Sorting Algorithms

July 17th, 2021 (Class #2)

A quick refresher...

- What is an algorithm?

A sequence of simple steps to solve a problem.

A quick refresher...

- How do we measure how fast an algorithm is?

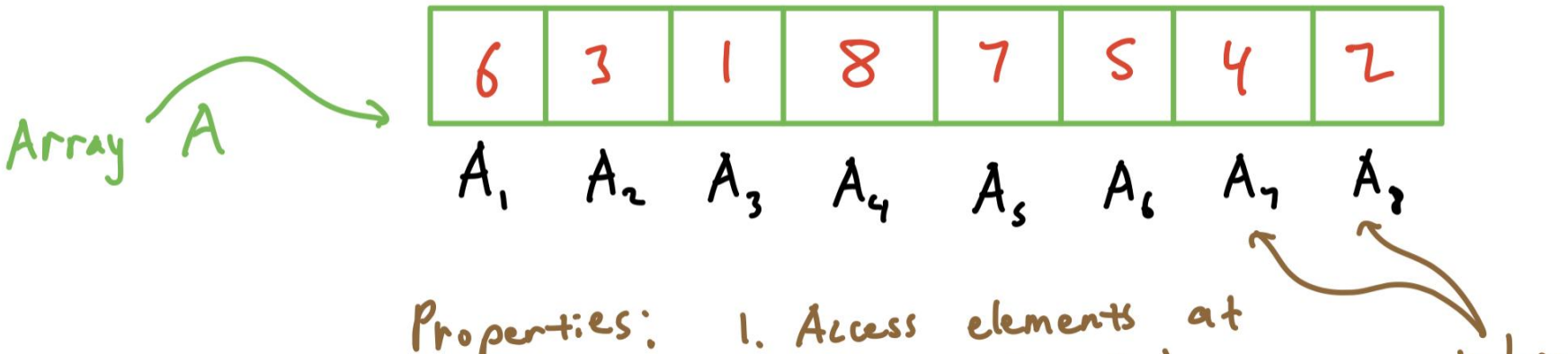
$T(n)$: The number of simple steps needed in the worst case.

We cared about how $T(n)$ scaled with n.

$O(2^n)$? $O(n^2)$? $O(n \log n)$? $O(n)$?

Array

Think "list of elements"

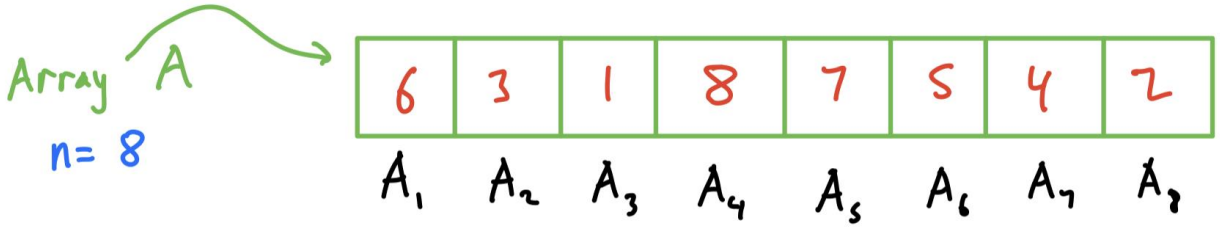


Properties:

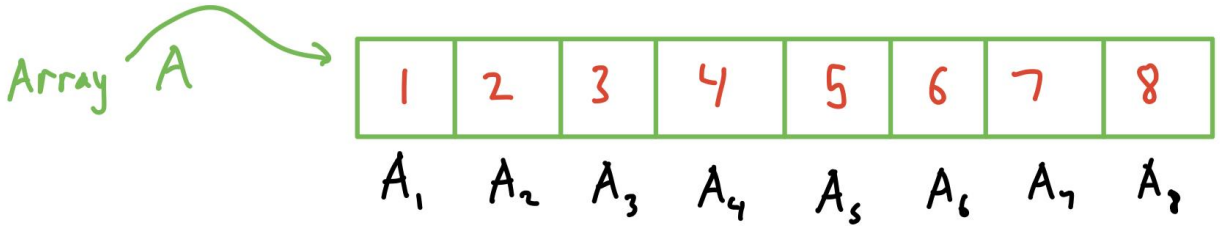
1. Access elements at a specific index
2. Write values to specific positions

