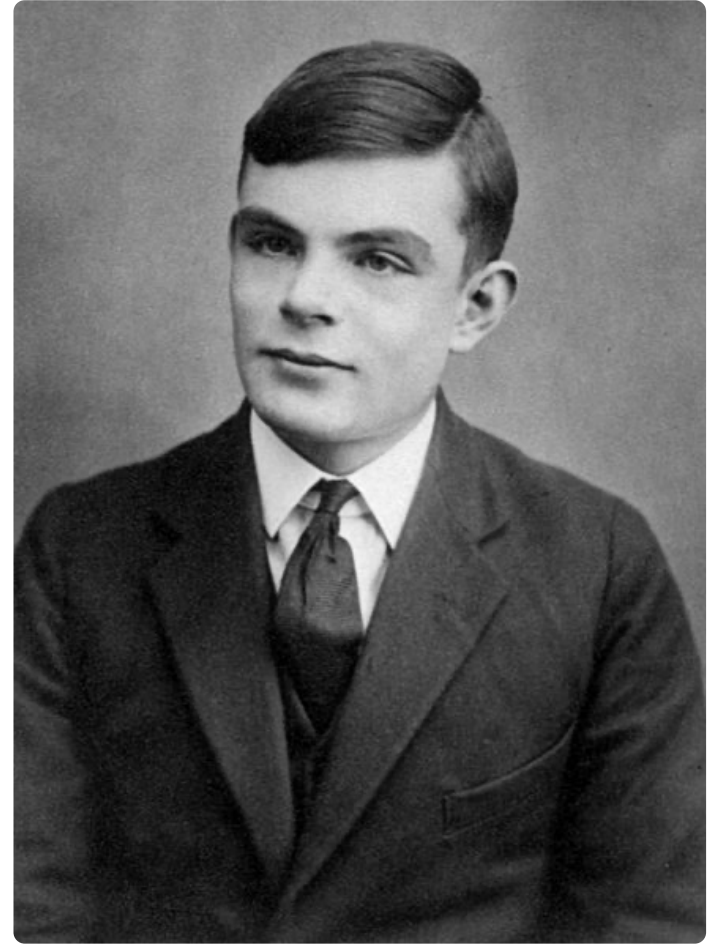




Ada Lovelace



Alan Turing

welcome to Algorithms

Sundamentalalgorithms.com



Last compiled January 12, 2026 at 12:45 Noon.
© Kent Quanrud, 2026.

Kent Quanrud, Spring 2026

Fundamental Algorithms, CS381, Spring 2026

Instructors:

- *Mike Atallah* (matallah@...).
Lectures: Tuesday and Thursday, 10:30–11:45AM, SMTH 108.
Office hours: Thursday, after lecture, in LWSN 2116D.
- *Kent Quanrud* (krq@...).
Lectures: Tuesday and Thursday, 1:30–2:45PM, BHEE 170.
Office hours: Tuesday, after lecture, in LWSN 1211.

Graduate TAs:	Email (@purdue.edu)	Office hours (LWSN Commons) ¹
Kyle Asbury	asbury7	Thursday, 1:30–2:30PM
Arnav Burudgunte	aburudgu	TBD
Qi Ling	ling102	Thursday, 9:30–10:30AM
Ethan Mader	emader	Thursday, 4–5PM
Han Qin	qin184	Tuesday, 2:30–3:30PM
Haoyu Song	song522	Tuesday, 1:30–2:30PM
Borna Tavasoli	btavasol	Thursday, 6–7PM
Justin Zhang	zhan3554	Wednesday, 12:30–1:30PM, Thursday, 3:30–4:30PM

Undergraduate TAs:	Email (@purdue.edu)	Office hours (LWSN Commons) ¹
Mukul Agarwal	agarw396	Thursday, 2:30–3:30PM
Wilbert Chu	chu285	TBD
Sharanya Devgon	sdevgon	Wednesday, 1:30–2:30PM
Jimmy Dinh	dinh16	Thursday, 11:30–12:30PM
Corbet Elkins	cdelkins	Thursday, 5–6PM
Aditya Gandhi	gandh105	Monday, 2:30–3:30PM
Pratyanch Jain	jain617	TBD
Amy Kang	kang568	Wednesday, 7–8PM
Maanas Karwa	mkarwa	Tuesday, 5:30–6:30PM
Parth Kulkarni	kulka142	TBD
Leo Lee	lee4247	Wednesday, 10:30–11:30AM
Matthew Li	li4691	Wednesday, 4:30–5:30PM
Philip Liu	liu3688	Monday, 4:30–5:30PM

Segyul Park	park1362	Wednesday, 2:30–3:30PM
Aakash Patel	pate2011	Wednesday, 11:30–12:30PM
Mridu Prashanth	mprasha	Wednesday, 3:30–4:30PM
Angela Qian	qian220	Thursday, 12:30–1:30PM
Alexandre Sauquet	asauquet	Tuesday, 4:30–5:30PM
Diya Singh	singh959	TBD
Sage Stefonic	sstefoni	TBD
Kovid Tandon	tdandon34	Tuesday, 3:30–4:30PM
Madhav Valiyaparambil	mvaliyap	Wednesday, 5:30–6:30PM
Nathan Vijitbenjaronk	wvijitbe	Thursday, 9:30–10:30AM
Aaryan Wadhwani	wadhwani2	TBD

PSO	Day	Time	Room
P01	Tuesday	3:30–4:20PM	ARMS B071
P02	Tuesday	4:30–5:20PM	ARMS B071
P03	Wednesday	10:30–11:20AM	WALC B091
P04	Wednesday	11:30AM–12:20PM	WALC B091
P08	Wednesday	2:30–3:20PM	WALC 3122
P07	Wednesday	3:30–4:20PM	WALC 3122
P05	Thursday	10:30–11:20AM	WALC B091
P06	Thursday	11:30AM–12:20PM	WALC B091

Please see the syllabus (Page 607) for more information.

Homework

- Homework 0 due January 22.
- Homework 1 due January 29.
- Homework 2 due February 5.
- Homework 3 due February 12.
- Homework 4 due February 26.
- Homework 5 due March 5.
- Homework 6 due March 12.
- Homework 7 due March 26.

¹TA office hours start in week 2.

one big .pdf
w/ everything

welcome to Graduate Algorithms

Fundamentalalgorithms.com

*Last compiled January 12, 2026 at 12:45 Noon.
© Kent Quanrud, 2026.*

Fundamental Algorithms, CS381, Spring 2026

Instructors:

- *Mike Atallah* (matallah@...).
Lectures: Tuesday and Thursday, 10:30–11:45AM, SMTH 108.
Office hours: Thursday, after lecture, in LWSN 2116D.
- *Kent Quanrud* (krq@...).
Lectures: Tuesday and Thursday, 1:30–2:45PM, BHEE 170.
Office hours: Tuesday, after lecture, in LWSN 1211.

