in the following array?

Array:	13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Sum to Mid:	-19	-32	-29	-4	-24	-21	-5	18	20	13	25	20	-2	13	9	16
-------------	-----	-----	-----	----	-----	-----	----	----	----	----	----	----	----	----	---	----

THE DIVIDE AND CONQUER ALGORITHM



What is the maximum crossing subarray in the following array?

Array:	13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Sum to Mid:	-19	-32	-29	-4	-24	-21	-5	18	20	13	25	20	-2	13	9	16



The maximum crossing subarray contains two parts:

- The maximum subarray entirely in the left half including the midpoint.
- The maximum subarray entirely in the right half including the midpoint.

FIND MAXIMUM CROSSING MIDPOINT SUBARRAY

```
1: Max-Crossing-Subarr( $A, l, m, h$ )
2: left-sum  $\leftarrow -\infty$ 
3: sum  $\leftarrow 0$ 
4: for  $i \leftarrow m$  downto  $l$  do
5:   sum  $\leftarrow$  sum +  $A[i]$ 
6:   if sum > left-sum then
7:     left-sum  $\leftarrow$  sum
8:     max-left  $\leftarrow i$ 
```

FIND MAXIMUM CROSSING MIDPOINT SUBARRAY

```
1: Max-Crossing-Subarr( $A, l, m, h$ )
2: left-sum  $\leftarrow -\infty$ 
3: sum  $\leftarrow 0$ 
4: for  $i \leftarrow m$  downto  $l$  do
5:   sum  $\leftarrow$  sum +  $A[i]$ 
6:   if sum > left-sum then
7:     left-sum  $\leftarrow$  sum
8:     max-left  $\leftarrow i$ 
9: right-sum  $\leftarrow -\infty$ 
10: sum  $\leftarrow 0$ 
11: for  $j \leftarrow m + 1$  to  $h$  do
12:   sum  $\leftarrow$  sum +  $A[j]$ 
13:   if sum > right-sum then
14:     right-sum  $\leftarrow$  sum
15:     max-right  $\leftarrow j$ 
16: return
      (max-left, max-right, left-sum +
       right-sum)
```



What is the time complexity of this procedure?

FIND MAXIMUM SUBARRAY PSEUDOCODE

```
1: Max-Subarr( $A, l, h$ )
2: if  $h == l$  then
3:   return ( $l, h, A[l]$ )
4:  $m \leftarrow \lfloor \frac{l+h}{2} \rfloor$ 
5: ( $ll, lh, ls$ )  $\leftarrow$  MAX-SUBARR( $A, l, m$ )
6: ( $rl, rh, rs$ )  $\leftarrow$  MAX-SUBARR( $A, m + 1, h$ )
7: ( $cl, ch, cs$ )  $\leftarrow$  MAX-CROSSING-SUBARR( $A, l, m, h$ )
8: if  $ls \geq rs$  and  $ls \geq cs$  then
9:   return ( $ll, lh, ls$ )
10: else if  $rs \geq ls$  and  $rs \geq cs$  then
11:   return ( $rl, rh, rs$ )
12: else
13:   return ( $cl, ch, cs$ )
```

TIME COMPLEXITY OF MAXIMUM SUBARRAY

Let $T(n)$ denote the running time of the previous algorithm. Then $T(n)$ satisfies

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ D(n) + 2T\left(\frac{n}{2}\right) + \Theta(n) + C(n) & \text{otherwise.} \end{cases}$$

TIME COMPLEXITY OF MAXIMUM SUBARRAY

Let $T(n)$ denote the running time of the previous algorithm. Then $T(n)$ satisfies

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ D(n) + 2T\left(\frac{n}{2}\right) + \Theta(n) + C(n) & \text{otherwise.} \end{cases}$$

This simplifies to

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

As the recursion is the same as merge sort, the solution is also

$$T(n) = \Theta(n \log n).$$

What is the *sum* of the maximum subarray that crosses the midpoint in the following array?

$$A = \langle 0, 3, 2, 2, -5, 1, -2, 2, 3, -1, -5, -2, 9, -4, -2, -2 \rangle.$$

ITA 4.2 STRASSEN'S ALGORITHM FOR MATRIX MULTIPLICATION

REVIEW: MATRIX MULTIPLICATION



Do you remember how to compute

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

SQUARE MATRIX MULTIPLICATION BY BRUTE FORCE

Let A and B be two $n \times n$ matrices.

Let a_{ij} and b_{ij} be the entries of A and B respectively.

Let $C = A \cdot B$ and c_{ij} be the entries of C .

Then

$$c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$$

This is called the **row-column rule**.

SQUARE MATRIX MULTIPLICATION BY BRUTE FORCE

Let A and B be two $n \times n$ matrices.

Let a_{ij} and b_{ij} be the entries of A and B respectively.

Let $C = A \cdot B$ and c_{ij} be the entries of C .

