

COL 351 : ANALYSIS & DESIGN OF ALGORITHMS

LECTURE 2

DIVIDE & CONQUER I :

INTEGER MULTIPLICATION AND MERGE SORT

JULY 28, 2026

|

ROHIT VAISH

INTEGER MULTIPLICATION

input: two n digit numbers x and y

output: the product $x \cdot y$

INTEGER MULTIPLICATION

input: two n digit numbers x and y

output: the product $x \cdot y$

e.g.,

5	6	7	8
1	2	3	4

Grade-school algorithm

INTEGER MULTIPLICATION

input: two n digit numbers x and y

output: the product $x \cdot y$

e.g.,

$$\begin{array}{r} 5678 \\ 1234 \\ \hline 22712 \end{array}$$

Grade-school algorithm

INTEGER MULTIPLICATION

input: two n digit numbers x and y

output: the product $x \cdot y$

eg.,

5	6	7	8	
1	2	3	4	
2	2	7	1	2
1	7	0	3	4

Grade-school algorithm

INTEGER MULTIPLICATION

input: two n digit numbers x and y

output: the product $x \cdot y$

eg.,

	5	6	7	8	
	1	2	3	4	
	2	2	7	1	2
	1	7	0	3	4
	1	1	3	5	6

Grade-school algorithm

INTEGER MULTIPLICATION

input: two n digit numbers x and y

output: the product $x \cdot y$

eg.,

		5	6	7	8	
		1	2	3	4	
		2	2	7	1	2
	1	7	0	3	4	
1	1	3	5	6		
5	6	7	8			

Grade-school algorithm

INTEGER MULTIPLICATION

input: two n digit numbers x and y

output: the product $x \cdot y$

e.g.,

5 6 7 8

1 2 3 4

2 2 7 1 2

1 7 0 3 4

1 1 3 5 6

5 6 7 8

7 0 0 6 6 5 2

Grade-school algorithm

INTEGER MULTIPLICATION

input: two n digit numbers x and y

output: the product $x \cdot y$

e.g.,

	5	6	7	8	
	1	2	3	4	
	2	2	7	1	2
	1	7	0	3	4
1	1	3	5	6	
5	6	7	8		
7	0	0	6	6	5 2

basic operations: add or multiply
two single-digit numbers

INTEGER MULTIPLICATION

input: two n digit numbers x and y

output: the product $x \cdot y$

e.g.,

		5	6	7	8	
		1	2	3	4	
		2	2	7	1	2
		1	7	0	3	4
	1	1	3	5	6	
	5	6	7	8		
	7	0	0	6	6	5 2

basic operations: add or multiply
two single-digit numbers

operations $\leq \underbrace{\text{constant} \cdot n^2}_{\leq 9}$

INTEGER MULTIPLICATION

input: two n digit numbers x and y

output: the product $x \cdot y$

e.g.,

		5	6	7	8	
		1	2	3	4	
		2	2	7	1	2
	1	7	0	3	4	
1	1	3	5	6		
5	6	7	8			
7	0	0	6	6	5	2

basic operations: add or multiply
two single-digit numbers

$$\# \text{ operations} \leq \underbrace{\text{constant}}_{\leq 9} \cdot n^2$$

Can we do better?

KARATSUBA MULTIPLICATION

KARATSUBA MULTIPLICATION

$$x = 5 \quad 6 \quad 7 \quad 8$$

$$y = 1 \quad 2 \quad 3 \quad 4$$

KARATSUBA MULTIPLICATION

$$\begin{array}{rcl} x = & \overset{a}{\textcircled{5 \quad 6}} & \overset{b}{\textcircled{7 \quad 8}} \\ y = & \underset{c}{\textcircled{1 \quad 2}} & \underset{d}{\textcircled{3 \quad 4}} \end{array}$$

KARATSUBA MULTIPLICATION

① compute $a \cdot c = 56 \times 12 = 672$

$$\begin{array}{cc} & a & & b \\ x = & \boxed{5} & \boxed{6} & \boxed{7} & \boxed{8} \\ & & & & \\ y = & \boxed{1} & \boxed{2} & \boxed{3} & \boxed{4} \\ & c & & d \end{array}$$

KARATSUBA MULTIPLICATION

① compute $a \cdot c = 56 \times 12 = 672$

② compute $b \cdot d = 78 \times 34 = 2652$

$$\begin{array}{cc} & a & & b \\ x = & \boxed{5} & \boxed{6} & \boxed{7} & \boxed{8} \\ y = & \boxed{1} & \boxed{2} & \boxed{3} & \boxed{4} \\ & c & & d \end{array}$$

KARATSUBA MULTIPLICATION

① compute $a \cdot c = 56 \times 12 = 672$

② compute $b \cdot d = 78 \times 34 = 2652$

③ compute $(a+b) \cdot (c+d) = 134 \times 46 = 6164$

$$\begin{array}{cc} & a & & b \\ x = & \boxed{5} & \boxed{6} & \boxed{7} & \boxed{8} \\ & & & & \\ y = & \boxed{1} & \boxed{2} & \boxed{3} & \boxed{4} \\ & c & & d \end{array}$$