Graduate TAs:	Email (@purdue.edu)	Office hours (LWSN Commons) ¹
Kyle Asbury	asbury7	Thursday, 1:30–2:30PM
Arnav Burudgunte	aburudgu	TBD
Qi Ling	ling102	Thursday, 9:30–10:30AM
Ethan Mader	emader	Thursday, 4–5PM
Han Qin	qin184	Tuesday, 2:30–3:30PM
Haoyu Song	song522	Tuesday, 1:30–2:30PM
Borna Tavasoli	btavasol	Thursday, 6–7PM
Justin Zhang	zhan3554	Wednesday, 12:30–1:30PM, Thursday, 3:30–4:30PM

Undergraduate TAs:	Email (@purdue.edu)	Office hours (LWSN Commons) ¹
--------------------	---------------------	--

Highlights

Awesome TA's

2 midterms, 5 final
weekly homework

drop lowest 20% of HW scores

late/resubmit HW policy

Please read the syllabus

– will ask for questions next class

Part I.

You already know TCS

- counting
- over/under

Counting



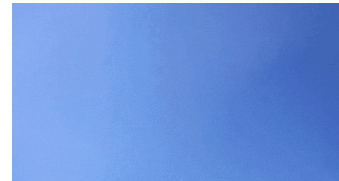
Counting

$$\text{|||||} \text{|||||} \text{|||||} \text{|||||} \text{|||} = 23$$

RHS much more efficient than LHS
"unary"

Decimal notation: 10^k #'s w/ k digits

Combinatorial
explosion!



Over/under

guess # between 1 and 100

A handwritten diagram in red ink. On the left, the numbers 50, 75, 67, 58, and 52 are listed vertically. On the right, the numbers 54, 57, 56, and 55 are listed vertically. A large red bracket spans from the top of the left column (50) down to the bottom (52). An arrow points from the top of this bracket to the number 54 in the right column.

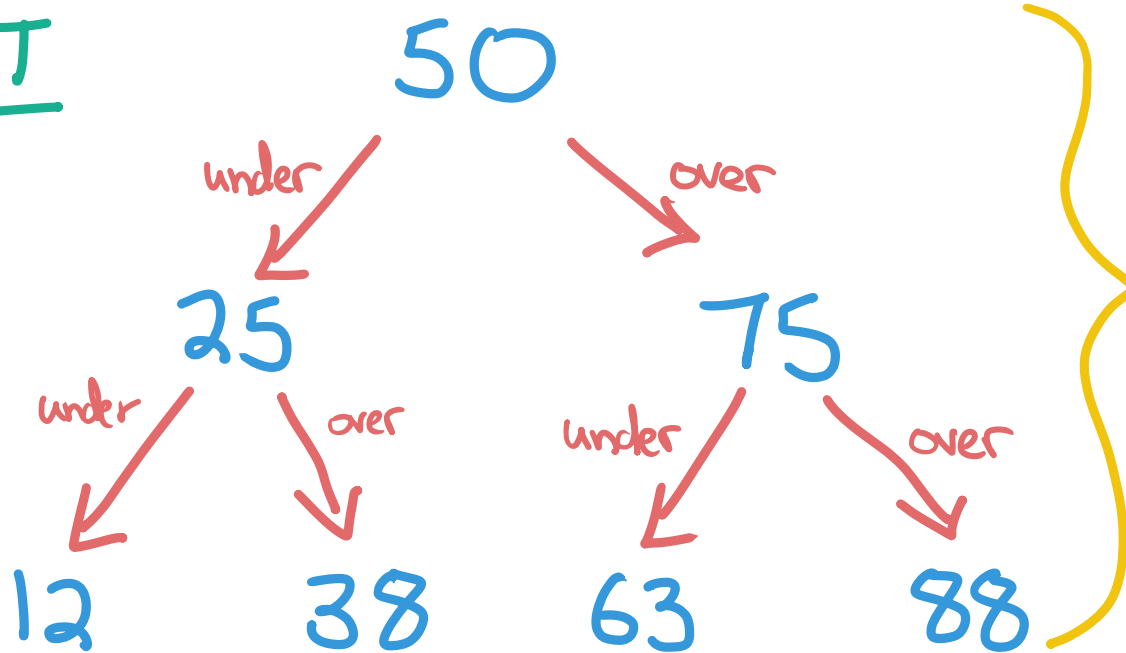
50	54
75	57
67	56
58	55
52	

Slow algo:

for $i=1, 2, \dots, 100$, guess i



FAST



rounds?

$$\log_2 100 \approx 6.67$$

$\Rightarrow$ 6 rounds

exponentials

$$2^n = \underbrace{2 \cdot 2 \cdot 2 \cdot \dots \cdot 2}_{n \text{ times}}$$

$$\begin{array}{ccc} 0 & 0 & 9 \\ \downarrow & \downarrow & \downarrow \\ 10^{23} & 10^{27} & 10^{46} \end{array}$$

87.5% 22%

logarithms

"logarithm (base 2) of n"

$$\log_2 n \text{ s.t. } 2^{\log_2 n} = n$$

also: $\log_e n$ (aka $\ln$)

$$\log_{10} n$$

$$\log_b n \text{ for some } b > 1$$

not

2

1

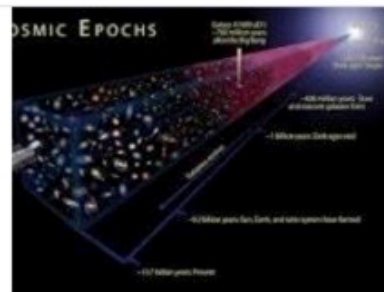
About 26,300,000 results (0.49 seconds)

10^{82} atoms

At this level, it is estimated that there are between 10^{78} to 10^{82} **atoms** in the known, observable **universe**. In layman's terms, that works out to between ten quadrillion vigintillion and one-hundred thousand quadrillion vigintillion **atoms**. Jul 30, 2009

How Many Atoms Are There in the Universe? - Universe Today

<https://www.universetoday.com/atoms-in-the-universe>



$$\log_2(\# \text{ atoms}) = \log_2(10^{82}) = 82 \underbrace{\log_2(10)}_{\approx 3.32} \approx 272.398$$

$$\log_2(\log_2(\# \text{ atoms}))$$

$$\approx \log_2(272.4)$$

$$\approx 8.~m$$

$$(2^8 = 256)$$

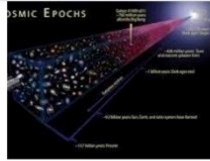
how many atoms in the universe

[All](#)
[Images](#)
[News](#)
[Shopping](#)
[Videos](#)
[More](#)
[Settings](#)
[Tools](#)

About 26,300,000 results (0.49 seconds)

