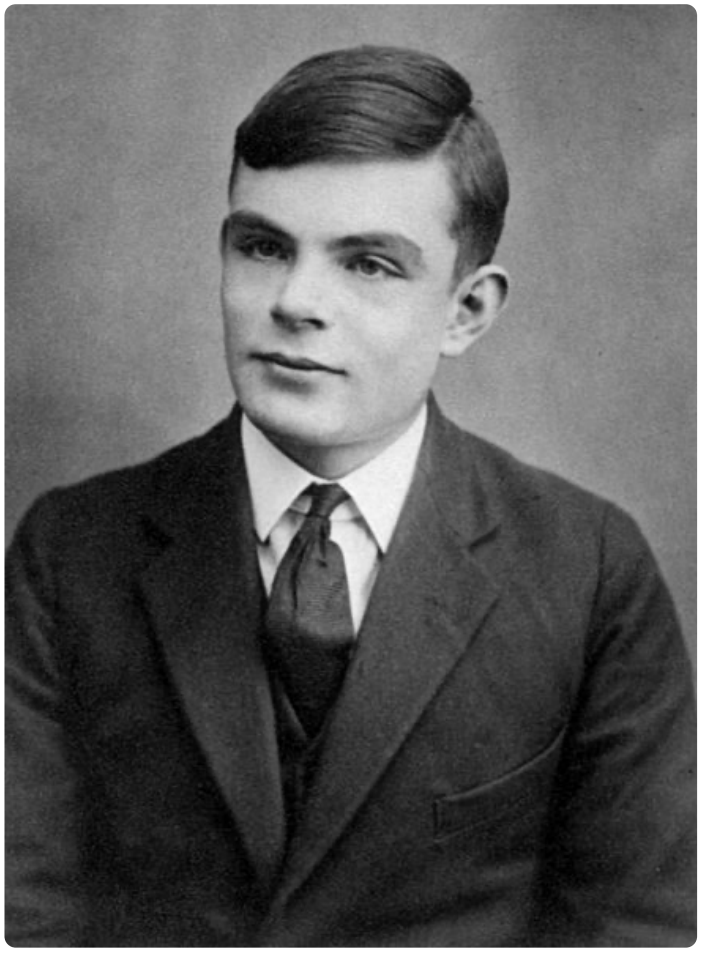
Ada Lovelace



Alan Turing

welcome to Graduate Algorithms

fundamentalalgorithms.com

one big .pdf w/ everything

Graduate Algorithms, CS580, Spring 2024

Lectures: 11:30AM to 12:20 PM; on Monday, Wednesday and Friday; in WALC 3087. *First two lectures are over Zoom.*

Staff:	Email (@purdue.edu)	Office hours
Kent Quanrud	krq	TBD at LWSN 1211[1]
Tiantian Gong (TA)	gong146	Tuesday, 11AM–12PM, at HAAS 143.
Haoyu Song (TA)	song522	Monday, 3–4PM, at HAAS 143.

Please see the syllabus (page 549) for more information.

Homework

- Homework 0 due January 17.
- Homework 1 due January 24.
- Homework 2 due January 31.
- Homework 3 due February 7.
- Homework 4 due February 21.
- Homework 5 due February 28.
- Homework 6 due March 6.
- Homework 7 due March 20.
- Homework 8 due April 3.
- Homework 9 due April 10.
- Homework 10 due April 17.

Class schedule[2]

1. *January 8.* Searching and sorting, including binary search, merge sort, and lower bounds for sorting (chapter 1). *On zoom:* https://purdue-edu.zoom.us/j/98283084796?pwd=MEE1RlhSdi9BUksyY3lCZWhzcXpNZz09

welcome to Graduate Algorithms

fundamentalalgorithms.com

Graduate Algorithms, CS580, Spring 2024

Lectures: 11:30AM to 12:20 PM; on Monday, Wednesday and Friday; in WALC 3087. *First two lectures are over Zoom.*

Staff:	Email (@purdue.edu)	Office hours
Kent Quanrud	krq	TBD at LWSN 1211[1]
Tiantian Gong (TA)	gong146	Tuesday, 11AM–12PM, at HAAS 143.
Haoyu Song (TA)	song522	Monday, 3–4PM, at HAAS 143.

Please see the syllabus (page 549) for more information.

Homework

- Homework 0 due January 17.
- Homework 1 due January 24.
- Homework 2 due January 31.
- Homework 3 due February 7.
- Homework 4 due February 21.
- Homework 5 due February 28.
- Homework 6 due March 6.
- Homework 7 due March 20.
- Homework 8 due April 3.
- Homework 9 due April 10.
- Homework 10 due April 17.