Array Sorting Problem

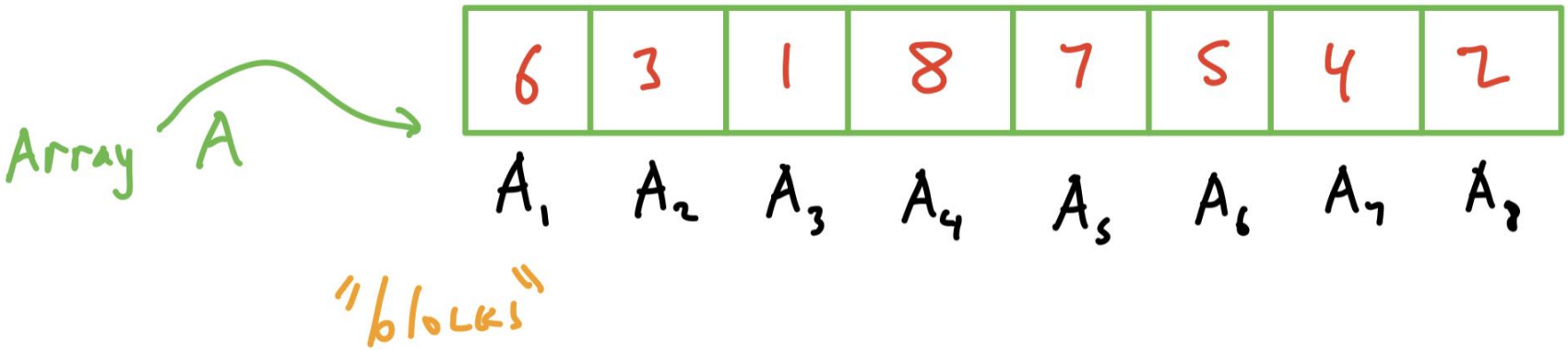
Input: An array of n integers



Output: The array sorted in increasing order

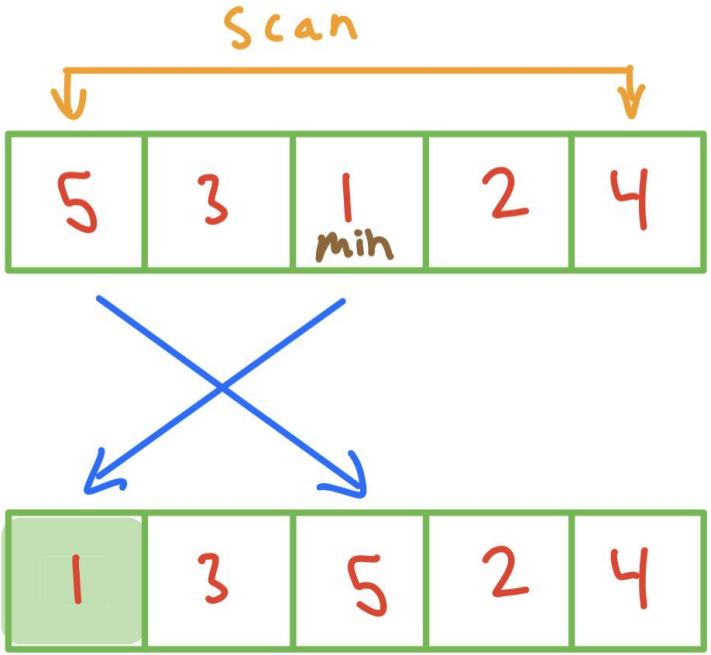


Simple operations

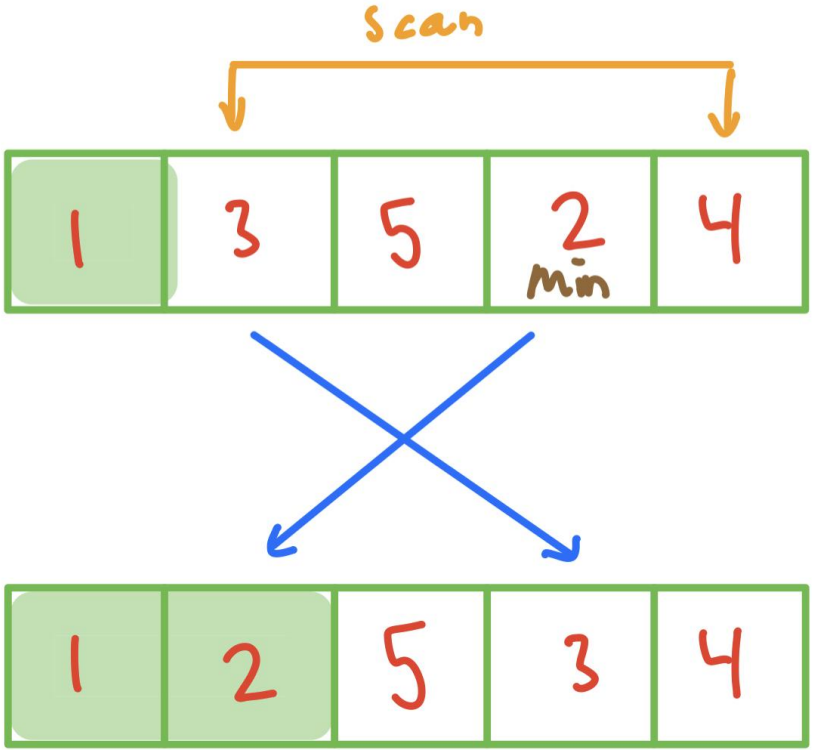