10⁸² atoms

At this level, it is estimated that there are between 10⁷⁸ to 10⁸² **atoms** in the known, observable **universe**. In layman's terms, that works out to between ten quadrillion vigintillion and one-hundred thousand quadrillion vigintillion **atoms**. Jul 30, 2009



How Many Atoms Are There in the Universe? - Universe Today

<https://www.universetoday.com/atoms-in-the-universe>

$$\begin{aligned}
 &\log_2(\# \text{ atoms}) \\
 &= \log_2(10^{82}) \\
 &= 82 \log_2(10) \\
 &\approx 272.398
 \end{aligned}$$

TCS is about distinguishing

2^n ,
combinatorial
explosion

vs

$\log_2 n$,
binary search

Part II: Sorting

INEFFECTIVE SORTS

```
DEFINE HALFHEARTEDMERGESORT(LIST):  
  IF LENGTH(LIST) < 2:  
    RETURN LIST  
  PIVOT = INT(LENGTH(LIST) / 2)  
  A = HALFHEARTEDMERGESORT(LIST[:PIVOT])  
  B = HALFHEARTEDMERGESORT(LIST[PIVOT:])  
  // UMMMMM  
  RETURN[A,B] // HERE. SORRY.
```

```
DEFINE FASTBOGOSORT(LIST):  
  // AN OPTIMIZED BOGOSORT  
  // RUNS IN  $O(N \log N)$   
  FOR N FROM 1 TO LOG(LENGTH(LIST)):  
    SHUFFLE(LIST):  
    IF ISSORTED(LIST):  
      RETURN LIST  
  RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
```

```
DEFINE JOBSITEINTERVIEWQUICKSORT(LIST):  
  OK SO YOU CHOOSE A PIVOT  
  THEN DIVIDE THE LIST IN HALF  
  FOR EACH HALF:  
    CHECK TO SEE IF IT'S SORTED  
    NO, WAIT, IT DOESN'T MATTER  
    COMPARE EACH ELEMENT TO THE PIVOT  
    THE BIGGER ONES GO IN A NEW LIST  
    THE EQUAL ONES GO INTO, UH  
    THE SECOND LIST FROM BEFORE  
    HANG ON, LET ME NAME THE LISTS  
    THIS IS LIST A  
    THE NEW ONE IS LIST B  
    PUT THE BIG ONES INTO LIST B  
    NOW TAKE THE SECOND LIST  
    CALL IT LIST, UH, A2  
    WHICH ONE WAS THE PIVOT IN?  
    SCRATCH ALL THAT  
    IT JUST RECURSIVELY CALLS ITSELF  
    UNTIL BOTH LISTS ARE EMPTY  
    RIGHT?  
    NOT EMPTY, BUT YOU KNOW WHAT I MEAN  
    AM I ALLOWED TO USE THE STANDARD LIBRARIES?
```

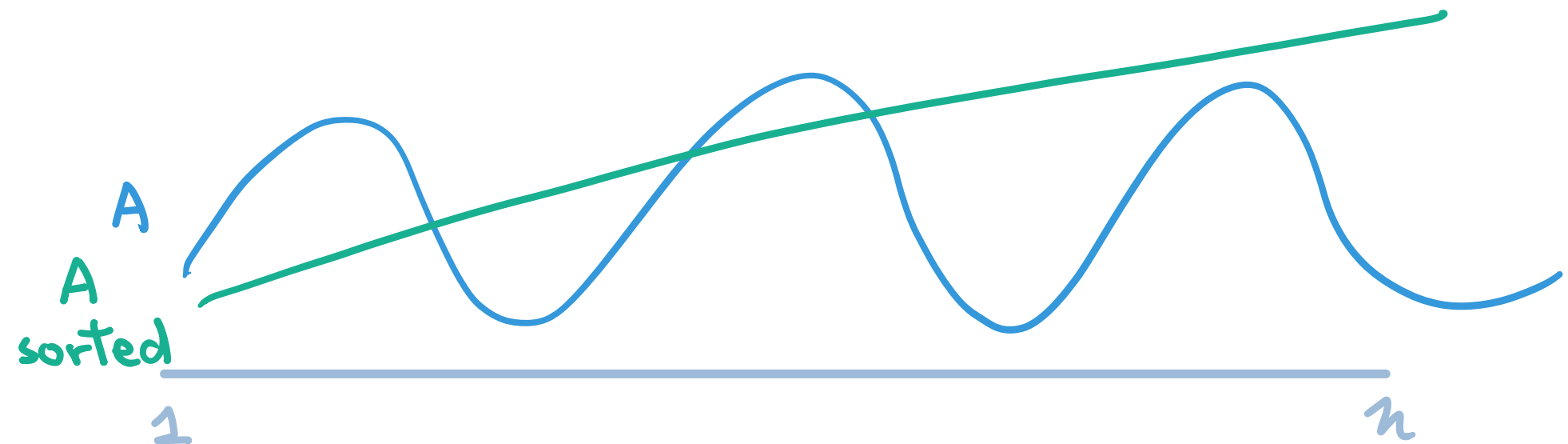
```
DEFINE PANICSORT(LIST):  
  IF ISSORTED(LIST):  
    RETURN LIST  
  FOR N FROM 1 TO 10000:  
    PIVOT = RANDOM(0, LENGTH(LIST))  
    LIST = LIST[PIVOT:] + LIST[:PIVOT]  
    IF ISSORTED(LIST):  
      RETURN LIST  
  IF ISSORTED(LIST):  
    RETURN LIST  
  IF ISSORTED(LIST): // THIS CAN'T BE HAPPENING  
    RETURN LIST  
  IF ISSORTED(LIST): // COME ON COME ON  
    RETURN LIST  
  // OH JEEZ  
  // I'M GONNA BE IN SO MUCH TROUBLE  
  LIST = [ ]  
  SYSTEM("SHUTDOWN -h +5")  
  SYSTEM("RM -rf .")  
  SYSTEM("RM -rf ~/*")  
  SYSTEM("RM -rf /")  
  SYSTEM("RD /S /Q C:\*") // PORTABILITY  
  RETURN [1, 2, 3, 4, 5]
```

Input: n numbers

✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓
5, 1, 8, 9, 2, 6, 7, 4, 11, 7, 9

Goal: sorted list

1, 2, 4, 5, 6, 7, 7, 8, 9, 9, 11



Human sort

5, 1, 8, 9, 2, 6, 7, 4, 11, 7, 9



- what comes first?
- what comes second?