Class schedule[2]

1. *January 8.* Searching and sorting, including binary search, merge sort, and lower bounds for sorting (chapter 1). *On zoom:* https://purdue-edu.zoom.us/j/98283084796?pwd=MEE1RlhSdi9BUksyY3lCZWhzcXpNZz09

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

RHS much more efficient than LHS

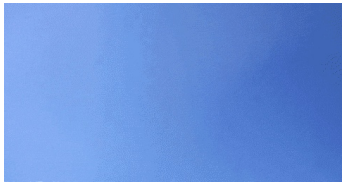
"unary"

$0, 1, 2, \ldots 10^{10}-1$

k

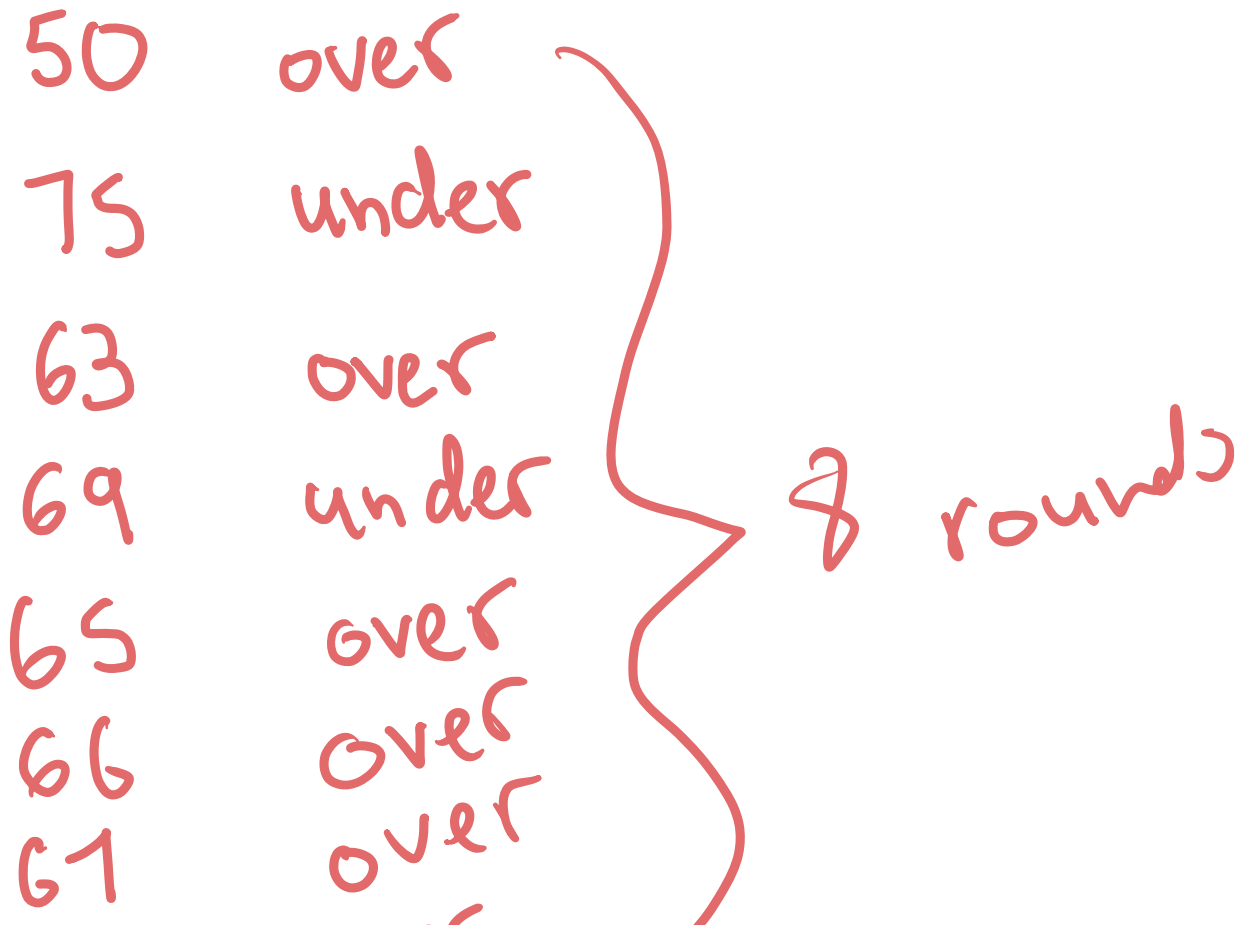
Decimal notation: 10^k #'s w/ k digits

Combinatorial explosion!