1. Comparisons
2. Swaps

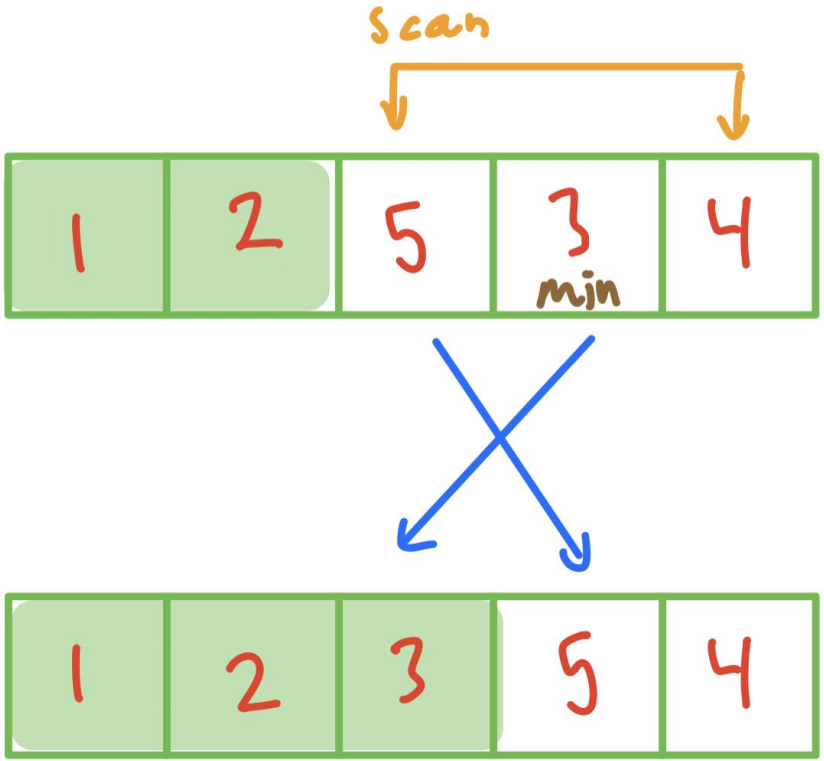
Iteration 1



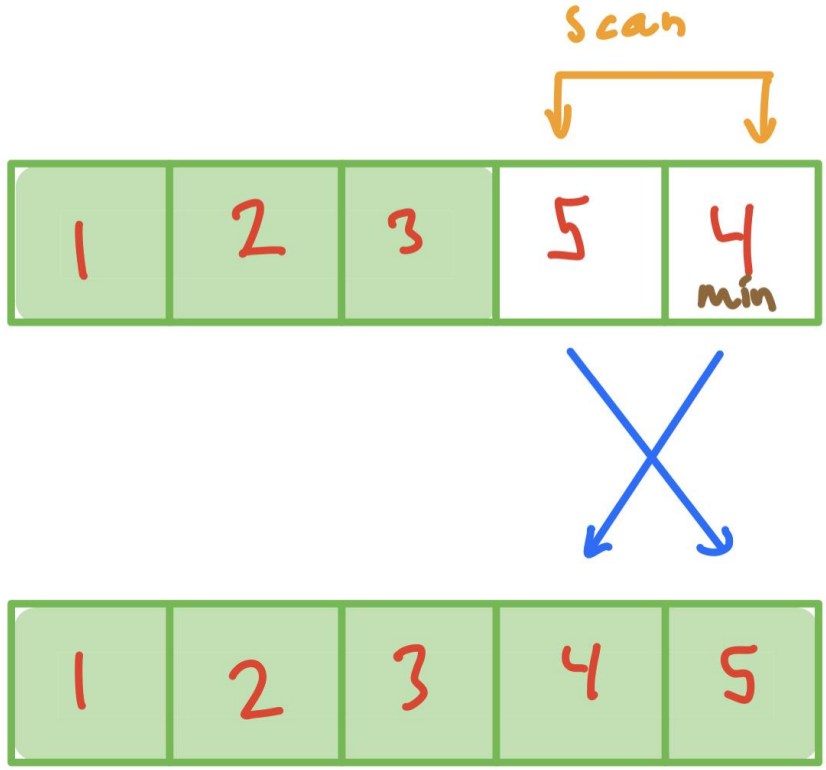
Iteration 2



Iteration 3



Iteration 4



Selection Sort

For an array A with n elements (length n)