Then

$$c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$$

This is called the **row-column rule**.

```
1: Square-Matrix-Multiply( $A, B$ )
2:  $n \leftarrow A.\text{rows}$ 
3: let  $C$  be a new  $n \times n$  matrix
4: for  $i = 1$  to  $n$  do
5:   for  $j = 1$  to  $n$  do
6:      $c_{ij} \leftarrow 0$ 
7:     for  $k = 1$  to  $n$  do
8:        $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$ 
9: return  $C$ 
```



What is the time complexity of this algorithm?

MATRIX PARTITIONING

Given $n \times n$ matrices A , B , and $C = AB$, we can partition them

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

where the sub-matrices are of size $n/2 \times n/2$.

MATRIX PARTITIONING

Given $n \times n$ matrices A , B , and $C = AB$, we can partition them

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

where the sub-matrices are of size $n/2 \times n/2$.

For instance, consider

$$A = \begin{pmatrix} 2 & 3 & 0 & 2 \\ 2 & 3 & 0 & 0 \\ 2 & 1 & 2 & 2 \\ 2 & 2 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 3 & 3 & 2 \\ 1 & 0 & 1 & 3 \\ 3 & 1 & 1 & 1 \\ 3 & 3 & 0 & 0 \end{pmatrix}, \quad C = \begin{pmatrix} 15 & 12 & 9 & 13 \\ 9 & 6 & 9 & 13 \\ 19 & 14 & 9 & 9 \\ 17 & 9 & 11 & 13 \end{pmatrix}$$

🎂 Can you identify what A_{11} , ..., C_{22} are?

MATRIX MULTIPLICATION BY PARTITIONING

Then, $C = A \cdot B$ can be written as:

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

This leads to four corresponding equations.

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21},$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22},$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21},$$

$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}.$$

RECURSIVE SQUARE MATRIX MULTIPLICATION ALGORITHM

Let us abbreviate Square-Matrix-Multiply-Recursive as SMMR.

SMMR(A, B)

$n \leftarrow A.\text{rows}$

$C \leftarrow \text{new } n \times n \text{ matrix}$

if $n = 1$ **then**

$C_{11} \leftarrow a_{11} \cdot b_{11}$

else

Partition A, B , and C as in equations (4.9)

$C_{11} \leftarrow \text{SMMR}(A_{11}, B_{11}) + \text{SMMR}(A_{12}, B_{21})$

$C_{12} \leftarrow \text{SMMR}(A_{11}, B_{12}) + \text{SMMR}(A_{12}, B_{22})$

$C_{21} \leftarrow \text{SMMR}(A_{21}, B_{11}) + \text{SMMR}(A_{22}, B_{21})$

$C_{22} \leftarrow \text{SMMR}(A_{21}, B_{12}) + \text{SMMR}(A_{22}, B_{22})$

return C

TIME COMPLEXITY OF RECURSIVE SQUARE MATRIX MULTIPLICATION

Let $T(n)$ denote the running time of SMMR. Then $T(n)$ satisfies

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 8T\left(\frac{n}{2}\right) + D(n) + C(n) & \text{otherwise.} \end{cases}$$

Thus,

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2).$$

In the next lecture, we'll see that this implies $T(n) = \Theta(n^{\lg 8}) = \Theta(n^3)$.

 Nothing has changed!

STRASSEN'S ALGORITHM

Strassen's method improves SMMR by computing

$$M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$M_2 = (A_{21} + A_{22})B_{11}$$

$$M_3 = A_{11}(B_{12} - B_{22})$$

$$M_4 = A_{22}(B_{21} - B_{11})$$

$$M_5 = (A_{11} + A_{12})B_{22}$$

$$M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$

$$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

which reduces the number of matrix-multiplications to 7.

STRASSEN'S ALGORITHM

The resulting product matrix $C = A \cdot B$ is then computed as: You can merge the given expressions into a single equation as follows:

$$C = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}.$$

Thus, the running time satisfies:

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2).$$

In the next lecture, we'll see that this implies

$$T(n) = \Theta(n^{\lg 7}) \approx \Theta(n^{2.81})$$

which is much better than $\Theta(n^3)$!

ONE LOGIC, Two BREAKTHROUGHS

🏆 AlphaGo

- Treated the complex game of Go as a search problem.
- Beat the best human players of Go in 2017.



Figure 5: Demis Hassabis (1976-), co-founder of Google DeepMind.

🏆 AlphaGo

- Treated the complex game of Go as a search problem.
- Beat the best human players of Go in 2017.

12 34 AlphaTensor

- Treats **matrix multiplication** as a game.
- Found algorithms faster than human limits to speed up all AI.



Figure 5: Demis Hassabis (1976-), co-founder of Google DeepMind.

ONE LOGIC, Two BREAKTHROUGHS

🏆 AlphaGo

- Treated the complex game of Go as a search problem.
- Beat the best human players of Go in 2017.

12 34 AlphaTensor

- Treats **matrix multiplication** as a game.
- Found algorithms faster than human limits to speed up all AI.

🧬 AlphaFold

- Treats **protein folding** as a search for the best shape.
- **Result:** 2024 Nobel Prize in Chemistry.



Figure 5: Demis Hassabis (1976-), co-founder of Google DeepMind.

WHAT ARE YOUR MAIN TAKEAWAYS TODAY? ANY QUESTIONS?