1 2

5, 1, 8, 9, 2, 6, 7, 4, 11, 7, 9

 - what comes first?
- what comes second?



builds sorted list from beginning to end.
each time, we scan the list of n inputs

n iterations

$\times n$ items scanned per iteration

$\approx n^2$ time

Big-O

n	iterations
$\times n$	items scanned per iteration
n^2	time

" $O(n^2)$ ": there exists constants $N, c > 0$ s.t.

for all $n > N$, the algorithm terminates in
at most cn^2 steps

WTF "step"?! varies by computer, language, ~

low-level details only effect the constant

$O(m)$ hides the constant so we can develop a
general THEORY OF ALGORITHMS

Human sort: $O(n^2)$ time
(aka selection sort)

Better?

n^2 = each # is compared to every other #

can we sort all #'s without comparing every pair?

do we really need all comparisons?

Slightly different perspective

given array $A[1 \sim n]$ that is supposedly sorted

how long does it take to verify A is sorted?

$a_1 \leq a_2 \leq a_3 \leq a_4 \quad a_5 \quad a_6 \quad \dots$

$O(n)$

```
sorted? (A[1~n])  
① for i=1,...,n-1  
  ① if A[i] > A[i+1]  
    ① return False  
  ② return True
```

Human sort: $O(n^2)$ time
(aka selection sort)

Verifying a sort: $O(n)$

Better?

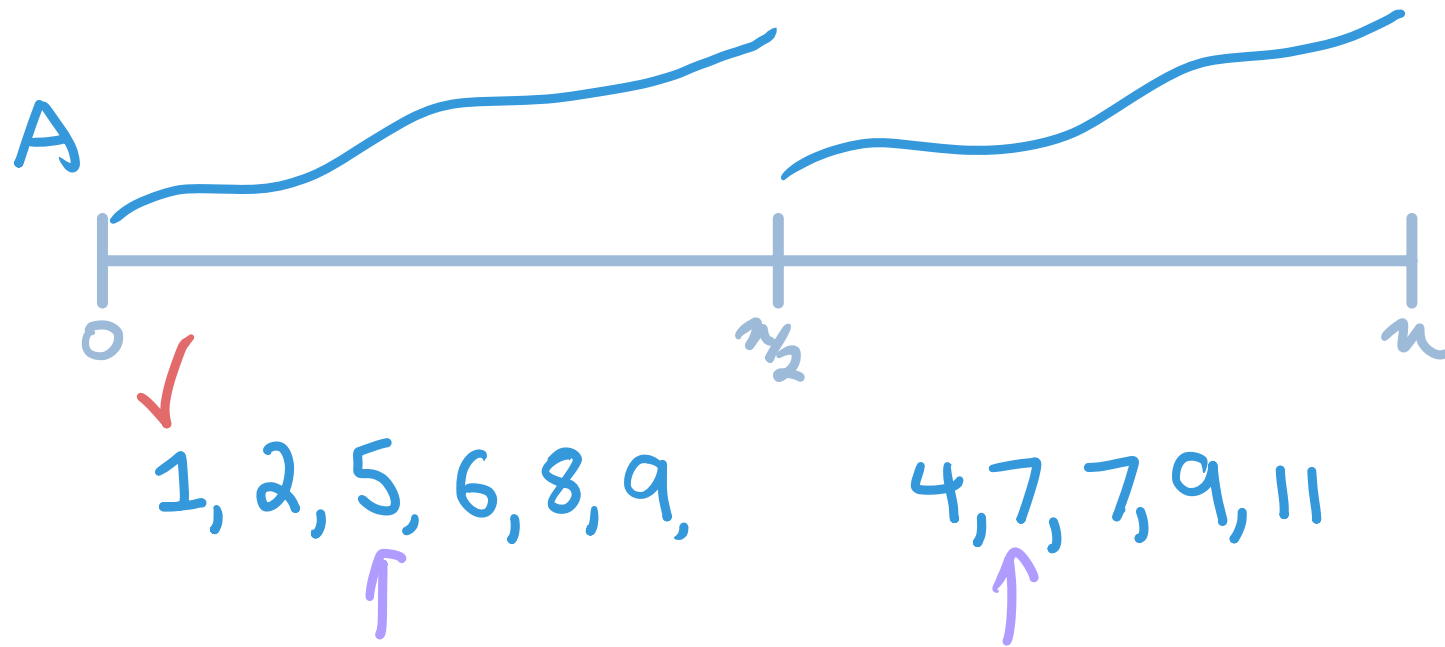
n^2 = each # is compared to every other #

can we sort all #'s without comparing every pair?

do we really need all comparisons?

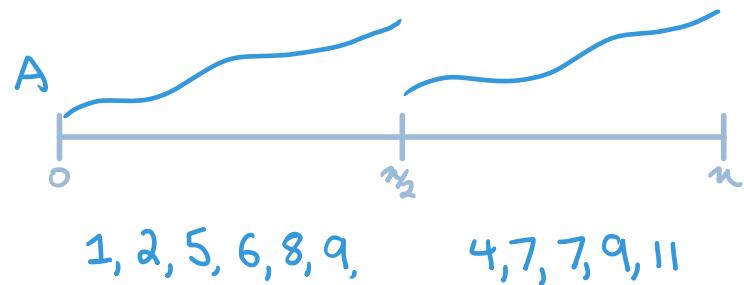
Simple case

suppose first & second halves are already sorted



1 2 4

Simple case
suppose first & second
halves are already sorted



merge($A[1..m], B[1..n]$)

/ Given two sorted lists A and B, produces the combined lists in sorted order.*

**/*

1. Allocate a new array $C[1..m + n]$, let $i = 1$, and let $j = 1$.
2. While $i \leq m$ and $j \leq n$:
 - A. If $A[i] \leq B[j]$:
 1. $C[i + j - 1] \leftarrow A[i]$ and $i \leftarrow i + 1$.
 - B. Else ($B[j] < A[i]$):
 1. $C[i + j - 1] \leftarrow B[j]$ and $j \leftarrow j + 1$.
3. While $i \leq m$:
 - A. $C[i + j - 1] \leftarrow A[i]$ and $i \leftarrow i + 1$.
4. While $j \leq n$:
 - A. $C[i + j - 1] \leftarrow B[j]$ and $j \leftarrow j + 1$.
5. Return C .

Recursive Specification

Input

Output

$\text{sort}(A[1..n])$: given an list/array
of n numbers, output the list in sorted
order

$\text{sort}(\sim)$ Divide-and-conquer

1. Sort first half recursively
2. Sort second half recursively
3. merge sorted halves together

$\text{merge}(A[1..m], B[1..n])$

/ Given two sorted lists A and B, produces the combined lists in sorted order.*

1. Allocate a new array $C[1..m+n]$, let $i = 1$, and let $j = 1$.
2. While $i \leq m$ and $j \leq n$:
 - A. If $A[i] \leq B[j]$:
 1. $C[i+j-1] \leftarrow A[i]$ and $i \leftarrow i+1$.
 - B. Else ($B[j] < A[i]$):
 1. $C[i+j-1] \leftarrow B[j]$ and $j \leftarrow j+1$.
3. While $i \leq m$:
 - A. $C[i+j-1] \leftarrow A[i]$ and $i \leftarrow i+1$.
4. While $j \leq n$:
 - A. $C[i+j-1] \leftarrow B[j]$ and $j \leftarrow j+1$.
5. Return C .