Iteration 1) Scan $A_1 \rightarrow A_n$, find the minimum, swap with A_1

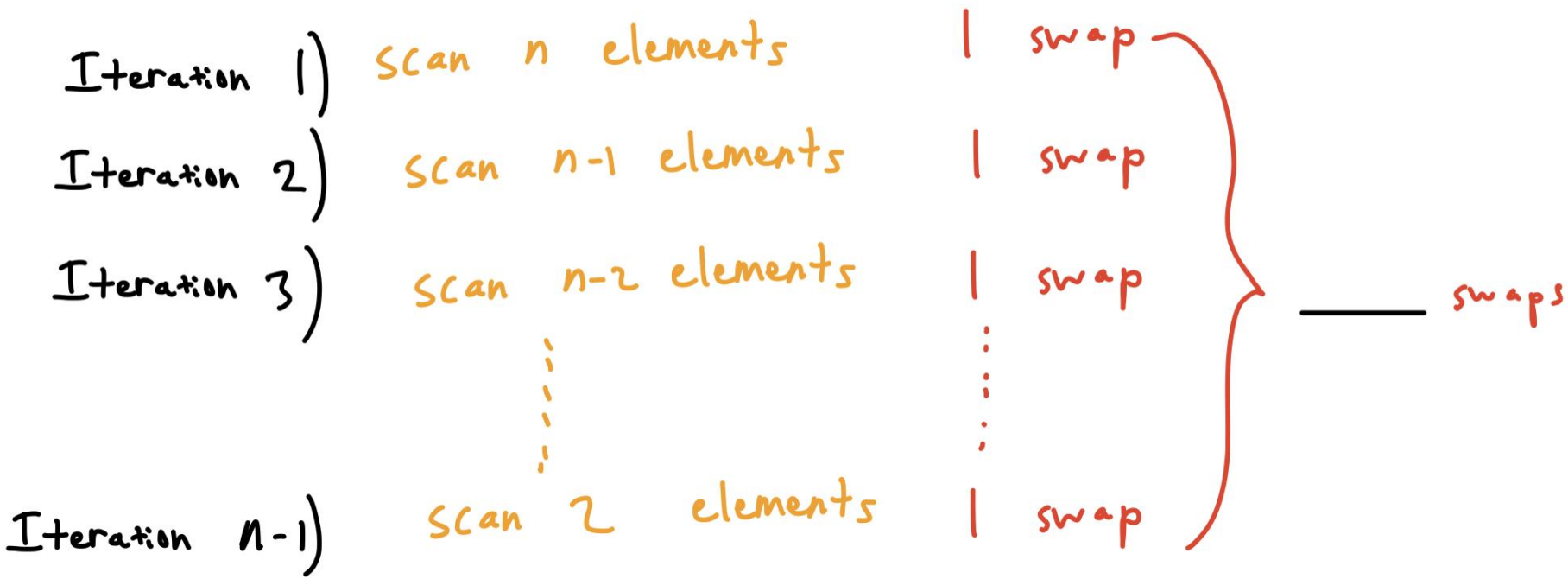
Iteration 2) Scan $A_2 \rightarrow A_n$, find the minimum, swap with A_2

Iteration 3) Scan $A_3 \rightarrow A_n$, find the minimum, swap with A_3

⋮

Iteration $n-1$) Scan $A_{n-1} \rightarrow A_n$, find the minimum, swap with A_{n-1}

Runtime Analysis



Runtime Analysis

$$T(n) = \underbrace{2 + 3 + 4 + \cdots + n}_{\text{total elements to scan}} + \underbrace{n-1}_{\text{swaps}}$$

Recall: $1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$

$T(n) =$

Runtime Analysis

$$T(n) = \frac{n(n+1)}{2} - 1 + n - 1$$

$$= \frac{1}{2}n^2 + \frac{1}{2}n - 1 + n - 1$$

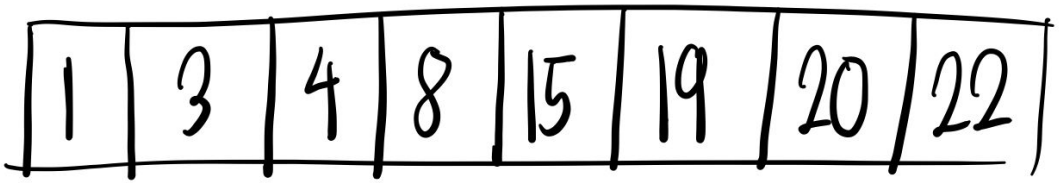
$$= \frac{1}{2}n^2 + \frac{3}{2}n - 2$$

$$= O(\_\_)$$

Selection Sort

https://youtu.be/92BfuxHn2XE

Can we do better than $O(n^2)$?

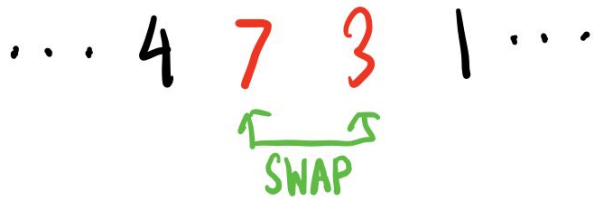


Selection sort takes $8+7+6+\cdots+1 = 36$ time.

We can do better, right?

Bubble sort

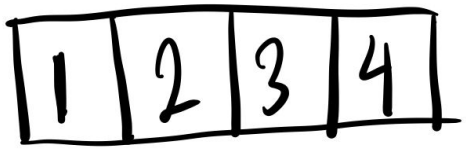
- Scan array from left to right, swapping out of order pairs.