Over/under

guess # between 1 and 100

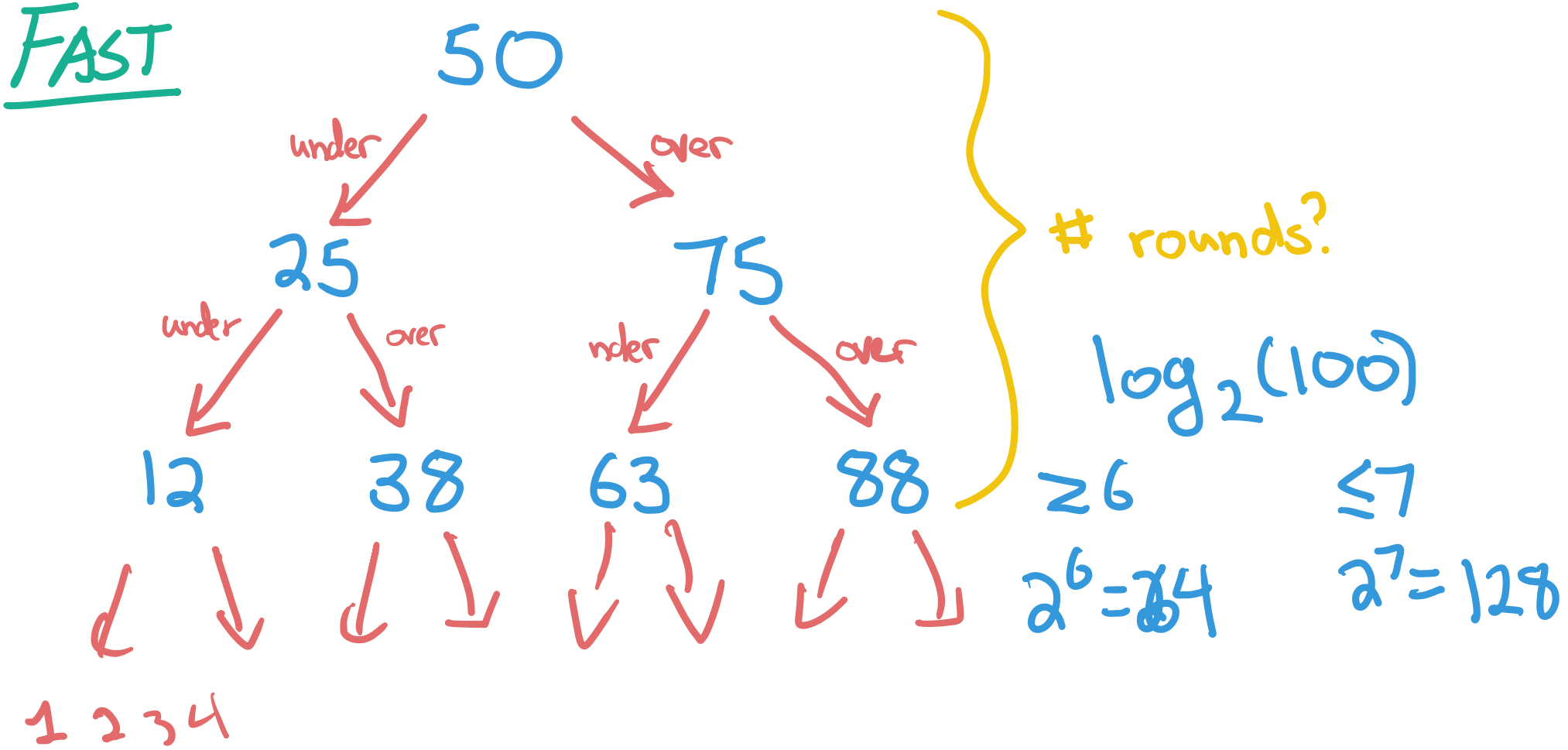


64 over

Slow algo:

for i=1, 2, ~, 100, guess i

FAST



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

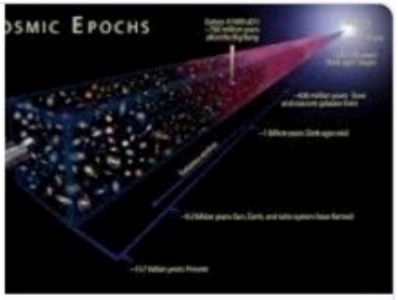
how many atoms in the universe

Sign in

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

$\log_2(\#\text{atoms}) = \log_2(10^{82}) = 82\log_2(10)$

≈ 272.39 ≈ 3.32

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

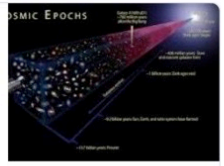
$$\approx \log_2(272.39)$$

$$\approx 8.089$$

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

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 JOBINTERVIEWQUICKSORT(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?
```

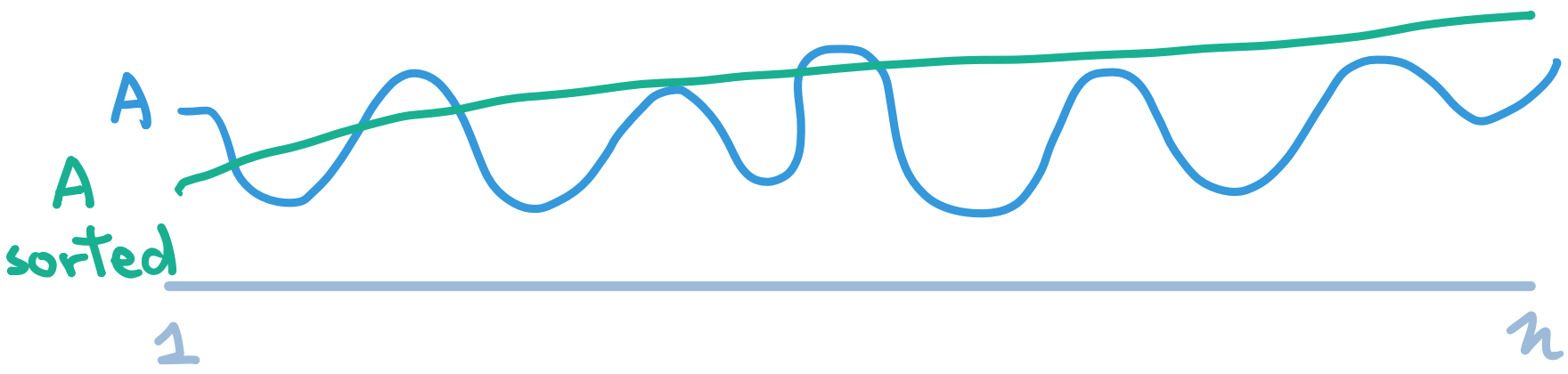
```
DEFINE PANICSORT(LIST):
    IF ISSORTED(LIST):
        RETURN LIST
    FOR N FROM 1 TO 10000:
        PIVOT = RANDOM(0, LENGTH(LIST))
        LIST = LIST[PIVOT:] + LIST[:PIVOT]
        IF ISSORTED(LIST):
            RETURN LIST
    IF ISSORTED(LIST):
        RETURN LIST:
    IF ISSORTED(LIST):  //THIS 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

$$\frac{n}{2} \times \frac{n}{2} \geq \frac{n^2}{4}$$

$$\overbrace{n + (n-1) + (n-2) + \frac{n}{2}}^{n/2} + 1 \geq \frac{n^2}{4}$$

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
x 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

$O(n)$ 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: verification

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

how long does it take to verify A is sorted?

$O(n)$

sorted? (A[1~n])
① for i=1,...,n-1
  ② 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?

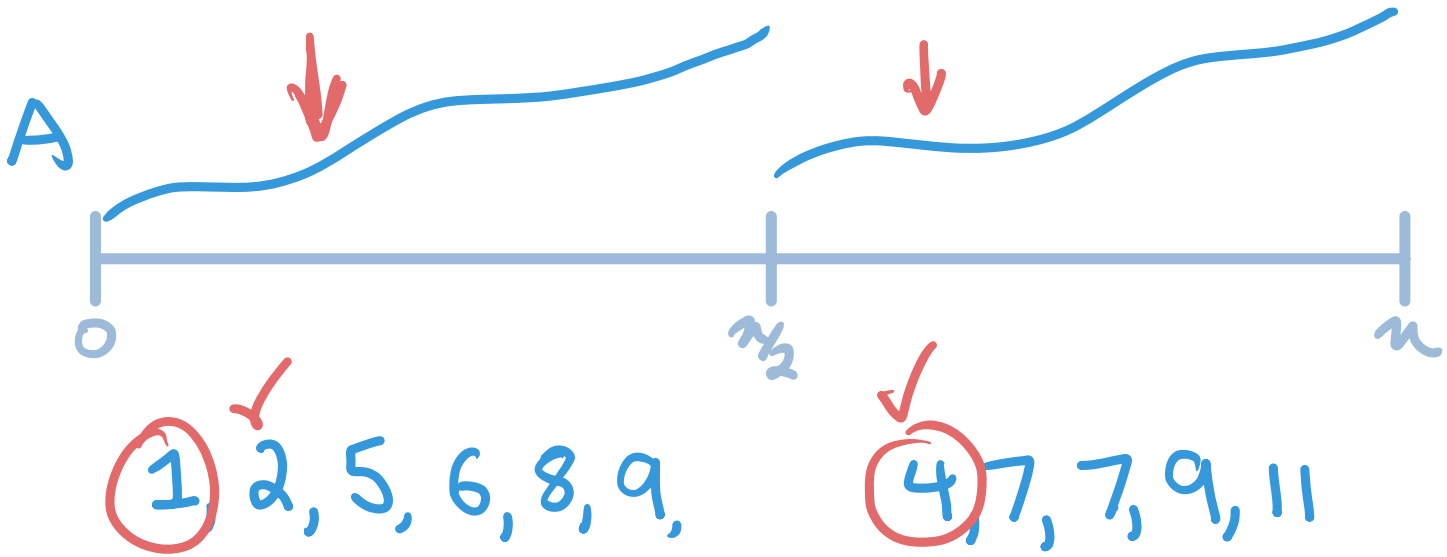
n^2 = each # is compared to every other #