KARATSUBA MULTIPLICATION

① compute $a \cdot c = 56 \times 12 = 672$

② compute $b \cdot d = 78 \times 34 = 2652$

③ compute $(a+b) \cdot (c+d) = 134 \times 46 = 6164$

④ compute ③ - ② - ① = 2840

$$\begin{array}{cc} & a & & b \\ x = & \boxed{5} & \boxed{6} & \boxed{7} & \boxed{8} \\ & & & & \\ y = & \boxed{1} & \boxed{2} & \boxed{3} & \boxed{4} \\ & c & & d \end{array}$$

KARATSUBA MULTIPLICATION

① compute $a \cdot c = 56 \times 12 = 672$

② compute $b \cdot d = 78 \times 34 = 2652$

③ compute $(a+b) \cdot (c+d) = 134 \times 46 = 6164$

④ compute $\textcircled{3} - \textcircled{2} - \textcircled{1} = 2840$

⑤ Compute

672	0000	from ①
2652		from ②
2840	00	from ④

$x = \begin{matrix} & a & \\ \textcircled{5} & 6 & \\ & c & \end{matrix} \quad \begin{matrix} & b & \\ \textcircled{7} & 8 & \\ & d & \end{matrix}$

$y = \begin{matrix} & & \\ \textcircled{1} & 2 & \\ & & \end{matrix} \quad \begin{matrix} & & \\ \textcircled{3} & 4 & \\ & & \end{matrix}$

KARATSUBA MULTIPLICATION

① compute $a \cdot c = 56 \times 12 = 672$

② compute $b \cdot d = 78 \times 34 = 2652$

③ compute $(a+b) \cdot (c+d) = 134 \times 46 = 6164$

④ compute ③ - ② - ① = 2840

⑤ Compute

672	0000	from ①
2652		from ②
2840	00	from ④
7006652		= $x \cdot y$

$x = \begin{matrix} & a & \\ \text{5} & 6 & \\ & c & \end{matrix} \quad \begin{matrix} & b & \\ \text{7} & 8 & \\ & d & \end{matrix}$

$y = \begin{matrix} \text{1} & 2 & \\ & c & \end{matrix} \quad \begin{matrix} \text{3} & 4 & \\ & d & \end{matrix}$

A RECURSIVE ALGORITHM

A RECURSIVE ALGORITHM

input: two n digit numbers x and y

output: the product $x \cdot y$

A RECURSIVE ALGORITHM

input: two n digit numbers x and y

output: the product $x \cdot y$

Write $x = 10^{n/2} \cdot a + b$ and $y = 10^{n/2} \cdot c + d$

assuming n is even

A RECURSIVE ALGORITHM

input: two n digit numbers x and y

output: the product $x \cdot y$

Write $x = 10^{n/2} \cdot a + b$ and $y = 10^{n/2} \cdot c + d$

assuming n is even

NOTE: a, b, c, d are $n/2$ digit numbers

e.g., $x = 5678$

$y = 1234$

$a = 56$

$c = 12$

$b = 78$

$d = 34$

A RECURSIVE ALGORITHM

input: two n digit numbers x and y

output: the product $x \cdot y$

Write $x = 10^{n/2} \cdot a + b$ and $y = 10^{n/2} \cdot c + d$

A RECURSIVE ALGORITHM

input: two n digit numbers x and y

output: the product $x \cdot y$

Write $x = 10^{n/2} \cdot a + b$ and $y = 10^{n/2} \cdot c + d$

$$x \cdot y = (10^{n/2} a + b) \cdot (10^{n/2} c + d)$$

A RECURSIVE ALGORITHM

input: two n digit numbers x and y

output: the product $x \cdot y$

Write $x = 10^{n/2} \cdot a + b$ and $y = 10^{n/2} \cdot c + d$

$$x \cdot y = (10^{n/2} a + b) \cdot (10^{n/2} c + d)$$

$$= 10^n ac + 10^{n/2} (ad + bc) + bd$$

A RECURSIVE ALGORITHM

input: two n digit numbers x and y

output: the product $x \cdot y$

Write $x = 10^{n/2} \cdot a + b$ and $y = 10^{n/2} \cdot c + d$

$$x \cdot y = (10^{n/2} a + b) \cdot (10^{n/2} c + d)$$

$$= 10^n ac + 10^{n/2} (ad + bc) + bd$$

involves only the product of smaller numbers

A RECURSIVE ALGORITHM

input: two n digit numbers x and y

output: the product $x \cdot y$

Write $x = 10^{n/2} \cdot a + b$ and $y = 10^{n/2} \cdot c + d$

$$x \cdot y = (10^{n/2} a + b) \cdot (10^{n/2} c + d)$$

$$= 10^n ac + 10^{n/2} (ad + bc) + bd$$



recursively solve smaller problems and combine the solutions

A RECURSIVE ALGORITHM

input: two n digit numbers x and y

output: the product $x \cdot y$

Write $x = 10^{n/2} \cdot a + b$ and $y = 10^{n/2} \cdot c + d$

$$x \cdot y = (10^{n/2} a + b) \cdot (10^{n/2} c + d)$$

$$= 10^n ac + 10^{n/2} (ad + bc) + bd$$

 **recursively** solve smaller problems and combine the solutions

Base case? Easy! When $n=1$, compute $x \cdot y$ in one step.