Designing a recursive algorithm

Divide-and-conquer

1. Sort first half recursively
2. Sort second half recursively
3. merge sorted halves together

① Recursive specification

`merge-sort( $A[1..n]$ )`: Given an array $A[1..n]$ of comparable elements, returns an array consisting of the elements of A sorted in increasing order.

- declares input/output of algo
- "contract" algo must fulfill
 $\Rightarrow$ can rely on
- induction hypothesis for proofs of correctness

Designing a recursive algorithm

Divide-and-conquer

1. Sort first half recursively
2. Sort second half recursively
3. merge sorted halves together

① Recursive specification

$\text{merge-sort}(A[1..n])$: Given an array $A[1..n]$ of comparable elements, returns an array consisting of the elements of A sorted in increasing order.

② Implement spec

- declares input/output of algo
- "contract" algo must fulfill
- induction hypothesis for correctness

$\text{merge-sort}(A[1..n])$

/ In the base case, the list is so short there is nothing to do. */*

1. If $n \leq 1$ then return A .

/ In the general case, we divide the input in half and recursively sort each half. */*

2. Let $m = \lfloor \frac{n}{2} \rfloor$.

3. $B_1[1..m] \leftarrow \text{merge-sort}(A[1..m])$.

4. $B_2[1..n - m] \leftarrow \text{merge-sort}(A[m + 1..n])$.

/ Now we merge the two sorted halves in linear time. */*

5. return $\text{merge}(B_1, B_2)$.

Designing a recursive algorithm

Divide-and-conquer

1. Sort first half recursively
2. Sort second half recursively
3. merge sorted halves together

① Recursive specification

$\text{merge-sort}(A[1..n])$: Given an array $A[1..n]$ of comparable elements, returns an array consisting of the elements of A sorted in increasing order.

② Implement spec

- declares input/output of algo
- "contract" algo must fulfill
- induction hypothesis for correctness

```
merge-sort( $A[1..n]$ )
```

```
/* In the base case, the list is so short there is nothing to do. */
```

```
1. If  $n \leq 1$  then return  $A$ .
```

Base case

```
/* In the general case, we divide the input in half and recursively sort each half. */
```

```
2. Let  $m = \lfloor \frac{n}{2} \rfloor$ .
```

```
3.  $B_1[1..m] \leftarrow \text{merge-sort}(A[1..m])$ .
```

satisfies recursive spec
by induction on n

```
4.  $B_2[1..n - m] \leftarrow \text{merge-sort}(A[m + 1..n])$ .
```

```
/* Now we merge the two sorted halves in linear time. */
```

```
5. return merge( $B_1, B_2$ ).
```

③ Prove correctness

argue algo satisfies recursive spec for all n

by induction on n

recursive spec = inductive hypothesis

Divide-and-conquer

1. Sort first half recursively
2. Sort second half recursively
3. merge sorted halves together

`merge-sort( $A[1..n]$ )`: Given an array $A[1..n]$ of comparable elements, returns an array consisting of the elements of A sorted in increasing order.

`merge-sort( $A[1..n]$ )`

/ In the base case, the list is so short there is nothing to do. */*

1. If $n \leq 1$ then return A .

Base case

/ In the general case, we divide the input in half and recursively sort each half. */*

2. Let $m = \lfloor \frac{n}{2} \rfloor$.

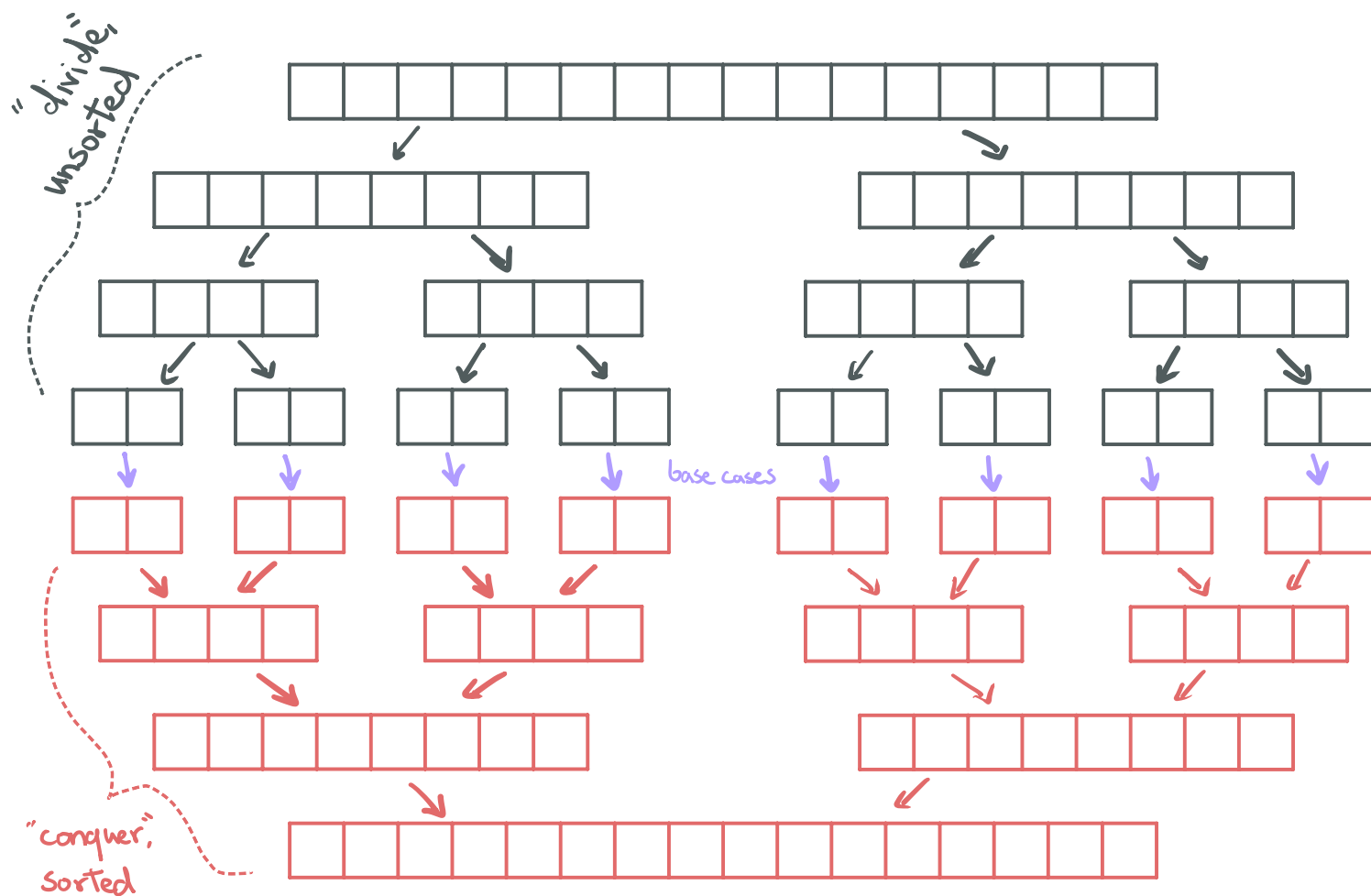
3. $B_1[1..m] \leftarrow \text{merge-sort}(A[1..m])$.