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

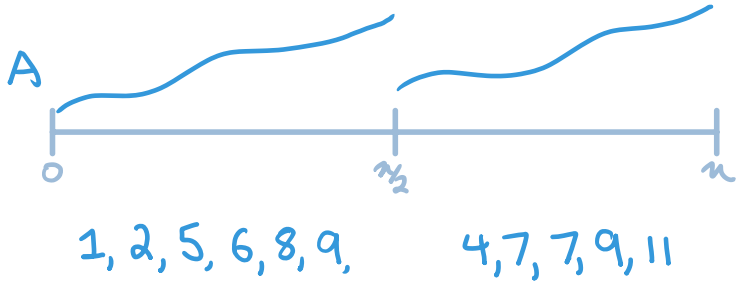
n iterations

O(1) time per iteration

O(n)

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. $C[i+j-1] \leftarrow B_2[j]$ 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

Sort($A[1,\ldots,n]$): Give n comparable elements $A[1,\ldots,n]$, returns an array of the n elements in sorted order

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

`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.

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.

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.

/ 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.

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. $B_1[1..m] \leftarrow$ merge-sort($A[1..m]$). ← satisfies recursive spec by induction on n
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).

③ Prove correctness

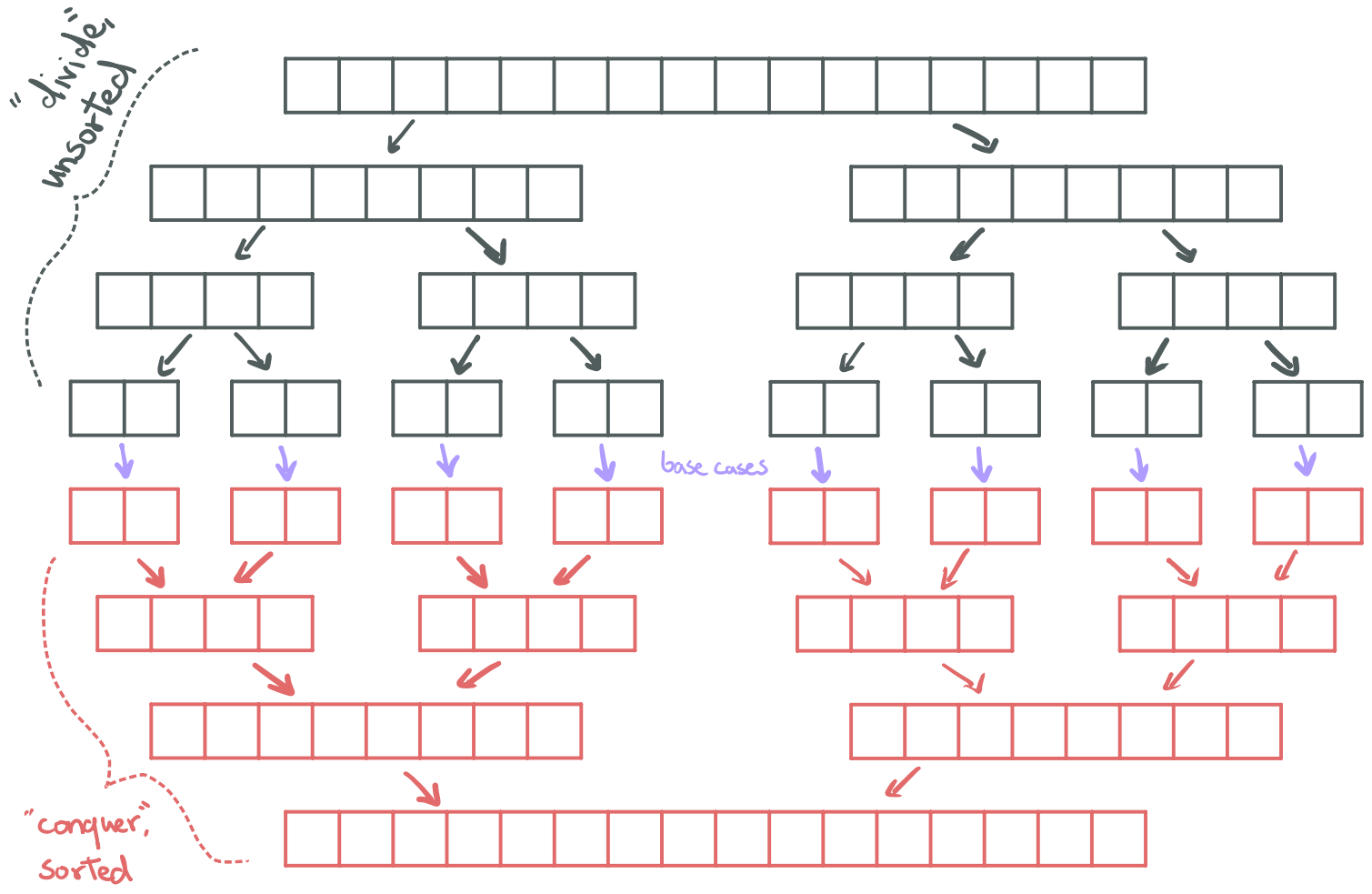
argue algo satisfies recursive spec for all n by induction on n

