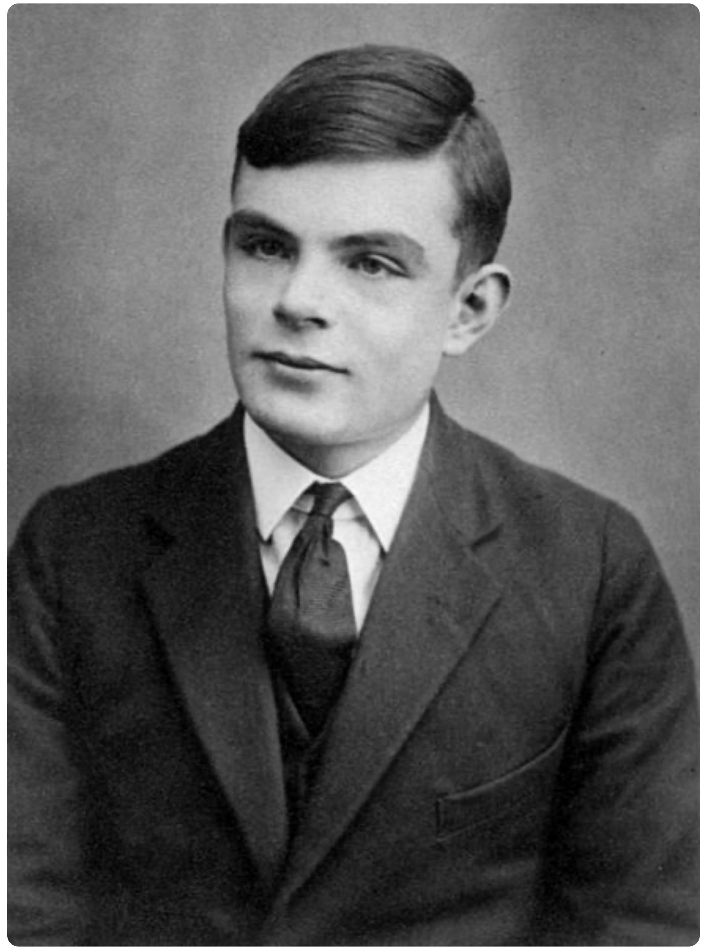
Ada Lovelace



Alan Turing

welcome to Algorithms


fundamentalalgorithms.com

one big .pdf w/ everything



**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 | tandon34 | Tuesday, 3:30–4:30PM |
| Madhav Valiyaparambil | mvaliyap | Wednesday, 5:30–6:30PM |
| Nathan Vijitbenjaronk | wvijitbe | Thursday, 9:30–10:30AM |
| Aaryan Wadhwani | wadhwan2 | 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.

[1] TA office hours start in week 2.

welcome to Graduate Algorithms

fundamentalalgorithms.com

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


**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] |
|---|---|---|

Please read the syllabus

- will ask for questions next class

Highlights

Awesome TA's

2 midterms, final

weekly homework

drop lowest 20% of HW scores

late/resubmit HW policy

# Part I.

You already know TCS

- counting
- over/under

# Counting

# Counting

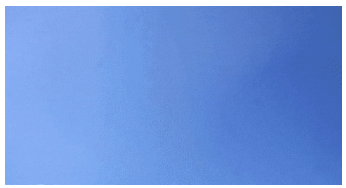
||||||||||||||||||||||| = 23

RHS much more efficient than LHS

"unary"

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

Combinatorial explosion!



# Over/under

guess # between 1 and 100

Slow algo:

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

FAST

50

under → 25, over → 75

25: under → 12, over → 38

75: under → 63, over → 88

12 38 63 88

\# rounds?