The Good, The Bad and The Ugly of Writing Algorithms

The Good

The Bad

The Ugly

The Good

The Bad

The Ugly

The recursive algorithm
takes as input two n -digit
numbers x and y and
returns their product $x \cdot y$.

The base case of the algorithm
involves checking whether
the numbers are single-digit.

If so, the algorithm directly
returns their product. Otherwise,
the algorithm computes ...

⋮

The Good

The recursive algorithm takes as input two n -digit numbers x and y and returns their product $x \cdot y$.

The base case of the algorithm involves checking whether the numbers are single-digit.

If so, the algorithm directly returns their product. Otherwise, the algorithm computes ...

⋮

The Bad

The Ugly

```
def recursive(x, y)
```

```
    if len(str(x)) == 1
```

```
    or len(str(y)) == 1:
```

```
        return x * y
```

```
    m = max(len(str(x)),  
            len(str(y)))
```

```
    m2 = m // 2
```

```
    a = x // 10 ** (m2)
```

```
    b = x % 10 ** (m2)
```

```
    ⋮
```


The Good

input: two n digit numbers x, y

output: the product $x \cdot y$

if $n=1$ **return** $x \cdot y$

else

$a, b :=$ first and second halves of x

$c, d :=$ " " " y

recursively compute $a \cdot c$, $a \cdot d$,
 $b \cdot c$, and $b \cdot d$

Compute $10^n a \cdot c + 10^{\frac{n}{2}} (a \cdot d + b \cdot c)$
 $+ b \cdot d$

using grade-school addition
and **return** the result

The Bad

The recursive algorithm
takes as input two n -digit
numbers x and y and
returns their product $x \cdot y$.

The base case of the algorithm
involves checking whether
the numbers are single-digit.

If so, the algorithm directly
returns their product. Otherwise,
the algorithm computes ...

⋮

The Ugly

```
def recursive(x, y)
```

```
    if len(str(x)) == 1
```

```
        or len(str(y)) == 1:
```

```
            return x * y
```

```
    m = max(len(str(x)),  
            len(str(y)))
```

```
    m2 = m // 2
```

```
    a = x // 10 ** (m2)
```

```
    b = x % 10 ** (m2)
```

```
    ⋮
```

The Good

input: two n digit numbers x, y

output: the product $x \cdot y$

if $n=1$ return $x \cdot y$

else

$a, b :=$ first and second halves of x

$c, d :=$ " " " y

recursively compute $a \cdot c$, $a \cdot d$,
 $b \cdot c$, and $b \cdot d$

Compute $10^n \cdot ac + 10^{n/2} (ad + b \cdot c)$
 $+ bd$

using grade-school addition
and return the result

How many basic operations does the recursive algorithm perform?

The Good

input: two n digit numbers x, y

output: the product $x \cdot y$

if $n=1$ return $x \cdot y$

else

$a, b :=$ first and second halves of x

$c, d :=$ " " " y

recursively compute $a \cdot c$, $a \cdot d$,
 $b \cdot c$, and $b \cdot d$

Compute $10^n \cdot ac + 10^{n/2} (ad + b \cdot c)$
 $+ bd$

using grade-school addition
and return the result

How many basic operations does the recursive algorithm perform?

Coming up shortly!

The Good

input: two n digit numbers x, y

output: the product $x \cdot y$

if $n=1$ return $x \cdot y$

else

$a, b :=$ first and second halves of x

$c, d :=$ " " " y

recursively compute $a \cdot c$, $a \cdot d$,
 $b \cdot c$, and $b \cdot d$

Compute $10^n \cdot ac + 10^{n/2} (ad + b \cdot c)$
 $+ bd$

using grade-school addition
and return the result

How many basic operations does the recursive algorithm perform?

Coming up shortly!

First, let's demystify Karatsuba

KARATSUBA MULTIPLICATION

① compute $a \cdot c = 56 \times 12 = 672$

② compute $b \cdot d = 78 \times 34 = 2652$

③ compute $(a+b) \cdot (c+d) = 134 \times 46 = 6164$

④ compute ③ - ② - ① = 2840

⑤ Compute

672	0000	from ①
2652		from ②
2840	00	from ④
7006652		= $x \cdot y$

$x = \begin{matrix} & a & \\ \text{5} & 6 & \\ & c & \end{matrix} \quad \begin{matrix} & b & \\ \text{7} & 8 & \\ & d & \end{matrix}$

$y = \begin{matrix} \text{1} & 2 & \\ & c & \end{matrix} \quad \begin{matrix} \text{3} & 4 & \\ & d & \end{matrix}$