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 ≤ 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 by induction on n
4. B2[1..n − m] ← 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(B1, B2).
```



Running time analysis

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

$$T(n) = \begin{cases} O(1) & \text{if } n \le 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n) & n > 1 \end{cases}$$

$T(\lceil n/2 \rceil)$

$T(\lceil n/2 \rceil)$

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

$$\le 2T(n/2) + O(n)$$

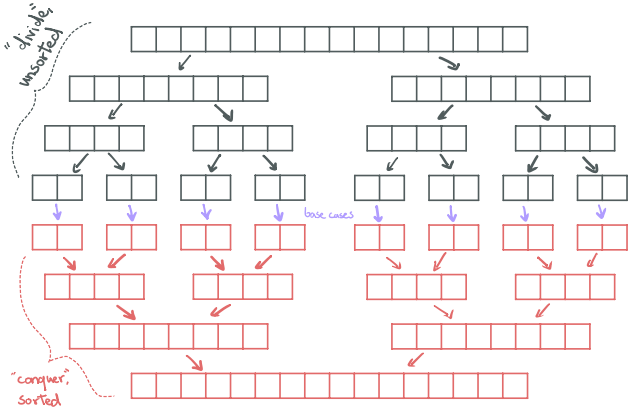
$T(n/2 + A)$

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.

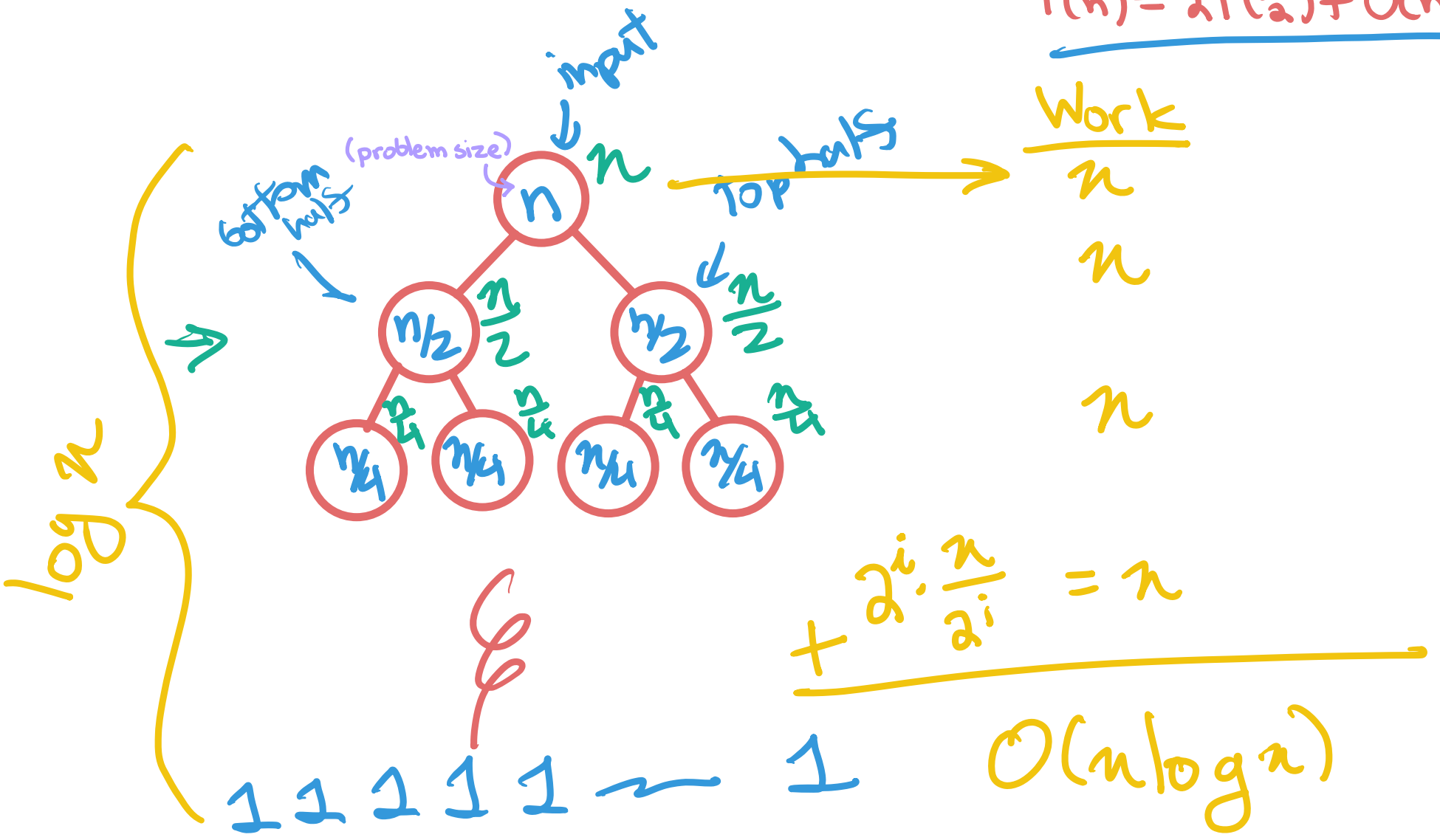
```
merge-sort(A[1..n])
/* In the base case, the list is so short there is nothing to do. */
1. If n ≤ 1 then return A.                                   ← Base case
/* In the general case, we divide the input in half and recursively sort each half. */
2. Let m = ⌊n/2⌋.                              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



