CS4102 Algorithms

Nate Brunelle

Spring 2018

Warm up

Show $\log(n!) = \Theta(n \log n)$

Hint: show $n! \leq n^n$

Hint 2: show $n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$

$\log n! = O(n \log n)$

$$
\begin{array}{ccccccccccccc}
n! & = & n & \cdot & (n-1) & \cdot & (n-2) & \cdot & \ldots & \cdot & 2 & \cdot & 1 \\
 & & \parallel & & \wedge & & \wedge & & & & \wedge & & \wedge \\
n^n & = & n & \cdot & n & \cdot & n & \cdot & \ldots & \cdot & n & \cdot & n
\end{array}
$$

$$
\begin{aligned}
& n! \leq n^n \\
\Rightarrow\ & \log(n!) \leq \log(n^n) \\
\Rightarrow\ & \log(n!) \leq n \log n \\
\Rightarrow\ & \log(n!) = O(n \log n)
\end{aligned}
$$

$\log n! = \Omega(n \log n)$

$$\begin{array}{ccccccccccccccc}
n! & = & n & \cdot & (n-1) & \cdot & (n-2) & \cdot \ldots \cdot & \frac{n}{2} & \cdot & \left(\frac{n}{2}-1\right) & \cdot \ldots \cdot & 2 & \cdot & 1 \\
 & & \vee & & \vee & & \vee & & \parallel & & \vee & & \vee & & \parallel \\
\left(\frac{n}{2}\right)^{\frac{n}{2}} & = & \frac{n}{2} & \cdot & \frac{n}{2} & \cdot & \frac{n}{2} & \cdot \ldots \cdot & \frac{n}{2} & \cdot & 1 & \cdot \ldots \cdot & 1 & \cdot & 1
\end{array}$$

$$n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

$$\Rightarrow \log(n!) \geq \log\left(\left(\frac{n}{2}\right)^{\frac{n}{2}}\right)$$

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

$$\Rightarrow \log(n!) = \Omega(n \log n)$$

Today’s Keywords

- Divide and Conquer
- Sorting
- Quicksort
- Decision Tree
- Worst case lower bound

CLRS Readings

- Chapter 7
- Chapter 8

Homeworks

- Hw3 Due 11pm Thursday Sept. 28
  - Divide and conquer
  - Written (use LaTeX!)

Partition (Divide step)

- Given: a list, a pivot value p

Start: unordered list

8	5	7	3	12	10	1	2	4	9	6	11

Goal: All elements $< p$ on left, all $> p$ on right

5	7	3	1	2	4	6	8	12	10	9	11

Is it worth it?