- If there are no more pairs, we are done!

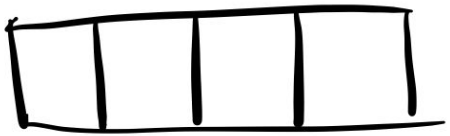
Otherwise, scan again.

Trying out bubble sort...

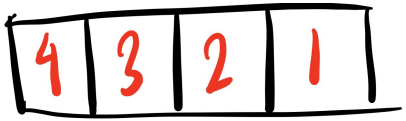


→ just one pass! O(n).

But is there a worse case?



Bubble Sort Worst Case



Bubble sort is not any better :(

Even though bubble sort sometimes runs in $O(n)$, in the worst case it is still order n^2.

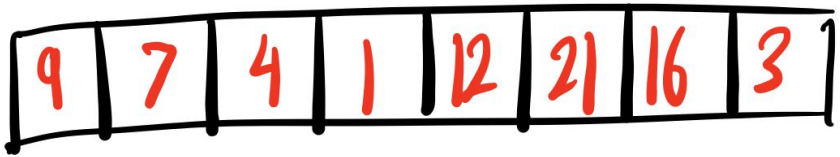
Bubble Sort

https://www.youtube.com/watch?v=Cq7SMsQBEUw

5 minute break

Divide and Conquer Algorithm for Sorting Numbers?

n = 8

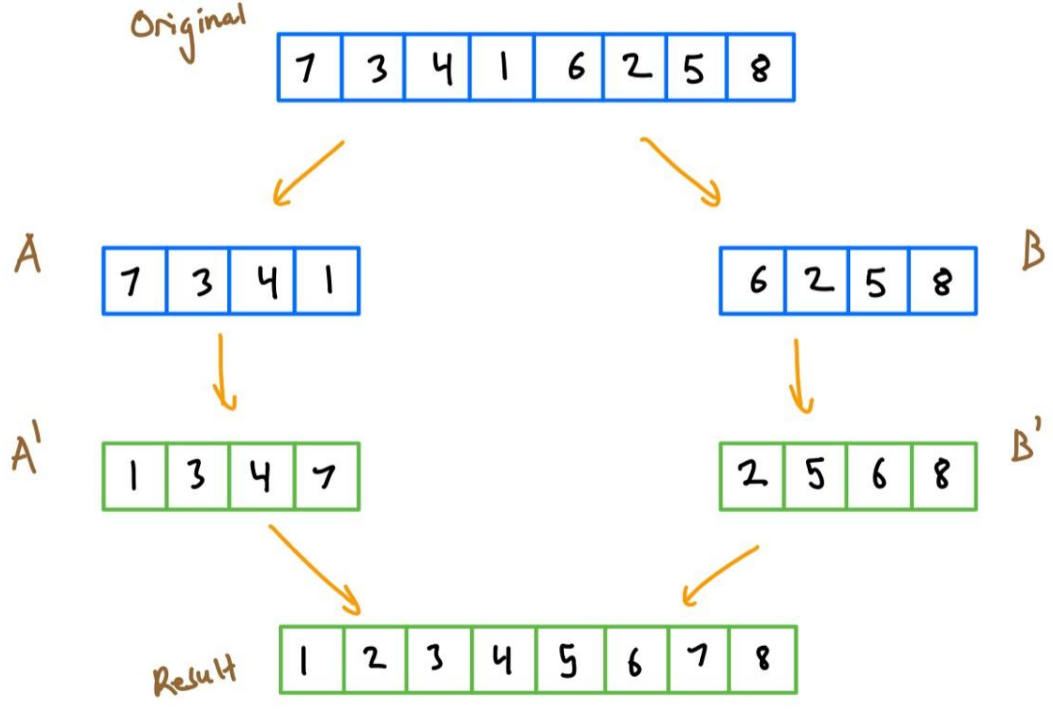


Divide?

Conquer?

Merge Sort

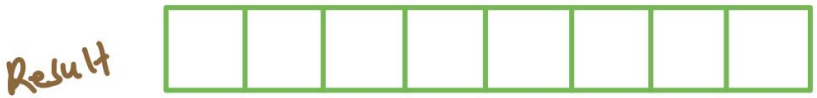
Merge Sort: Given an array, sorts it



1. Split array into half, A, B
2. Use merge sort on A
3. Use merge sort on B
4. Use merge routine on A' and B'

Merge Routine

- Pointers to beginning of A' and B', compare elements, take smallest for Result, move pointer, repeat
- When A' or B' is exhausted, copy other array