KARATSUBA MULTIPLICATION

Recall from recursive algorithm :

$$x \cdot y = 10^n ac + 10^{n/2} (ad + bc) + bd$$

$$\begin{array}{rcl} x = & \overset{a}{\text{5}} \ \overset{b}{\text{6}} & \\ y = & \underset{c}{\text{1}} \ \underset{d}{\text{2}} & \end{array}$$

KARATSUBA MULTIPLICATION

Recall from recursive algorithm :

$$x \cdot y = 10^n ac + 10^{n/2} (ad + bc) + bd$$

$$\begin{array}{rcl} & a & b \\ x = & \boxed{5 \quad 6} & \boxed{7 \quad 8} \\ & c & d \\ y = & \boxed{1 \quad 2} & \boxed{3 \quad 4} \end{array}$$

Recursive algo. needs **four** recursive calls

KARATSUBA MULTIPLICATION

Recall from recursive algorithm :

$$x \cdot y = 10^n ac + 10^{n/2} (ad + bc) + bd$$

$$\begin{array}{rcl} & a & b \\ x = & \boxed{5 \quad 6} & \boxed{7 \quad 8} \\ & c & d \\ y = & \boxed{1 \quad 2} & \boxed{3 \quad 4} \end{array}$$

Recursive algo. needs **four** recursive calls

Can we get away with only **three** recursive calls?

KARATSUBA MULTIPLICATION

Recall from recursive algorithm :

$$x \cdot y = 10^n ac + 10^{n/2} (ad + bc) + bd$$

$$\begin{array}{cc} & a & & b \\ x = & \boxed{5} & \boxed{6} & \boxed{7} & \boxed{8} \\ & c & & d \\ y = & \boxed{1} & \boxed{2} & \boxed{3} & \boxed{4} \\ & & & & d \end{array}$$

Recursive algo. needs **four** recursive calls

Can we get away with only **three** recursive calls?

YES!

KARATSUBA MULTIPLICATION

Recall from recursive algorithm :

$$x \cdot y = 10^n ac + 10^{n/2} (ad + bc) + bd$$

$$\begin{array}{cc} & a \\ x = & \begin{pmatrix} 5 & 6 \end{pmatrix} \end{array} \begin{array}{cc} & b \\ y = & \begin{pmatrix} 7 & 8 \end{pmatrix} \\ & \begin{pmatrix} 1 & 2 \end{pmatrix} \\ & c \end{array} \begin{array}{cc} & b \\ & \begin{pmatrix} 3 & 4 \end{pmatrix} \\ & d \end{array}$$

we only care about the sum of ad and bc

KARATSUBA MULTIPLICATION

Recall from recursive algorithm :

$$x \cdot y = 10^n ac + 10^{n/2} (ad + bc) + bd$$

$$\begin{array}{rcl} & a & b \\ x = & \boxed{5 \quad 6} & \boxed{7 \quad 8} \\ & c & d \\ y = & \boxed{1 \quad 2} & \boxed{3 \quad 4} \end{array}$$

we only care about the sum of ad and bc



$$ad + bc = (a + b)(c + d) - ac - bd$$

Ganes'
trick

KARATSUBA MULTIPLICATION

Recall from recursive algorithm :

$$x \cdot y = 10^n ac + 10^{n/2} (ad + bc) + bd$$

$$\begin{array}{cc} & a & & b \\ x = & \boxed{5} & \boxed{6} & \boxed{7} & \boxed{8} \\ & & & & \\ y = & \boxed{1} & \boxed{2} & \boxed{3} & \boxed{4} \\ & & c & & d \end{array}$$

we only care about the sum of ad and bc



Ganes'
trick

$$\underbrace{ad + bc}_{\text{two recursive calls}} = \underbrace{(a+b)(c+d)}_{\text{one recursive call}} - ac - bd$$

already have these

KARATSUBA MULTIPLICATION

Recall from recursive algorithm :

$$x \cdot y = 10^n ac + 10^{n/2} (ad + bc) + bd$$

$$\begin{array}{cc} a & b \\ \hline x = & \begin{array}{|c|c|} \hline 5 & 6 \\ \hline \end{array} \begin{array}{|c|c|} \hline 7 & 8 \\ \hline \end{array} \\ y = & \begin{array}{|c|c|} \hline 1 & 2 \\ \hline \end{array} \begin{array}{|c|c|} \hline 3 & 4 \\ \hline \end{array} \\ c & d \end{array}$$

we only care about the **sum** of **ad** and **bc**



$$\underbrace{ad}_{\text{two recursive calls}} + \underbrace{bc}_{\text{two recursive calls}} = \underbrace{(a+b)(c+d)}_{\text{one recursive call}} - \underbrace{ac}_{\text{already have these}} - \underbrace{bd}_{\text{already have these}}$$

Ganes'
trick

Karatsuba only needs **three** recursive calls!
(and some additions)