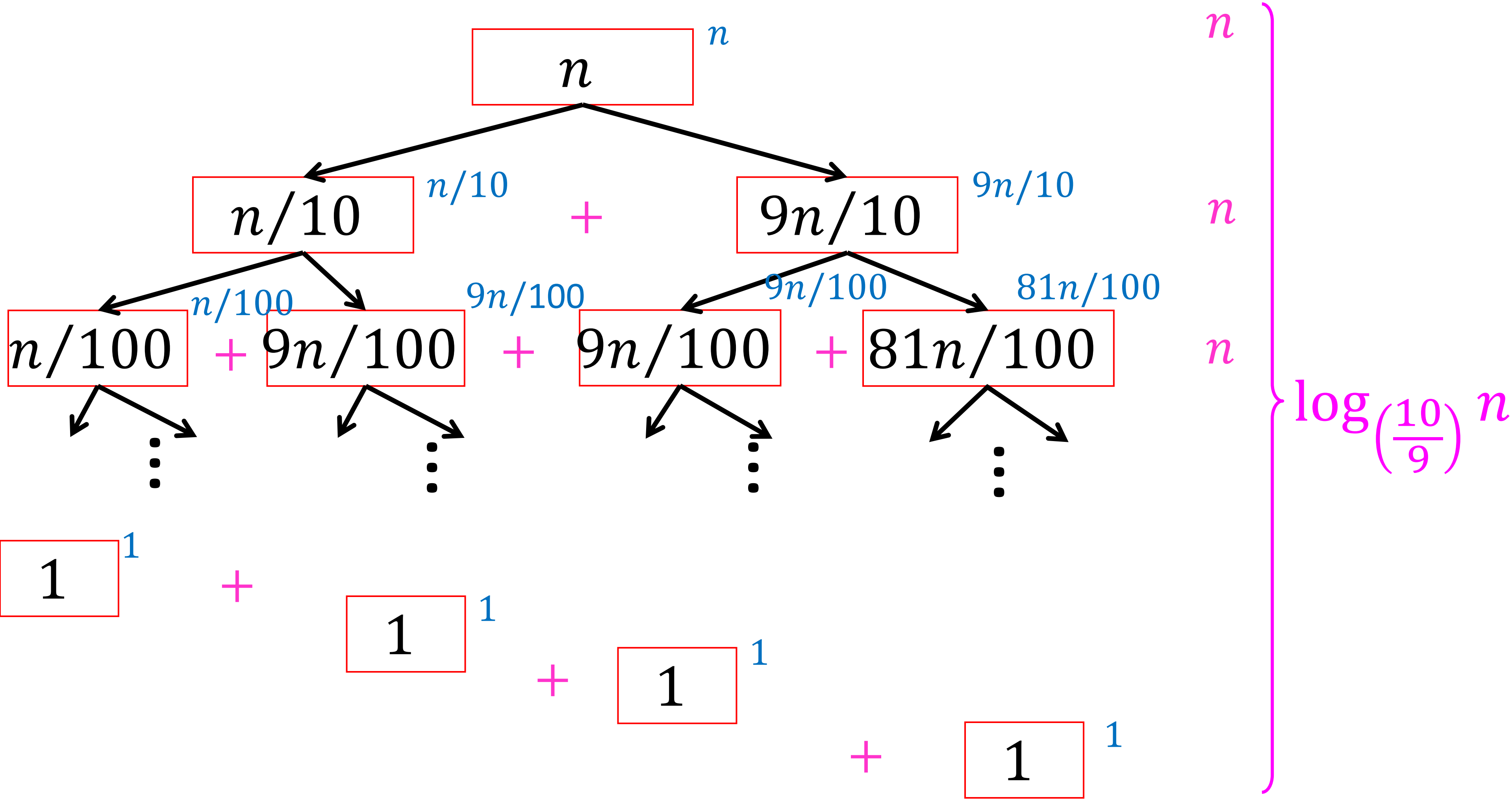
- Using Quickselect to pick median guarantees $\Theta(n \log n)$ run time
- Approach has very large constancts
  – If you really want $\Theta(n \log n)$, better off using MergeSort
- Better approach: Random pivot
  – Very small constant (very fast algorithm)
  – Expected to run in $\Theta(n \log n)$ time
    - Why? Unbalanced partitions are very unlikely

Quicksort Run Time

- If the partition is always $\frac{n}{10}$th order statistic:

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + n$$

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + n$$



Quicksort Run Time

- If the partition is always $\frac{n}{10}$th order statistic:

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + n$$

$$T(n) = \Theta(n \log n)$$

Quicksort Run Time

- If the partition is always d^{th} order statistic:

1	5	2	3	6	4	7	8	10	9	11	12

1	2	3	5	6	4	7	8	10	9	11	12

- Then we shorten by d each time

$$T(n) = T(n-d) + n$$

$$T(n) = O(n^2)$$

What's the probability of this occurring?

Probability of n^2 run time

We must consistently select partition from within the first d terms

Probability first partition is among d smallest: $\frac{d}{n}$

Probability second partition is among d smallest: $\frac{d}{n-d}$

Probability all partitions are among d smallest:

$$\frac{d}{n} \cdot \frac{d}{n-d} \cdot \frac{d}{n-2d} \cdot \ldots \cdot \frac{d}{2d} \cdot 1 = \frac{1}{\left(\frac{n}{d}\right)!}$$

Quicksort

- Idea: pick a pivot element, recursively sort two sublists around that element
- Divide: select an element p, Partition(p)
- Conquer: recursively sort left and right sublists
- Combine: Nothing!