Recursion tree

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

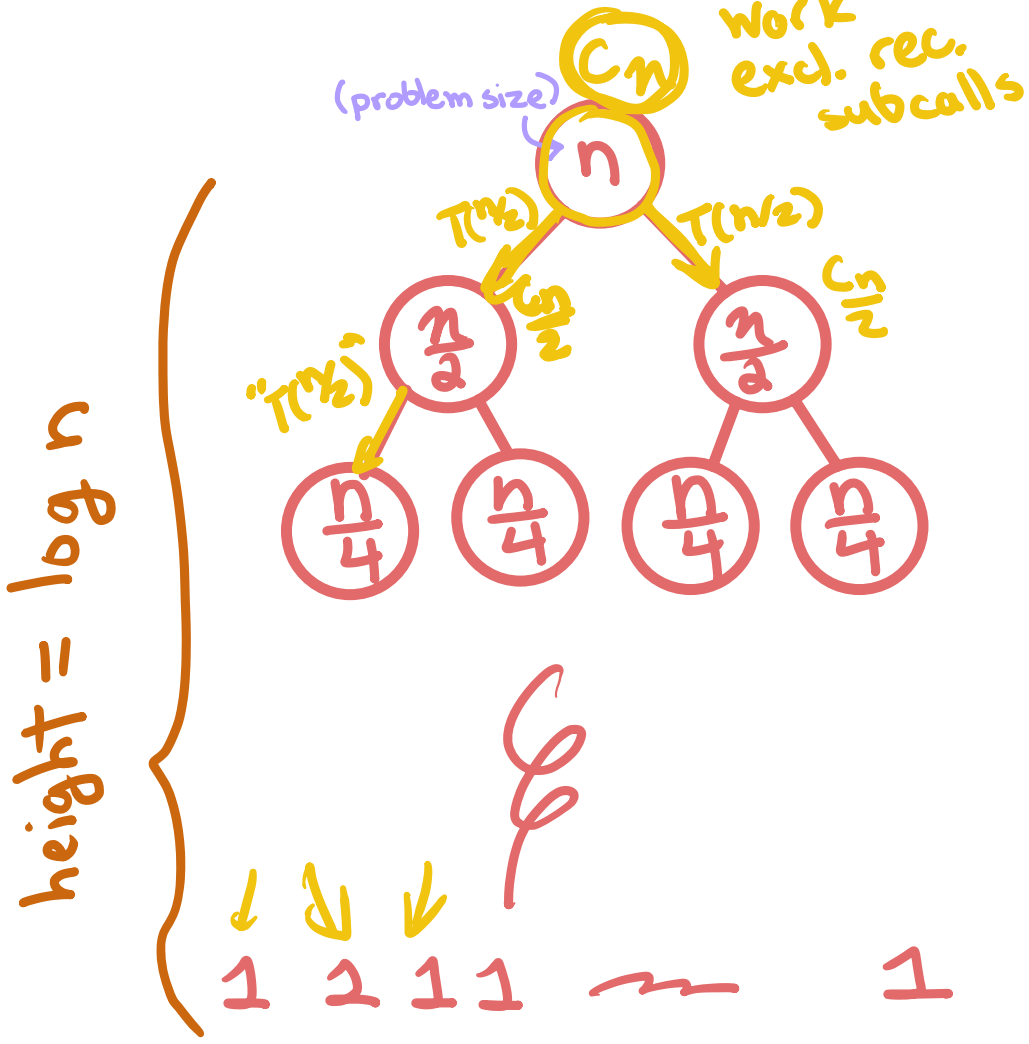
$T(n) = 2T(\frac{n}{2}) + O(n)$



Recursion tree

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

$T(n) = 2T(\frac{n}{2}) + cn$ (c ↓, cn: merge)



Work	
cn	cn
$\frac{cn}{2} + \frac{cn}{2} = cn$	cn
$4 \cdot (\frac{n}{4}) = n$	cn
$2^k \cdot \frac{n}{2^k} = n$	cn
$+ \; n \cdot 1 = n$	
$cn \times \log(n)$	
$= O(n \log n)$	

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

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

Comparison model

- Input: $x_1, \ldots, x_n$ 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 $\Omega(f(n))$ time

"big-Omega"

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

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. $x_{i_1} < x_{j_1}$?
2. $x_{i_2} < x_{j_2}$?
3. $x_{i_3} < x_{j_3}$?
4. $x_{i_4} < x_{j_4}$?

{

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

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

- output is a function of k queries
- each query has 2 outcomes

$\Rightarrow$ at most $\boxed{2^k}$ possible outputs

meanwhile...

$n!$ possible lists

$n! = n \cdot (n-1) \cdot (n-2) \cdots \underbrace{\geq \frac{n}{2}}_{n/2} \cdots 1$

$\boxed{2^k \geq n!}$

$k \geq \log_2(n!) \quad [n! \geq (\frac{n}{2})^{n/2}]$

$\geq \frac{n}{2} \log(\frac{n}{2})$

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

Comparison model

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

Goal lower bound # comparisons

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

1. $x_{i_1} < x_{j_1}$?
2. $x_{i_2} < x_{j_2}$?
3. $x_{i_3} < x_{j_3}$?
4. $x_{i_4} < x_{j_4}$?

$\}$

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