KARATSUBA MULTIPLICATION

① compute $a \cdot c = 56 \times 12 = 672$

② compute $b \cdot d = 78 \times 34 = 2652$

③ compute $(a+b) \cdot (c+d) = 134 \times 46 = 6164$

④ compute ③ - ② - ① = 2840

⑤ Compute

672	0000	from ①
2652		from ②
2840	00	from ④
7006652		= $x \cdot y$

$x = \begin{matrix} & a & \\ \text{5} & 6 & \\ & c & \end{matrix} \quad \begin{matrix} & b & \\ \text{7} & 8 & \\ & d & \end{matrix}$

$y = \begin{matrix} \text{1} & 2 & \\ & c & \end{matrix} \quad \begin{matrix} \text{3} & 4 & \\ & d & \end{matrix}$

KARATSUBA MULTIPLICATION

① compute $a \cdot c = 56 \times 12 = 672$

② compute $b \cdot d = 78 \times 34 = 2652$

③ compute $(a+b) \cdot (c+d) = 134 \times 46 = 6164$

④ compute $\textcircled{3} - \textcircled{2} - \textcircled{1} = 2840$

$$\begin{array}{cc} a & b \\ \hline 5 & 6 \\ \hline 1 & 2 \\ c & d \end{array}$$

$x =$ (5 6) (7 8)
 $y =$ (1 2) (3 4)

Gauss' trick!

⑤ Compute

$$\begin{array}{r} 6720000 \text{ from } \textcircled{1} \\ 2652 \text{ from } \textcircled{2} \\ 284000 \text{ from } \textcircled{4} \\ \hline 7006652 = x \cdot y \end{array}$$

KARATSUBA PSEUDOCODE

input: two n -digit positive integers x and y

output: product $x \cdot y$

KARATSUBA PSEUDOCODE

input: two n -digit positive integers x and y

output: product $x \cdot y$

if $n=1$

 Compute $x \cdot y$ in one step and return the result

KARATSUBA PSEUDOCODE

input: two n -digit positive integers x and y

output: product $x \cdot y$

if $n=1$

 Compute $x \cdot y$ in one step and return the result

else

$a, b :=$ first and second halves of x

$c, d :=$ " " " y

KARATSUBA PSEUDOCODE

input: two n -digit positive integers x and y

output: product $x \cdot y$

if $n=1$

 Compute $x \cdot y$ in one step and return the result

else

$a, b :=$ first and second halves of x

$c, d :=$ " " " y

 recursively compute $a \cdot c$, $b \cdot d$, and $(a+b) \cdot (c+d)$

KARATSUBA PSEUDOCODE

input: two n -digit positive integers x and y

output: product $x \cdot y$

if $n=1$

 Compute $x \cdot y$ in one step and return the result

else

$a, b :=$ first and second halves of x

$c, d :=$ " " " y

 recursively compute $a \cdot c$, $b \cdot d$, and $(a+b) \cdot (c+d)$

 compute $z := (a+b) \cdot (c+d) - ac - bd$ via grade-school addition

KARATSUBA PSEUDOCODE

input: two n -digit positive integers x and y

output: product $x \cdot y$

if $n=1$

 Compute $x \cdot y$ in one step and return the result

else

$a, b :=$ first and second halves of x

$c, d :=$ " " " y

 recursively compute $a \cdot c$, $b \cdot d$, and $(a+b) \cdot (c+d)$

 compute $z := (a+b) \cdot (c+d) - ac - bd$ via grade-school addition

 return $10^n \cdot a \cdot c + 10^{n/2} \cdot z + bd$

STORY SO FAR

Grade-school multiplication

$\leq 9n^2$ basic operations

Recursive algorithm (4 calls)

?

karatsuba algorithm (3 calls)

?

STORY SO FAR

Grade-school multiplication

$\leq 9n^2$ basic operations

Recursive algorithm (4 calls)

?

karatsuba algorithm (3 calls)

?

Is karatsuba faster than recursive and grade-school algorithms?

STORY SO FAR

Grade-school multiplication

$\leq 9n^2$ basic operations

Recursive algorithm (4 calls)

?

Karatsuba algorithm (3 calls)

?

Is Karatsuba faster than recursive and grade-school algorithms?

We will come back to this question.

MERGE SORT

MERGE SORT

Canonical divide-and-conquer algorithm

- break the problem into smaller subproblems
- solve the subproblems recursively
- combine the solutions of subproblems

MERGE SORT

Canonical divide-and-conquer algorithm

- break the problem into smaller subproblems
- solve the subproblems recursively
- combine the solutions of subproblems