Random Pivot

- Using Quickselect to pick median guarantees $\Theta(n \log n)$ run time
- Approach has very large constants
  - If you really want $\Theta(n \log n)$, better off using MergeSort
- Better approach: Random pivot
  - Very small constant (very fast algorithm)
  - Expected to run in $\Theta(n \log n)$ time
    - Why? Unbalanced partitions are very unlikely

Formal Argument for $n \log n$ Average

- Remember, run time counts comparisons!
- Quicksort only compares against the pivot
  - Element i only compared to element j if one of them was the pivot

Partition (Divide step)

- Given: a list, a pivot value p

Start: unordered list

8	5	7	3	12	10	1	2	4	9	6	11

Goal: All elements $< p$ on left, all $> p$ on right

5	7	3	1	2	4	6	8	12	10	9	11

Formal Argument for $n \log n$ Average

- What is the probability of comparing two given elements?

1	2	3	4	5	6	7	8	9	10	11	12

- (Probability of comparing 3 and 4) = 1
  - Why? Otherwise I wouldn't know which came first
  - ANY sorting algorithm must compare adjacent elements

Formal Argument for $n \log n$ Average

- What is the probability of comparing two given elements?

1	2	3	4	5	6	7	8	9	10	11	12

- (Probability of comparing 1 and 12) = $\frac{2}{12}$
  - Why?
    - I only compare 1 with 12 if either was chosen as the first pivot
    - Otherwise they would be divided into opposite sublists

Formal Argument for $n \log n$ Average

- Probability of comparing i and j (where $j > i$):
  - dependent on the number of elements between i and j
    - $\frac{2}{j-i+1}$
- Expected (average) number of comparisons:
    - $\sum_{i<j} \frac{2}{j-i+1}$

Expected number of Comparisons

Consider when $i = 1$

$$\sum_{i<j} \frac{2}{j - i + 1}$$

1	2	3	4	5	6	7	8	9	10	11	12

Compared if 1 or 2 are chosen as partition
(these will always be compared)

Sum so far: $\frac{2}{2}$

Expected number of Comparisons

$$\sum_{i<j} \frac{2}{j-i+1}$$

Consider when $i = 1$

1	2	3	4	5	6	7	8	9	10	11	12

Compared if 1 or 3 are chosen as partition
(but not if 2 is ever chosen)

Sum so far: $\frac{2}{2} + \frac{2}{3}$

Expected number of Comparisons

$$\sum_{i<j} \frac{2}{j-i+1}$$

Consider when $i = 1$

1	2	3	4	5	6	7	8	9	10	11	12

Compared if 1 or 4 are chosen as partition
(but not if 2 or 3 are chosen)

Sum so far: $\frac{2}{2} + \frac{2}{3} + \frac{2}{4}$

Expected number of Comparisons

$$\sum_{i<j} \frac{2}{j-i+1}$$

Consider when $i = 1$

1	2	3	4	5	6	7	8	9	10	11	12

Compared if 1 or 12 are chosen as partition
(but not if 2 -> 11 are chosen)

Overall sum: $\frac{2}{2} + \frac{2}{3} + \frac{2}{4} + \frac{2}{5} + \cdots + \frac{2}{n}$

Expected number of Comparisons

$$\sum_{i<j} \frac{2}{j-i+1}$$

When $i = 1$: $2\left(\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n}\right)$

n terms overall

$$\sum_{i<j} \frac{2}{j-i+1} \leq 2n \boxed{\left(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right)} \quad \Theta(\log n)$$

Quicksort overall: expected $\Theta(n \log n)$

Sorting, so far

- Sorting algorithms we have discussed:
  - Mergesort $O(n \log n)$
  - Quicksort $O(n \log n)$
- Other sorting algorithms (will discuss):
  - Bubblesort $O(n^2)$
  - Insertionsort $O(n^2)$
  - Heapsort $O(n \log n)$