$\log_2 100 \approx 6.67$

$\Rightarrow$ 6 rounds

## exponentials

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

## logarithms

"logarithm (base 2) of n"

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

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

$\log_{10} n$

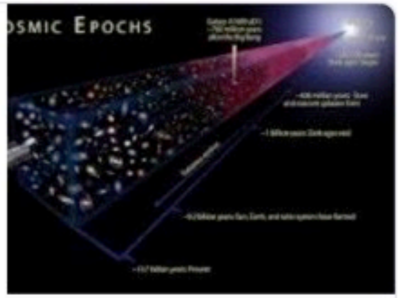
$\log_b n$ for some $b > 1$

Google

how many atoms in the universe

All | Images | News | Shopping | Videos | More | Settings | Tools

About 26,300,000 results (0.49 seconds)



## $10^{82}$ atoms

At this level, it is estimated that the 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

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

$$\approx 272.398$$

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

$$\approx \log_2(272.4)$$

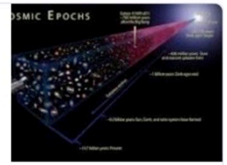
$$\approx 8.\sim$$

$$(2^8 = 256)$$

how many atoms in the universe

All Images News Shopping Videos More Settings Tools

About 26,300,000 results (0.49 seconds)




## $10^{82}$ atoms

At this level, it is estimated that the 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

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

$$= \log_2(10^{82})$$

$$= 82\log_2(10)$$

$$\approx 272.398$$

TCS is about distinguishing

$2^n$, combinatorial explosion vs $\log_2 n$, binary search

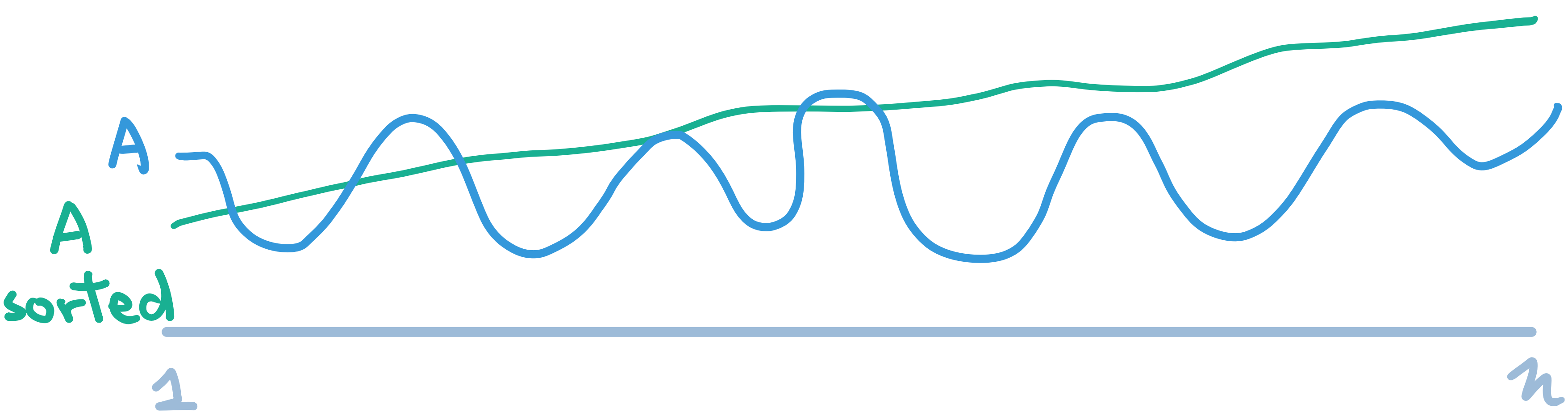
# Part II: Sorting

Input: n numbers

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

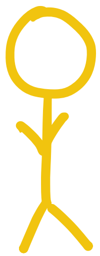
Goal: sorted list

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



# Human sort

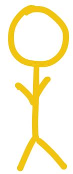
5, $\boxed{1}$, 8, 9, $\boxed{2}$, 6, 7, 4, 11, 7, 9



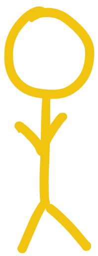
- what comes first?
- what comes second?