Example

1) Compare 1 and 2, insert 1 into Result

2) Compare 3 and 2, insert 2 into Result

3) Compare 3 and 5, insert 3 into Result

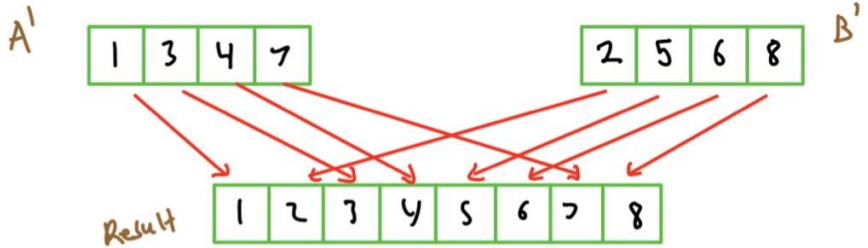
4) Compare 4 and 5, insert 4 into Result

5) Compare 7 and 5, insert 5 into Result

6) Compare 7 and 6, insert 6 into Result

7) Compare 7 and 8, insert 7 into Result

8) A' exhausted, insert 8 into Result

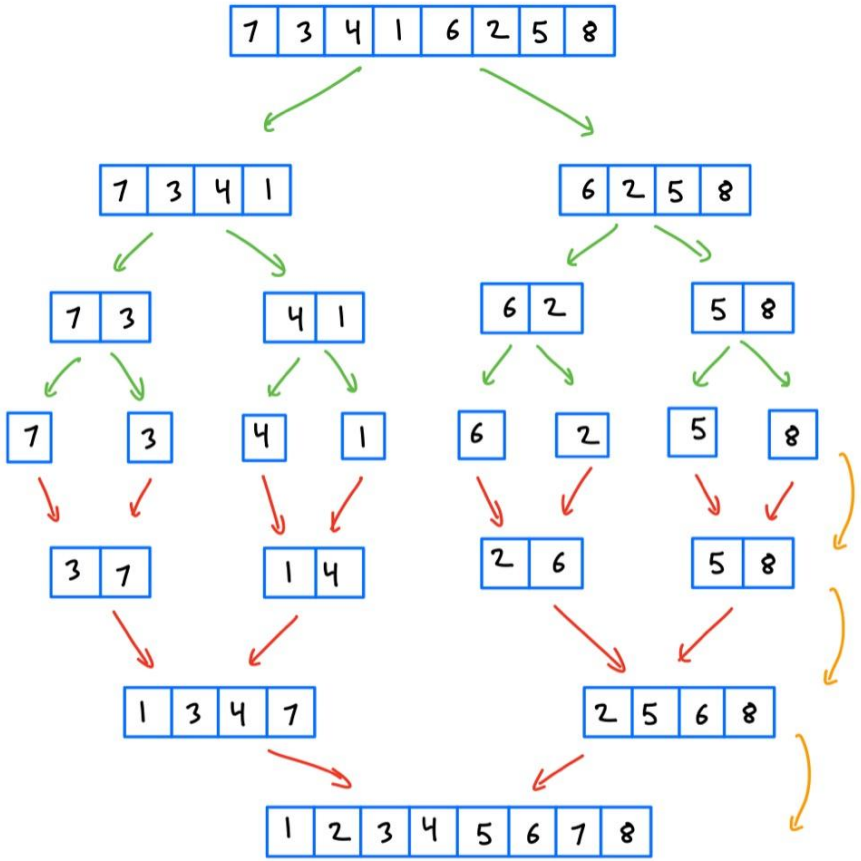


Merge Routine Analysis

- If A' and B' have a total of n elements
- We do n comparisons and insertions

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

Merge Sort Example



8 elements to compare

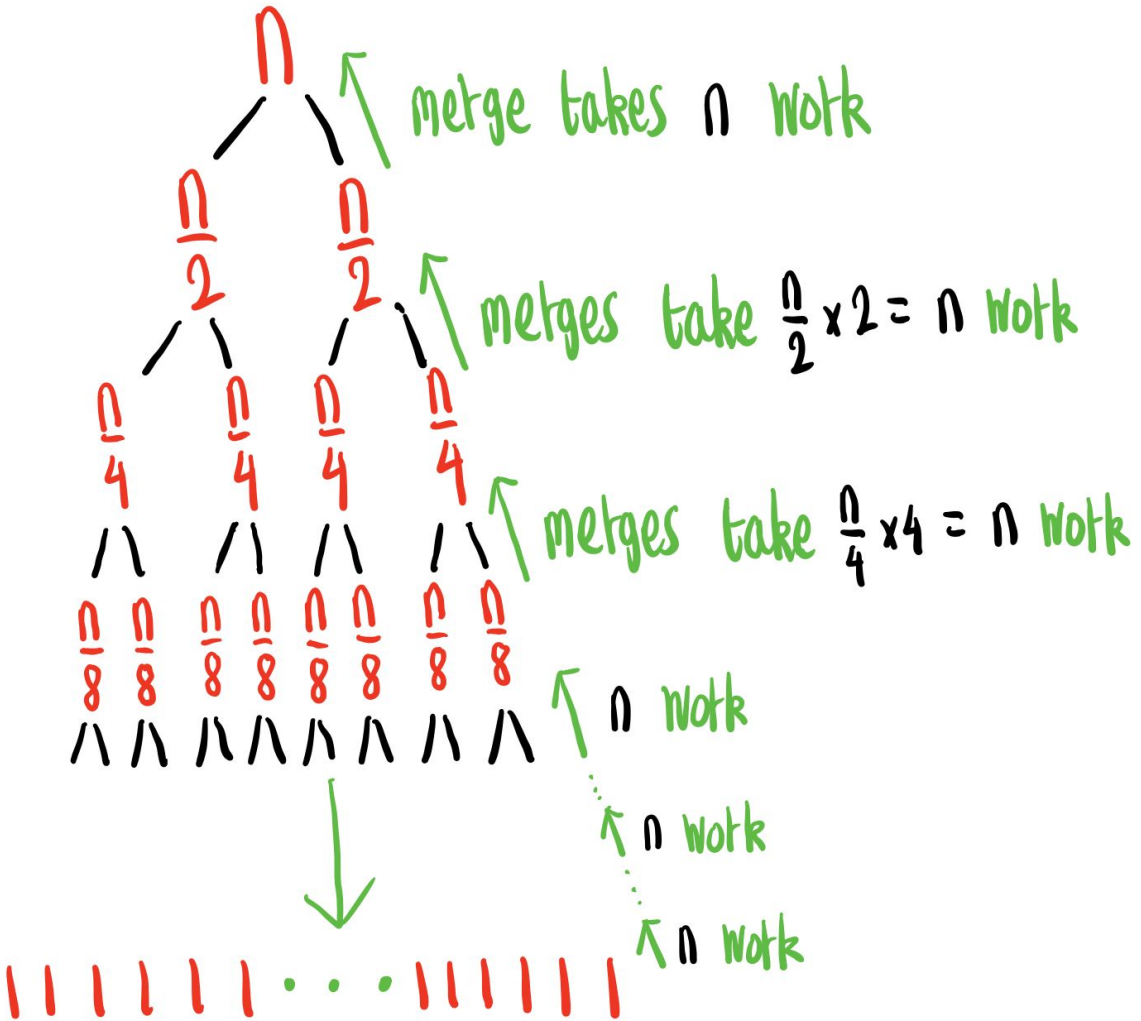
8 elements to compare

8 elements to compare

$\Longrightarrow$ 24 elements to compare

$T(8) = 24$

Recursion Tree for Merge Sort



Recursion Tree Analysis

n work per "level" of the recursion tree.

How many levels?

Logarithms