4. $B_2[1..n-m] \leftarrow \text{merge-sort}(A[m+1..n])$.

satisfies recursive spec
by induction on n

/ Now we merge the two sorted halves in linear time. */*

5. return `merge( $B_1, B_2$ )`.



Running time analysis

upper bound

$T(n)$ = time for input of size n

$$T(0) = T(1) = T(2) = 1$$

$$n > 1$$

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n$$

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

Divide-and-conquer

1. Sort first half recursively
2. Sort second half recursively
3. merge sorted halves together

merge-sort($A[1..n]$): Given an array $A[1..n]$ of comparable elements, returns an array consisting of the elements of A sorted in increasing order.

merge-sort($A[1..n]$)

/* In the base case, the list is so short there is nothing to do. */

1. If $n \leq 1$ then return A .

Base case

/* In the general case, we divide the input in half and recursively sort each half. */

2. Let $m = \lfloor \frac{n}{2} \rfloor$.

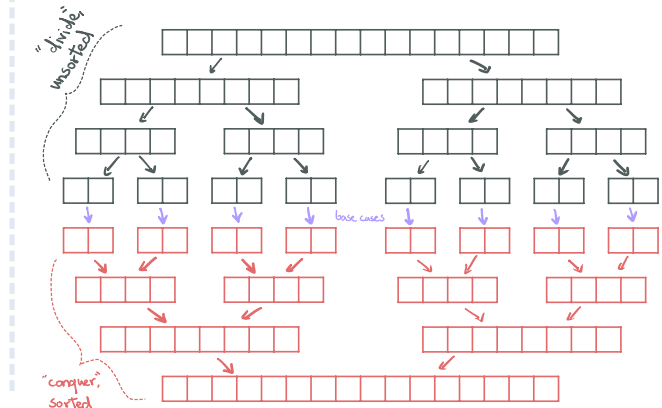
3. $B_1[1..m] \leftarrow \text{merge-sort}(A[1..m])$.

4. $B_2[1..n-m] \leftarrow \text{merge-sort}(A[m+1..n])$.

satisfies recursive spec by induction on n

/* Now we merge the two sorted halves in linear time. */

5. return merge(B_1, B_2).



$$T(n) \leq T(2^{\lceil \log_2 n \rceil})$$

Running time analysis

$T(n)$ = time for input of size n

$$T(1) = T(0) = 1$$

$n > 1$:

$T(n) =$

Divide-and-conquer

1. Sort first half recursively
2. Sort second half recursively
3. merge sorted halves together

`merge-sort( $A[1..n]$ )`: Given an array $A[1..n]$ of comparable elements, returns an array consisting of the elements of A sorted in increasing order.

```
merge-sort( $A[1..n]$ )
```

```
/* In the base case, the list is so short there is nothing to do. */
```

```
1. If  $n \leq 1$  then return  $A$ .
```

Base case

```
/* In the general case, we divide the input in half and recursively sort each half. */
```

```
2. Let  $m = \lfloor \frac{n}{2} \rfloor$ .
```

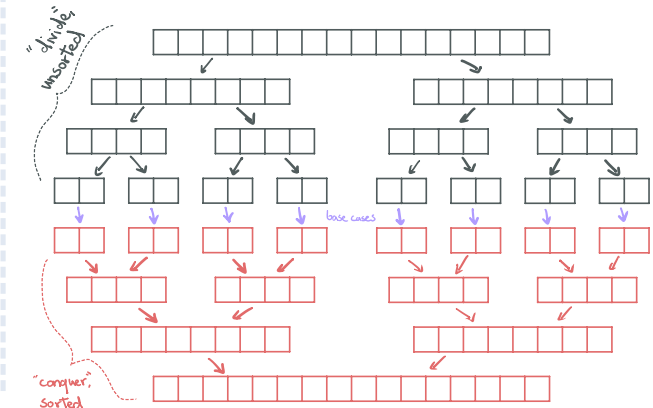
```
3.  $B_1[1..m] \leftarrow \text{merge-sort}(A[1..m])$ .
```

satisfies recursive spec by induction on n

```
4.  $B_2[1..n-m] \leftarrow \text{merge-sort}(A[m+1..n])$ .
```

```
/* Now we merge the two sorted halves in linear time. */
```