1 2

5, $\boxed{1}$, 8, 9, $\boxed{2}$, 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

$$\begin{array}{ll} & n \text{ iterations} \\ \times & n \text{ items scanned per iteration} \\ \hline \approx & n^2 \text{ time} \end{array}$$

# Big-O

$$\begin{array}{ll} & n \text{ iterations} \\ \times & n \text{ items scanned per iteration} \\ \hline & n^2 \text{ time} \end{array}$$

"$O(n^2)$": 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

$O(\sim)$ 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?

$$a_1 \overset{\leq ?}{\wedge} a_2 \overset{\leq ?}{\wedge} a_3 \quad a_4 \quad a_5 \quad a_6 \quad \sim$$

$O(n)$

sorted? $(A[1 \sim n])$

1. for $i = 1, \ldots, n-1$
   a. 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?

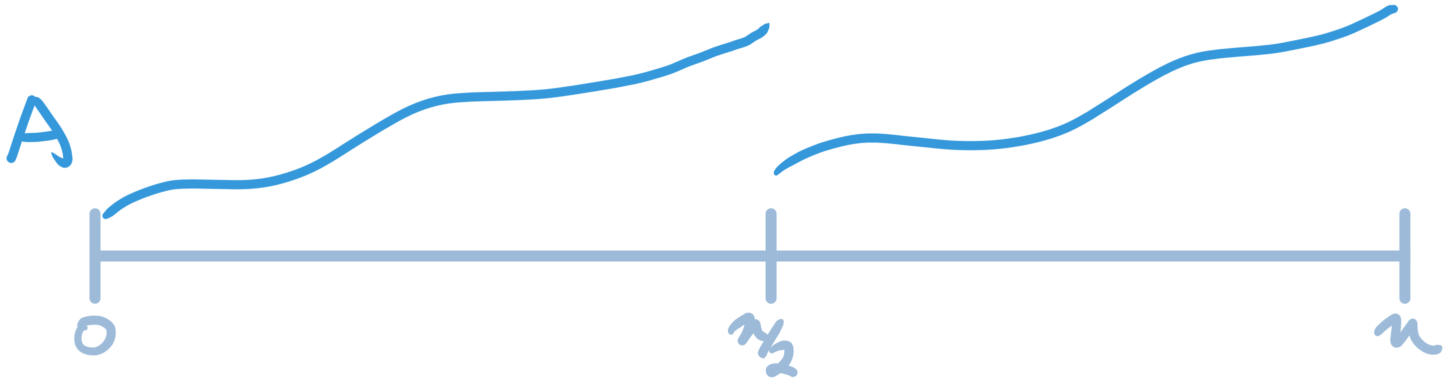
$n^2$ = each # is compared to every other #

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

do we really need all comparisons?

# Simple case

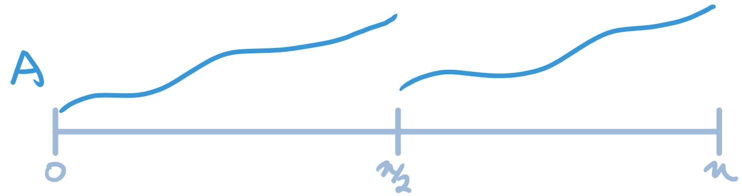
suppose first & second halves are already sorted




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

Simple case

suppose first & second halves are already sorted




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

`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. $C[i+j-1] \leftarrow B_2[j]$ and $j \leftarrow j+1$.
5. Return $C$.

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 $i \le m$ and $j \le n$:
   A. If $A[i] \le B[j]$:
      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 $i \le m$:
   A. $C[i+j-1] \leftarrow A[i]$ and $i \leftarrow i+1$.
4. While $j \le n$:
   A. $C[i+j-1] \leftarrow B_2[j]$ and $j \leftarrow j+1$.
5. Return $C$.