Warm up for recursion tree and master method

- useful in analyzing recursive multiplication and Karatsuba's algorithms

SORTING

input : an array of numbers

eg.,

2	6	3	8	7	1	4	5
---	---	---	---	---	---	---	---

output : Same numbers , sorted in increasing order

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

SORTING

input: an array of numbers

eg,

2	6	3	8	7	1	4	5
---	---	---	---	---	---	---	---

output: same numbers, sorted in increasing order

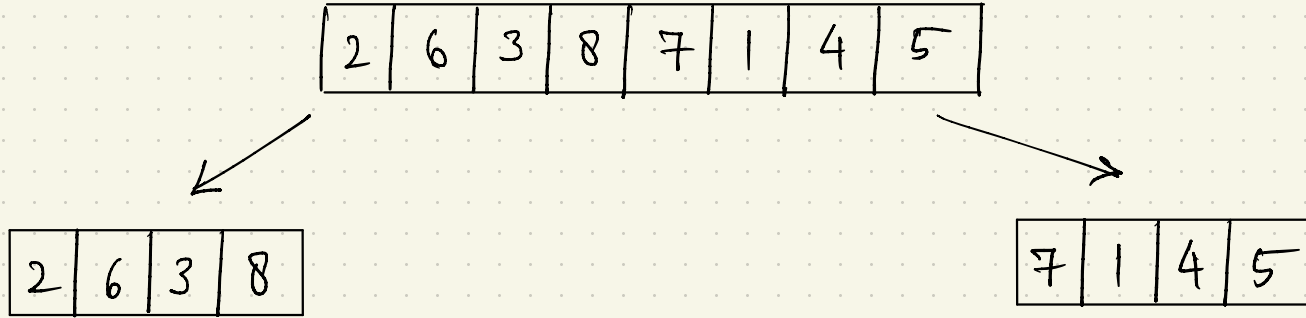
1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