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

- 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	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) for all n

$O(n \log n)$ $\approx$ $\Omega(n \log n)$

[$O(n \log n)$] [$\Omega(n \log n)$]

some constant indep. of 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

Last compiled January 8, 2024 at 10:01am.

Graduate Algorithms, CS580, Spring 2024

Lectures: 11:30AM to 12:20 PM; on Monday, Wednesday and Friday; in WALC 3087. *First two lectures are over Zoom.*

Staff:	Email (@purdue.edu)	Office hours
Kent Quanrud	krq	TBD at LWSN 1211[1]
Tiantian Gong (TA)	gong146	Tuesday, 11AM–12PM, at HAAS 143.
Haoyu Song (TA)	song522	Monday, 3–4PM, at HAAS 143.

Please see the syllabus (page 549) for more information.

Homework

- Homework 0 due January 17.
- Homework 1 due January 24.
- Homework 2 due January 31.
- Homework 3 due February 7.
- Homework 4 due February 21.
- Homework 5 due February 28.
- Homework 6 due March 6.
- Homework 7 due March 20.
- Homework 8 due April 3.
- Homework 9 due April 10.
- Homework 10 due April 17.

Class schedule[2]

1. *January 8.* Searching and sorting, including binary search, merge sort, and lower bounds for sorting (chapter 1). *On zoom:* https://purdue-edu.zoom.us/j/98283084796?pwd=MEE1RlhSdi9BUksyY3lCZWhzcXpNZz09
2. *January 10.* Induction and recursion (chapter 2). *On zoom:* https://purdue-edu.zoom.us/j/93963400762?pwd=ZVY1bXd4MGVKVDdjZnlESkh2N2Izdz09
3. *January 12.* Catch-up day.

[1] Office hours canceled on the week of January 8 due to travel.

[2] All future dates are tentative. Lecture materials and video recordings can be found at the end of each chapter.