# Divide-and-conquer

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

# 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
  ⇝ can rely on
- induction hypothesis for proof 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

`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
- induction hypothesis for correctness

## ② Implement spec

`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. $B_1[1..m] \leftarrow$ merge-sort($A[1..m]$).
4. $B_2[1..n-m] \leftarrow$ 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`($B_1$, $B_2$).

# 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
- induction hypothesis for correctness

## ② Implement spec

```
merge-sort(A[1..n])
/* In the base case, the list is so short there is nothing to do.                          */
1. If n ≤ 1 then return A.
/* In the general case, we divide the input in half and recursively sort each half.        */
2. Let m = ⌊n/2⌋.
3. B_1[1..m] ← merge-sort(A[1..m]).
4. B_2[1..n − m] ← merge-sort(A[m + 1..n]).
/* Now we merge the two sorted halves in linear time.                                       */
5. return merge(B_1, B_2).
```

Base case

satisfies recursive spec by induction on n

## ③ Prove correctness

argue algo satisfies recursive spec for all n by induction on 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 $n \le 1$ then return $A$. ← Base case

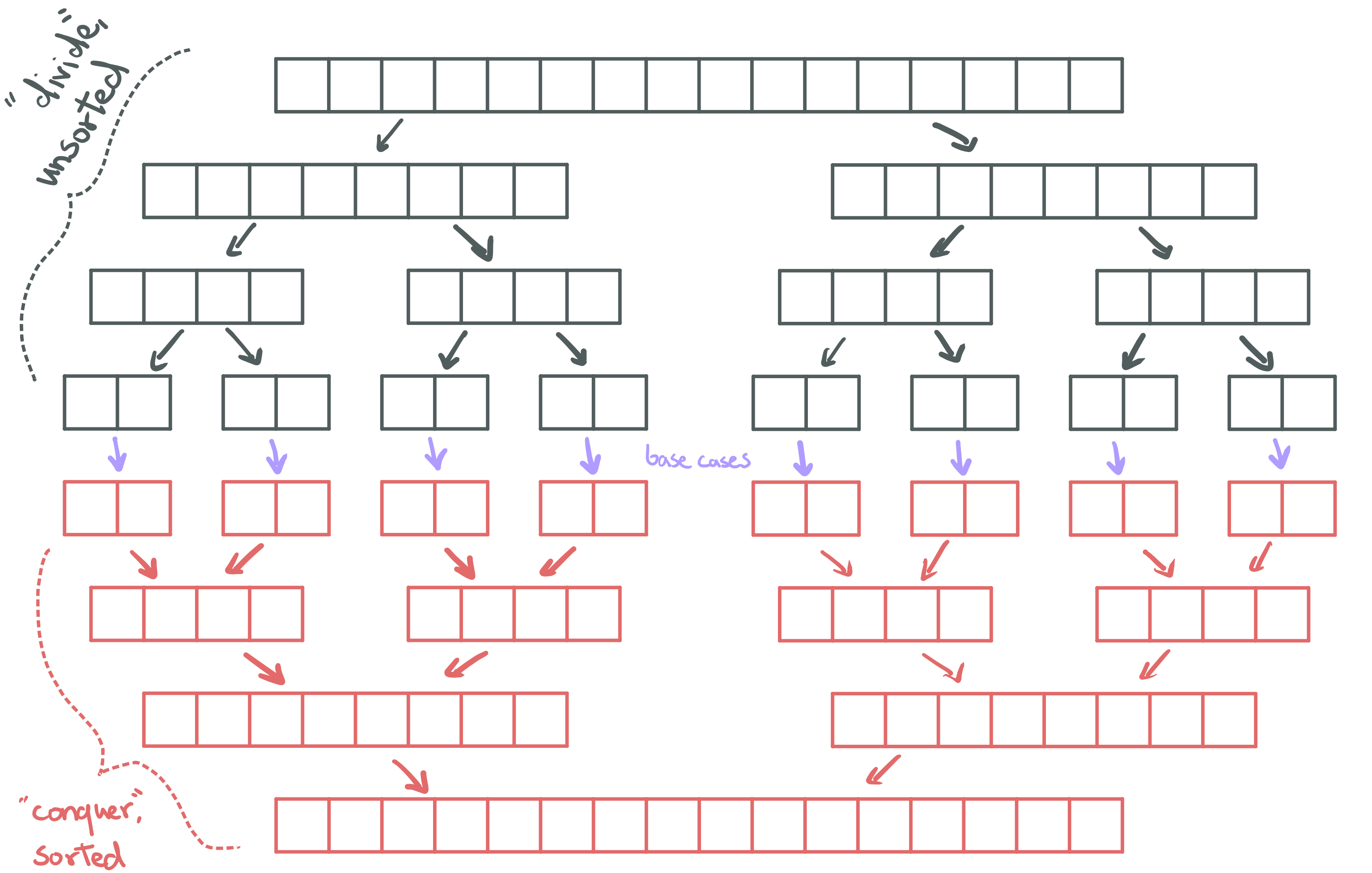
*/* In the general case, we divide the input in half and recursively sort each half. */*