for simplicity, assume numbers to be distinct

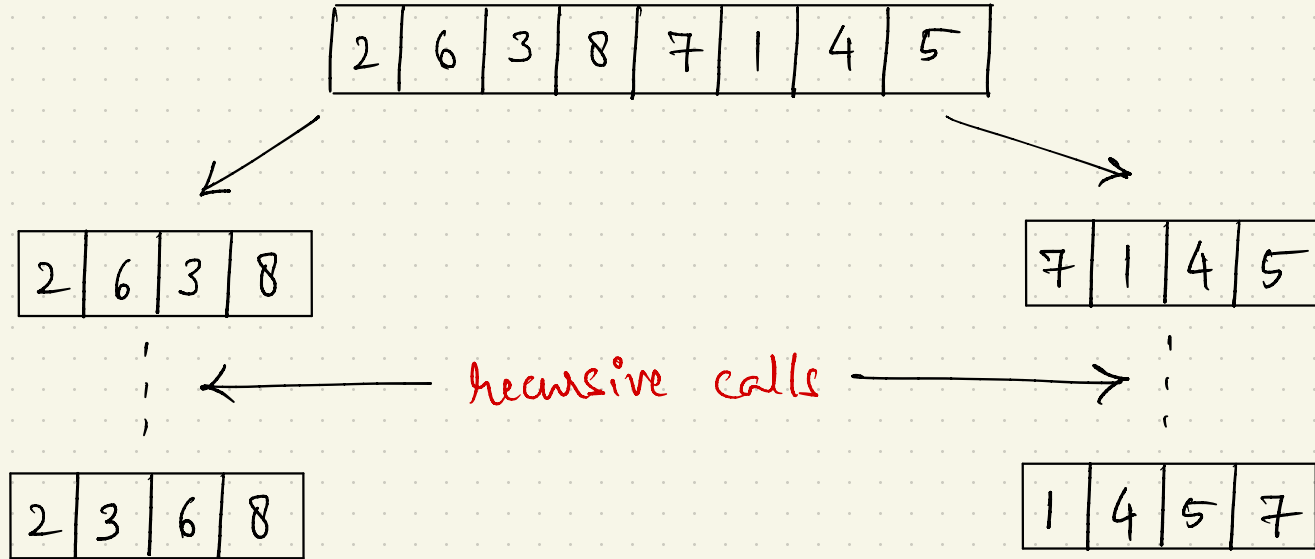
MERGE SORT : EXAMPLE

2	6	3	8	7	1	4	5
---	---	---	---	---	---	---	---

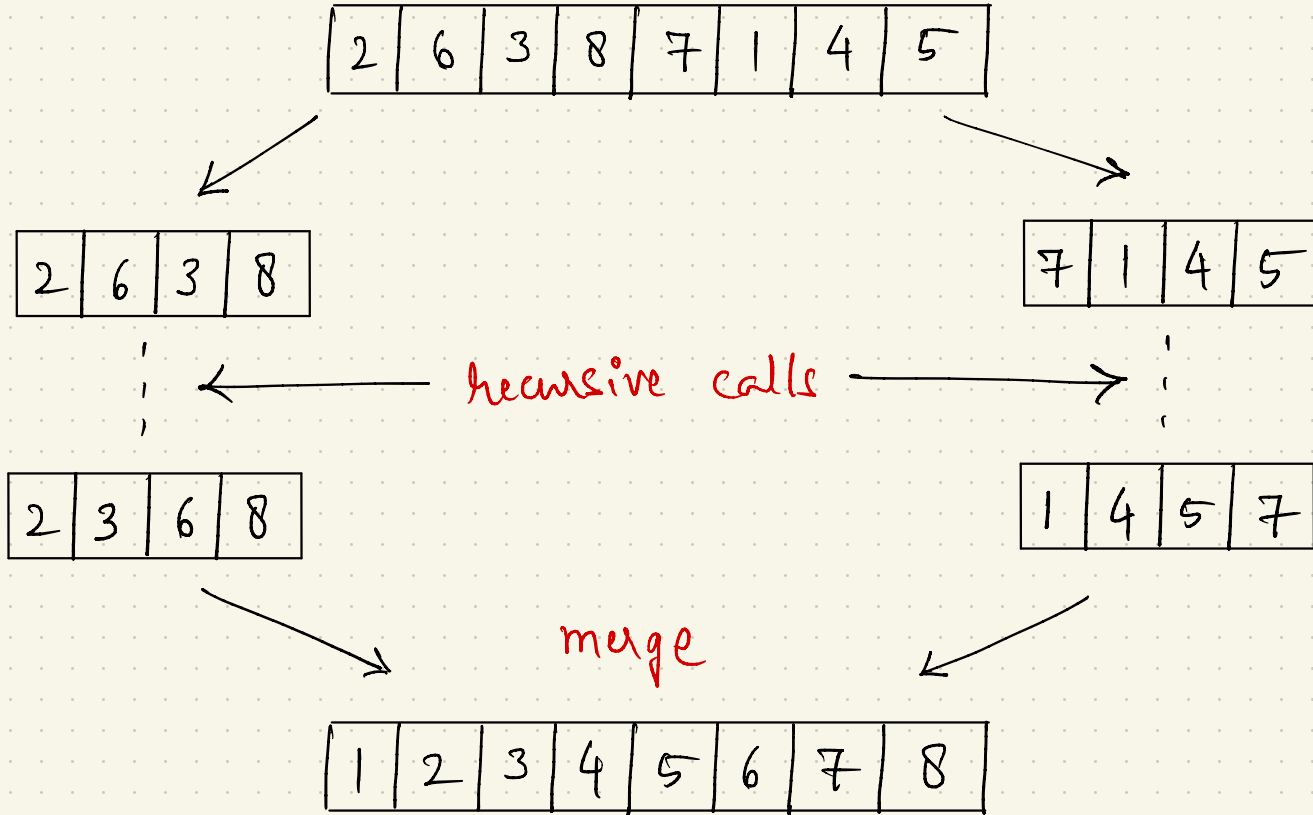
MERGE SORT : EXAMPLE



MERGE SORT : EXAMPLE



MERGE SORT : EXAMPLE



MERGE SORT : PSEUDOCODE

input : array A of n distinct integers

output : array with the same numbers sorted in increasing order

MERGE SORT : PSEUDOCODE

input : array A of n distinct integers

output : array with the same numbers sorted in increasing order

if $n \leq 1$

return A

else

$L :=$ left half of A sorted recursively

$R :=$ right " " " "

return $\text{MERGE}(L, R)$

MERGE SUBROUTINE

input: sorted arrays L and R of length $n/2$ each

output: sorted array B of length n

MERGE SUBROUTINE

input: sorted arrays L and R of length $n/2$ each

output: sorted array B of length n

assume n is even

MERGE SUBROUTINE

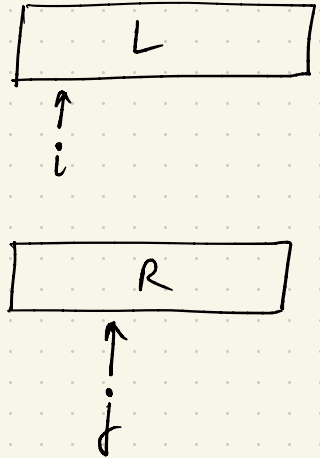
input: sorted arrays L and R of length $n/2$ each