$\log_b n \Longrightarrow$ How many times do I have to divide n by b until I reach 1?

$\log_3 9 =$ ___ $\quad 9 \to 3 \to 1$

$\log_5 125 =$ ___ $\quad 125 \to 25 \to 5 \to 1$

$\log_2 64 =$ ___ $\quad 64 \to 32 \to 16 \to 8 \to 4 \to 2 \to 1$

Logarithms Practice

$\log_b n \Longrightarrow$ How many times do I have to divide n by b until I reach 1?

In this class, $\log n$ means $\log_2 n$

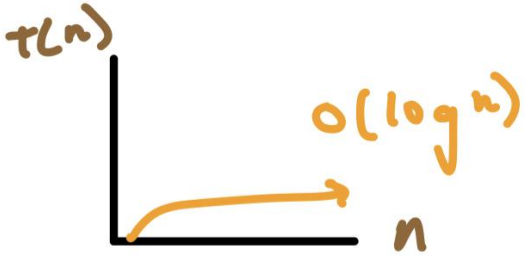
$\log 8 =$ ___

$\log 12 \approx$ ____

$\log 16 =$ ___

Logarithms

- for $\log n$, doubling input increases output by 1



$O(\log n)$ vs $O(n)$

$O(n \log n)$ vs $O(n^2)$

Runtime

$$(\text{levels}) \times \left(\frac{\text{comparisons}}{\text{level}}\right)$$

# elements	levels	comparisons per level	= total # of comparisons
2	1	2	$1 \times 2 = 2$
4	2	4	$2 \times 4 = 8$
8	3	8	$3 \times 8 = 24$
16	4	16	$4 \times 16 = 64$
⋮	⋮	⋮	⋮
n	_____	_____	_____