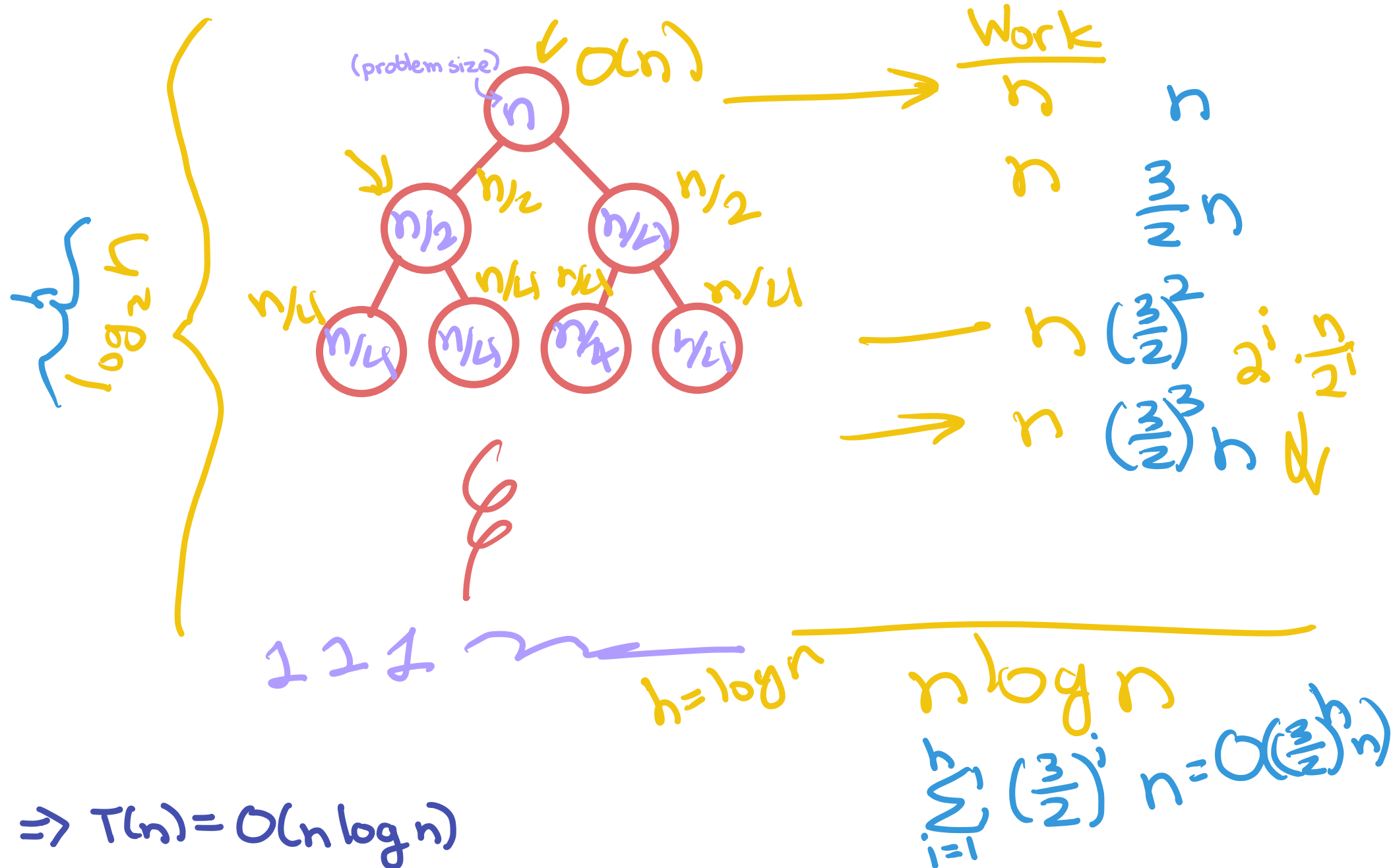
```
5. return merge( $B_1, B_2$ ).
```



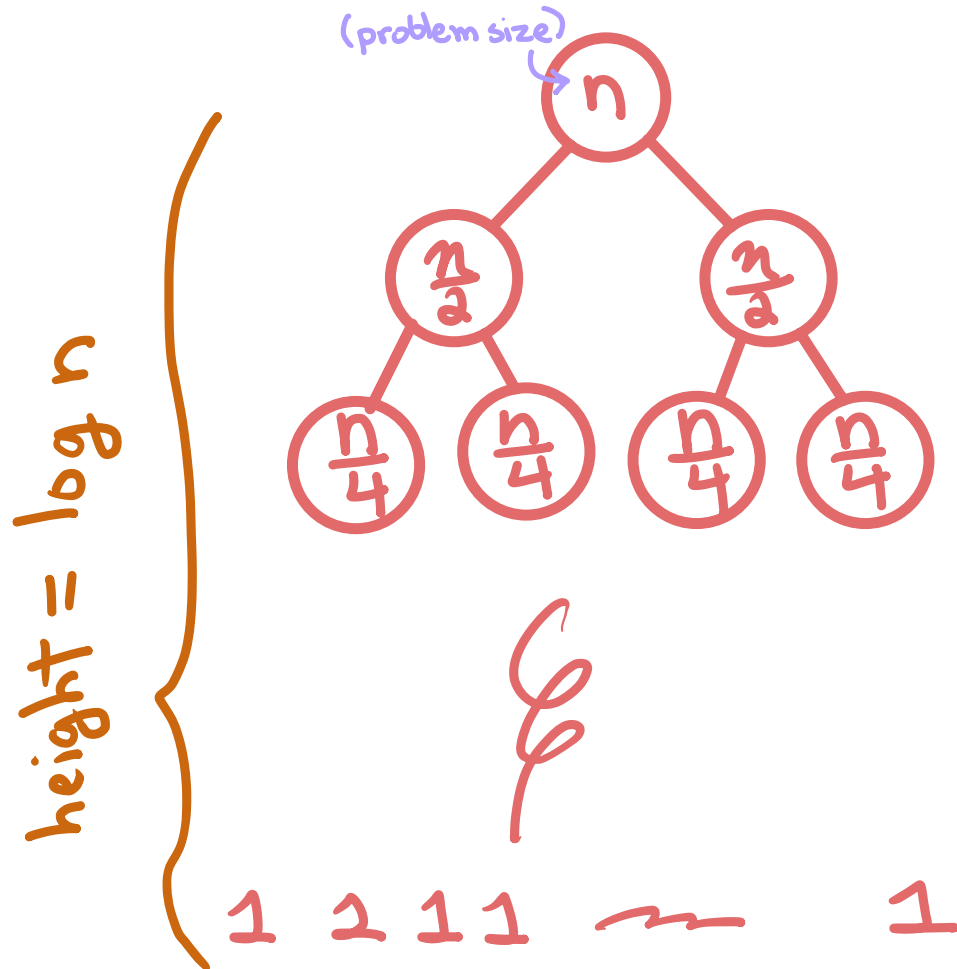
Recursion tree

$$T(1) = T(0) = 1$$

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



Recursion tree



$$T(1) = T(0) = 1$$

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

Work

$$n$$

$$2 \cdot \left(\frac{n}{2}\right) = n$$

$$4 \cdot \left(\frac{n}{4}\right) = n$$

$$\left\{ \begin{array}{l} 2^k \cdot \frac{n}{2^k} = n \end{array} \right.$$

$$+ n \cdot 1 = n$$

$$n \times \log(n)$$

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

III

Lower Bounds For Sorting

Lower Bounds For Sorting

merge-sort: $O(n \log n)$. better? optimal?

Goal: prove lower bound for all sorting algorithms

e.g. prove any correct algorithm has to take at least

$\Omega(f(n))$ time

"big-Omega"

(like $O(\sim)$ but for lower bounds)

Comparison model

- Input: $x_1, \dots, x_n$ comparable
- only info is via comparison queries
"is $x_i < x_j$?"

Goal lower bound # comparison Q's

Lower Bounds for Sorting
merge-sort: $O(n \log n)$. better? optimal?
Goal: prove lower bound for all sorting algorithms
eg. prove any correct algorithm has to take at least

$\Omega(S(n))$ time
"big-Omega"
(like $O(\sim)$ but for lower bounds)

Suppose algo makes k queries
and outputs list $(x_{i_1}, x_{i_2}, \dots, x_{i_n})$

1. $x_{i_1} < x_{j_1}?$
2. $x_{i_2} < x_{j_2}?$
3. $x_{i_3} < x_{j_3}?$
4. $x_{i_4} < x_{j_4}?$
- $\vdots$
- k. $x_{i_k} < x_{j_k}?$

$\Rightarrow (x_{i_1}, x_{i_2}, x_{i_3}, \dots, x_{i_n})$

Comparison model

- Input: $x_1, \dots, x_n$ comparable
- only info via comparison queries
"is $x_i < x_j$?"