output: sorted array B of length n

initialize $i := 1$

initialize $j := 1$

assume n is even



MERGE SUBROUTINE

input: sorted arrays L and R of length $n/2$ each

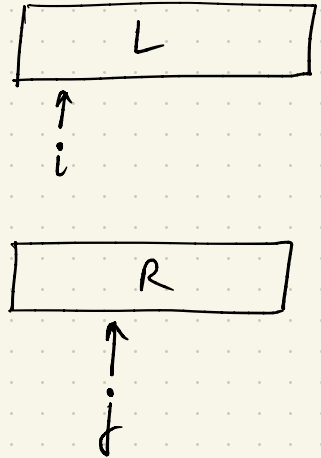
output: sorted array B of length n

initialize $i := 1$

initialize $j := 1$

for $k := 1$ to n

assume n is even



MERGE SUBROUTINE

input: sorted arrays L and R of length $n/2$ each

output: sorted array B of length n

assume n is even

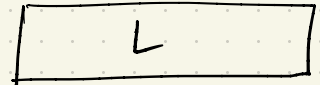
initialize $i := 1$

initialize $j := 1$

for $k := 1$ to n

if $L[i] < R[j]$ then

else



MERGE SUBROUTINE

input: sorted arrays L and R of length $n/2$ each

output: sorted array B of length n

assume n is even

initialize $i := 1$

initialize $j := 1$

for $k := 1$ to n

if $L[i] < R[j]$ then

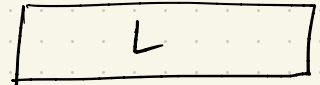
$B[k] := L[i]$

increment i by 1

else

$B[k] := R[j]$

increment j by 1



MERGE SUBROUTINE

input: sorted arrays L and R of length $n/2$ each

output: sorted array B of length n

assume n is even

initialize $i := 1$

initialize $j := 1$

for $k := 1$ to n

if $L[i] < R[j]$ then

$B[k] := L[i]$

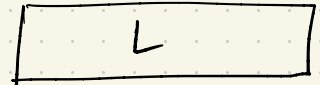
increment i by 1

else

$B[k] := R[j]$

increment j by 1

ignoring end cases
when i or j falls off



MERGE SORT RUNNING TIME

What is the basic operation?

MERGE SORT RUNNING TIME

What is the **basic operation**?

- commonly, any pairwise comparison
- for now, a line-by-line implementation using a debugger

MERGE SORT RUNNING TIME

What is the **basic operation**?

- commonly, any pairwise comparison
- for now, a line-by-line implementation using a debugger

Let's start by analyzing MERGE subroutine.

MERGE SUBROUTINE

input: sorted arrays L and R of length $n/2$ each

output: sorted array B of length n

assume n is even

initialize $i := 1$

initialize $j := 1$

for $k := 1$ to n

if $L[i] < R[j]$ then

$B[k] := L[i]$

increment i by 1

else

$B[k] := R[j]$

increment j by 1

MERGE SUBROUTINE

input: sorted arrays L and R of length $n/2$ each

output: sorted array B of length n

assume n is even

initialize $i := 1$
initialize $j := 1$ } 2 operations

for $k := 1$ to n

if $L[i] < R[j]$ then

$B[k] := L[i]$

increment i by 1

else

$B[k] := R[j]$

increment j by 1

MERGE SUBROUTINE

input: sorted arrays L and R of length $n/2$ each

output: sorted array B of length n

assume n is even

initialize $i := 1$

initialize $j := 1$

} 2 operations

for $k := 1$ to n

executes n times

if $L[i] < R[j]$ then

$B[k] := L[i]$

increment i by 1

else

$B[k] := R[j]$

increment j by 1

MERGE SUBROUTINE

input: sorted arrays L and R of length $n/2$ each

output: sorted array B of length n

assume n is even

initialize $i := 1$

initialize $j := 1$

} 2 operations

for $k := 1$ to n

executes n times

if $L[i] < R[j]$ then

$B[k] := L[i]$

increment i by 1

} ?? operations

else

$B[k] := R[j]$

increment j by 1

MERGE SUBROUTINE

input: sorted arrays L and R of length $n/2$ each

output: sorted array B of length n

assume n is even

initialize $i := 1$

initialize $j := 1$

} 2 operations

for $k := 1$ to n

executes n times

if $L[i] < R[j]$ then

$B[k] := L[i]$

increment i by 1

} $3 + 1 = 4$ operations
 — incrementing k

else

$B[k] := R[j]$

increment j by 1

MERGE SUBROUTINE

input: sorted arrays L and R of length $n/2$ each

output: sorted array B of length n

assume n is even

initialize $i := 1$

initialize $j := 1$

} 2 operations

for $k := 1$ to n

executes n times

if $L[i] < R[j]$ then

$B[k] := L[i]$

increment i by 1

} $3 + 1 = 4$ operations
└ incrementing k

else

$B[k] := R[j]$

increment j by 1

} 4 operations

MERGE SUBROUTINE

input: sorted arrays L and R of length $n/2$ each

output: sorted array B of length n

initialize $i := 1$

initialize $j := 1$

} 2 operations

for $k := 1$ to n

executes n times

if $L[i] < R[j]$ then

$B[k] := L[i]$

increment i by 1

else

$B[k] := R[j]$

increment j by 1

assume n is even

$4n + 2$ operations

$3 + 1 = 4$ operations
└ incrementing k

4 operations

MERGE SUBROUTINE

Lemma: Running time of MERGE subroutine on input arrays of length $n/2$ each is $\leq 4n + 2$.

MERGE SUBROUTINE

Lemma: Running time of MERGE subroutine on input arrays of length $n/2$ each is $\leq 4n + 2$.

$\leq 6n$ since $n \geq 1$

MERGE SUBROUTINE

Lemma: Running time of MERGE subroutine on input arrays of length $n/2$ each is $\leq 4n + 2$.

$\leq 6n$ since $n \geq 1$

MERGE SUBROUTINE

Lemma: Running time of MERGE subroutine on input arrays of length $l/2$ each is $\leq 4l + 2$.

$$\leq 6l \quad \text{since } l \geq 1$$

MERGE SUBROUTINE

Lemma: Running time of MERGE subroutine on input arrays of length $l/2$ each is $\leq 4l + 2$.

$$\leq 6l \quad \text{since } l \geq 1$$

(So we can plug in other values of l later)

MERGE SORT RUNNING TIME

MERGE SORT RUNNING TIME