$T(n) =$ _____ $= O($_____$)$

Merge Sort

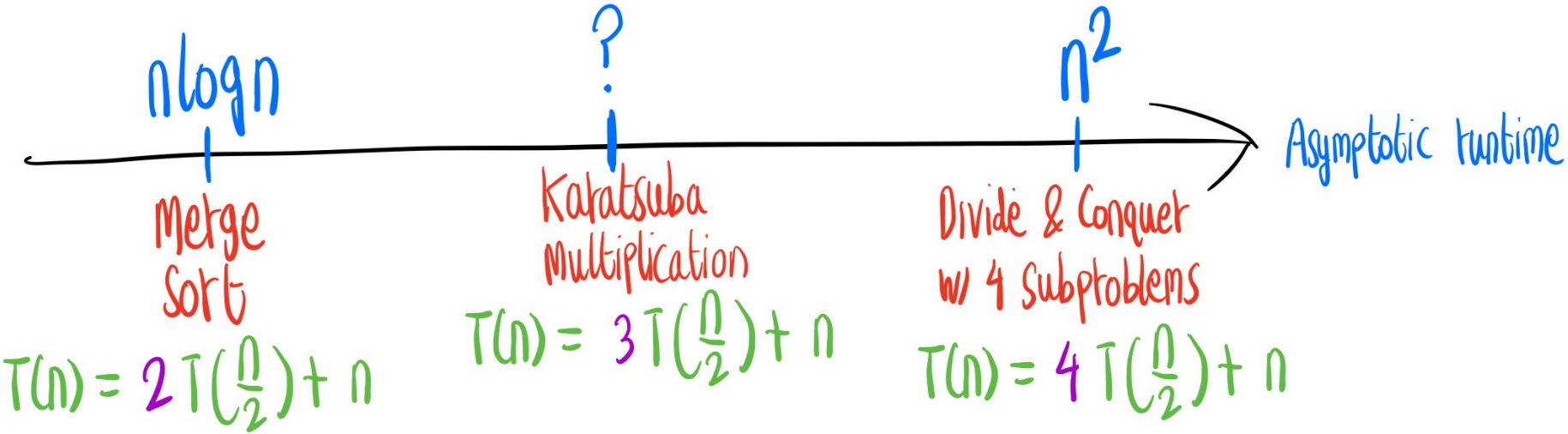
https://youtu.be/ZRPoEKHXTJg

Conclusion

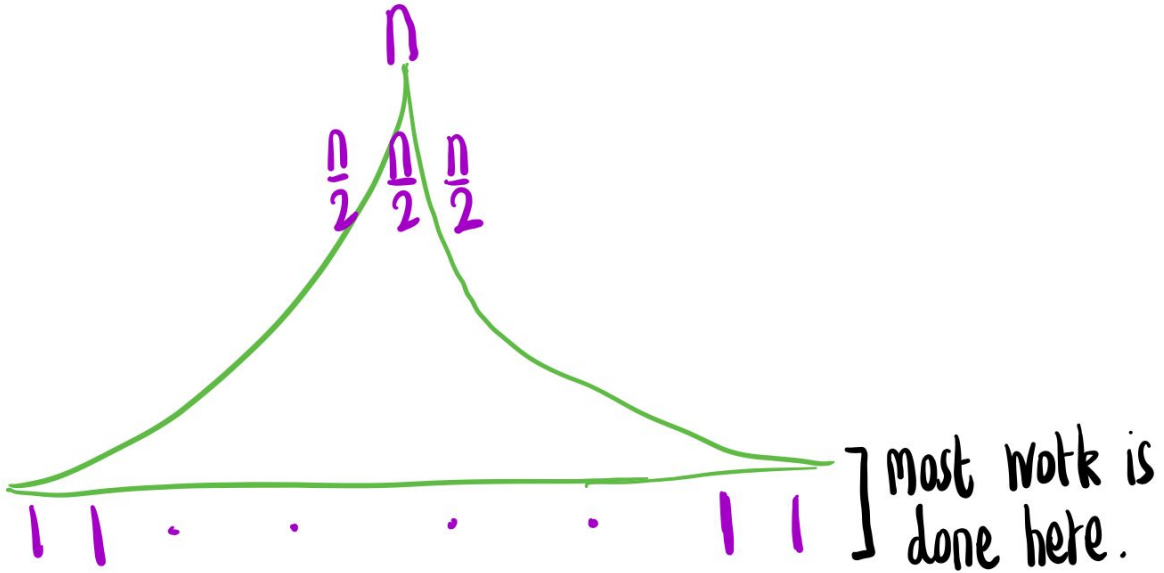
Merge sort is $O(n \log n)$.

In fact, this is an optimal algorithm for comparison-based sorting.

Returning to Karatsuba Multiplication...



Recursion Tree for Karatsuba Multiplication



The recursion tree for Karatsuba multiplication is leaf heavy.

Number of Leaves in Recursion Tree

$\log_2 n$ levels...

Each level, number of problems $\times 3$.

Hence, the number of leaves is

$3^{\log_2 n}$

...

$= \left(2^{\log_2 3}\right)^{\log_2 n} = \left(2^{\log_2 n}\right)^{\log_2 3} = \boxed{n^{\log_2 3}}$.

Can we do better for multiplication?

Karatsuba multiplication is $O(n^{\log_2 3}) = O(n^{1.58\ldots})$.

Out of scope...

Schonhage-Strassen (1971) is $O(n \log n \cdot \log\log n)$.

Efficient for $n > 10000$...

Harvey & Hoeven (2019) is $O(n \log n)$.

Will never be useful in practice!