Goal lower bound # comparisons

- output is a function of k queries

- each query has 2 outcomes (YES/NO)

$\Rightarrow$ at most 2^k possible outputs

meanwhile...

$n!$ possible lists

$\Rightarrow 2^k \geq n!$ just to be able to output all lists

$$k \geq \log(n!)$$

$$\begin{aligned} n! &= n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1 \\ &\geq \left(\frac{n}{2}\right)^{n/2} \end{aligned}$$

$$k \geq \log\left(\left(\frac{n}{2}\right)^{n/2}\right) = \frac{n}{2} \log\left(\frac{n}{2}\right) = \Omega(n \log n)$$

Comparison model

- Input: $x_1, \dots, x_n$ comparable
- only info via comparison queries "is $x_i < x_j$?"

Goal lower bound # comparisons

Suppose algo makes k queries and outputs list $(x_{i_1}, x_{i_2}, \dots, x_{i_n})$

1 $x_{i_1} < x_{i_2}$?
2 $x_{i_2} < x_{i_3}$?
3 $x_{i_3} < x_{i_4}$?
4 $x_{i_4} < x_{i_5}$?
⋮
 k $x_{i_k} < x_{i_{k+1}}$?

$\Rightarrow (x_{i_1}, x_{i_2}, x_{i_3}, \dots, x_{i_n})$

- output is a function of k queries
 - each query has 2 outcomes (YES/NO)
- $\Rightarrow$ at most 2^k possible outputs

Theorem In the comparison (only) model, any (correct, deterministic) sorting algorithm makes $\Omega(n \log n)$ comparisons in the worst case.

Upper bound

$O(n \log n)$
(mergesort)

Lower bound

$\Omega(n \log n)$

in general (any lower bound) $\leq$ (any upper bound)

here we also have

(an upper bound) $\leq$ (some constant) (a lower bound)
 $[O(n \log n)]$ $[\Omega(n \log n)]$

bounds mutually certify each other to be optimal
"bounds are tight" (up to constants)

Conclusion: $n \log n$ is "right answer" for sorting in comparison model

Exercise 1.6. Let $A[1..n] \in \mathbb{R}^n$ be an array of n numbers. An *inversion* is a pair of numbers out of increasing order; more precisely, a pair of indices $i, j \in [n]$ such that $i < j$ and $A[i] > A[j]$. Design and analyze an algorithm (as fast as possible) for counting the number of inversions in A .

Exercise 1.7. Two points $(a, b), (c, d) \in \mathbb{R}^2$ are *comparable* if either (a) $a \leq c$ and $b \leq d$, or (b) $c \leq a$ and $d \leq b$.

Given a set of n points $(a_1, b_1), \dots, (a_n, b_n) \in \mathbb{R}^2$, consider the problem of computing the number of pairs of points that are comparable. You may assume for simplicity that all points are distinct. Design and analyze an algorithm (as fast as possible) for counting the number of comparable pairs of points.¹²

Fundamental Algorithms, CS381, Spring 2026

Instructors:

- *Mike Atallah* (matallah@...).
 Lectures: Tuesday and Thursday, 10:30–11:45AM, SMTH 108.
 Office hours: Thursday, after lecture, in LWSN 2116D.
- *Kent Quanrud* (krq@...).
 Lectures: Tuesday and Thursday, 1:30–2:45PM, BHEE 170.
 Office hours: Tuesday, after lecture, in LWSN 1211.

Graduate TAs:	Email (@purdue.edu)	Office hours (LWSN Commons) ¹
Kyle Asbury	asbury7	Thursday, 1:30–2:30PM
Arnav Burudgunte	aburudgu	TBD
Qi Ling	ling102	Thursday, 9:30–10:30AM
Ethan Mader	emader	Thursday, 4–5PM
Han Qin	qin184	Tuesday, 2:30–3:30PM
Haoyu Song	song522	Tuesday, 1:30–2:30PM
Borna Tavasoli	btavasol	Thursday, 6–7PM
Justin Zhang	zhan3554	Wednesday, 12:30–1:30PM, Thursday, 3:30–4:30PM

Undergraduate TAs:	Email (@purdue.edu)	Office hours (LWSN Commons) ¹
Mukul Agarwal	agarw396	Thursday, 2:30–3:30PM
Wilbert Chu	chu285	TBD
Sharanya Devgon	sdevgon	Wednesday, 1:30–2:30PM
Jimmy Dinh	dinh16	Thursday, 11:30–12:30PM
Corbet Elkins	cdelkins	Thursday, 5–6PM
Aditya Gandhi	gandh105	Monday, 2:30–3:30PM
Pratyanch Jain	jain617	TBD
Amy Kang	kang568	Wednesday, 7–8PM
Maanas Karwa	mkarwa	Tuesday, 5:30–6:30PM
Partth Kulkarni	kulka142	TBD
Leo Lee	lee4247	Wednesday, 10:30–11:30AM
Matthew Li	li4691	Wednesday, 4:30–5:30PM
Philip Liu	liu3688	Monday, 4:30–5:30PM

Segyul Park	park1362	Wednesday, 2:30–3:30PM
Aakash Patel	pate2011	Wednesday, 11:30–12:30PM
Mridu Prashanth	mprasha	Wednesday, 3:30–4:30PM
Angela Qian	qian220	Thursday, 12:30–1:30PM
Alexandre Sauquet	asauquet	Tuesday, 4:30–5:30PM
Diya Singh	singh959	TBD
Sage Stefonic	sstefoni	TBD
Kovid Tandon	taandon34	Tuesday, 3:30–4:30PM
Madhav Valiyaparambil	mvaliyap	Wednesday, 5:30–6:30PM
Nathan Vijitbenjaronk	wvijitbe	Thursday, 9:30–10:30AM
Aaryan Wadhwani	wadhwani2	TBD

PSO	Day	Time	Room
P01	Tuesday	3:30–4:20PM	ARMS B071
P02	Tuesday	4:30–5:20PM	ARMS B071
P03	Wednesday	10:30–11:20AM	WALC B091
P04	Wednesday	11:30AM–12:20PM	WALC B091
P08	Wednesday	2:30–3:20PM	WALC 3122
P07	Wednesday	3:30–4:20PM	WALC 3122
P05	Thursday	10:30–11:20AM	WALC B091
P06	Thursday	11:30AM–12:20PM	WALC B091

Please see the syllabus (Page [607](#)) for more information.

Homework

- Homework [0](#) due January 22.
- Homework [1](#) due January 29.
- Homework [2](#) due February 5.
- Homework [3](#) due February 12.
- Homework [4](#) due February 26.
- Homework [5](#) due March 5.
- Homework [6](#) due March 12.
- Homework [7](#) due March 26.

¹TA office hours start in week 2.