2. Let $m = \lfloor \frac{n}{2} \rfloor$.
3. $B_1[1..m] \leftarrow$ merge-sort$(A[1..m])$.
4. $B_2[1..n-m] \leftarrow$ 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

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

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

$n > 1$:

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

$$\overset{*}{\approx} 2T(n/2) + O(n)$$

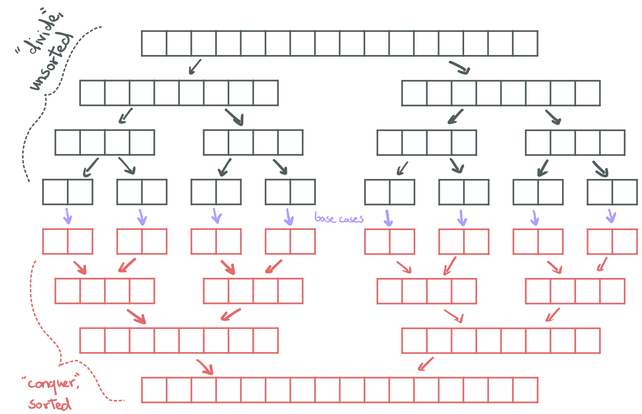
* see notes for justification

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.

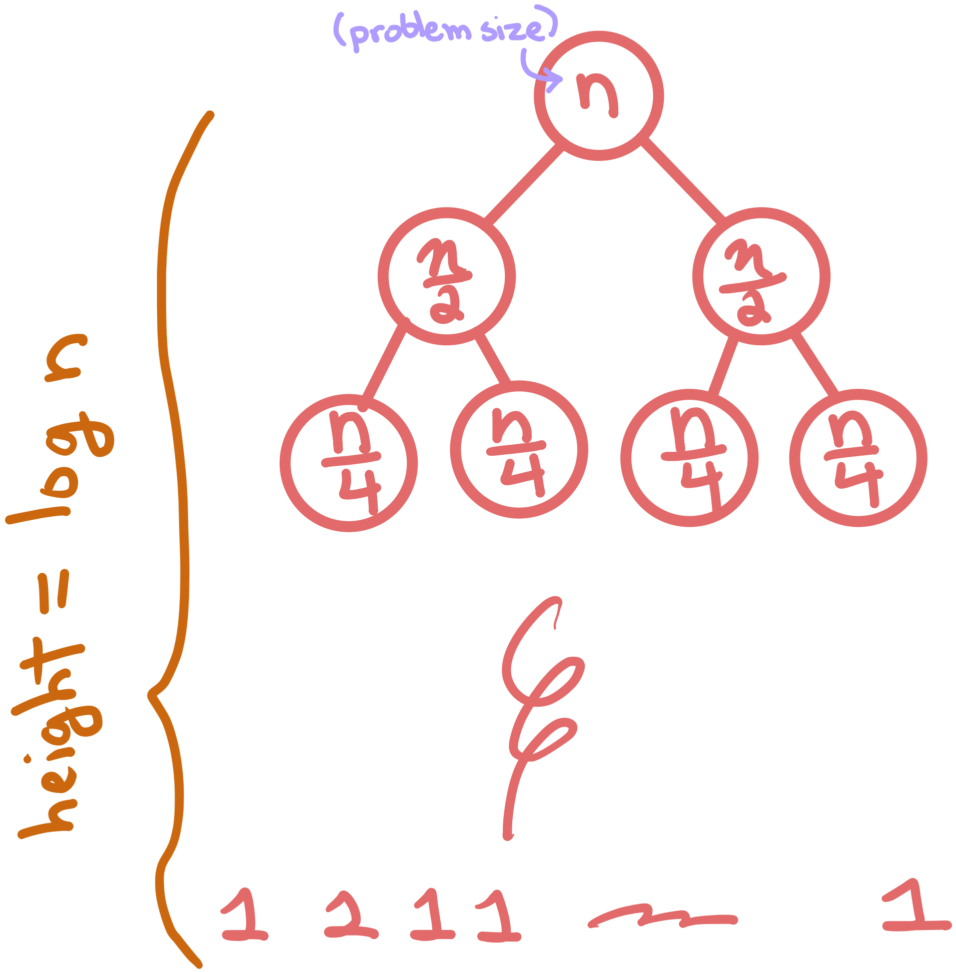
```
merge-sort(A[1..n])
/* In the base case, the list is so short there is nothing to do.          */
1. If n ≤ 1 then return A.                              ← Base case
/* In the general case, we divide the input in half and recursively sort each half. */
2. Let m = ⌊n/2⌋.
3. B1[1..m] ← merge-sort(A[1..m]).                      ← satisfies recursive spec
4. B2[1..n − m] ← merge-sort(A[m + 1..n]).                by induction on n
/* Now we merge the two sorted halves in linear time.                       */
5. return merge(B1, B2).
```



# Recursion tree

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

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




**Work**

$n$

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

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

$\vdots$

$2^k\cdot\frac{n}{2^k}=n$

$+\ n\cdot 1=n$

$n\times\log(n)$

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

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

$\underbrace{\Omega(f(n))}_{\text{"big-Omega"}}$ time

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

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 $\underbrace{\Omega(f(n))}_{\text{"big-Omega"}}$ time

(like $O(\cdot)$ 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

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

**Comparison model**

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

Goal lower bound # comparisons

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. $x_{i_k} < x_{j_k}$?

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

- 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

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

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

## Comparison model

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

Goal: lower bound # comparisons

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

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. $x_{i_k} < x_{j_k}$?

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

- output is a function of $k$ queries
- each query has 2 outcomes (YES/NO)

$\Rightarrow$ at most $2^k$ possible outputs

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

**<u>Theorem</u>** In the comparison (only) model, any (correct, determistic) sorting algorithm makes

$$\Omega(n \log n) \text{ comparisons}$$

in the worst case.

| Upper bound | Lower bound |
| --- | --- |
| $O(n \log n)$ (mergesort) | $\Omega(n \log n)$ |

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 $(a_1, b_1), \ldots, (a_n, b_n) \in \mathbb{R}^2$, 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,<br>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 | `tandon34` | Tuesday, 3:30–4:30PM |
| Madhav Valiyaparambil | `mvaliyap` | Wednesday, 5:30–6:30PM |
| Nathan Vijitbenjaronk | `wvijitbe` | Thursday, 9:30–10:30AM |
| Aaryan Wadhwani | `wadhwan2` | 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.

[1]TA office hours start in week 2.