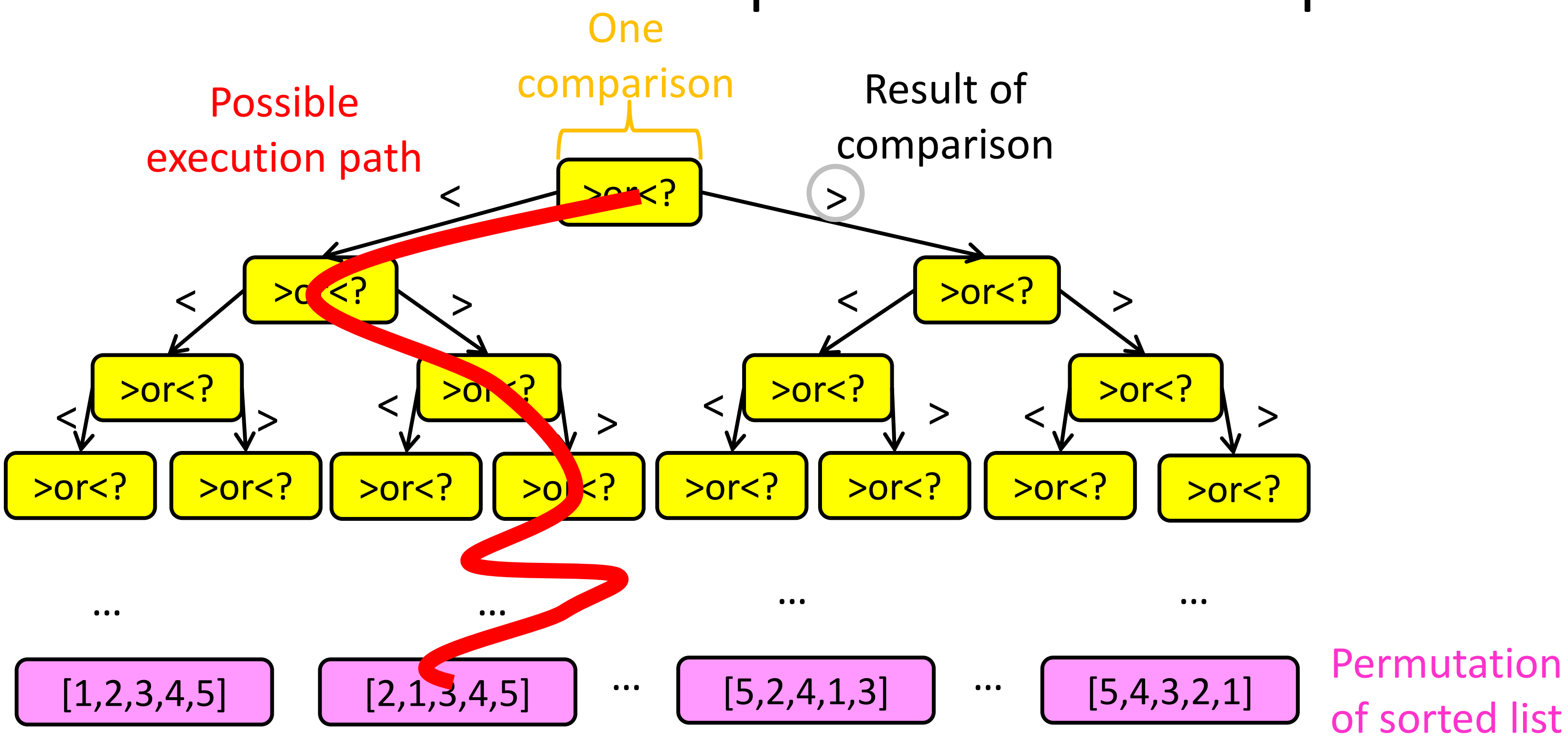
Can we do better than $O(n \log n)$?

Worst Case Lower Bounds

- Prove that there is no algorithm which can sort faster than $O(n \log n)$
- Non-existence proof!
  - Very hard to do

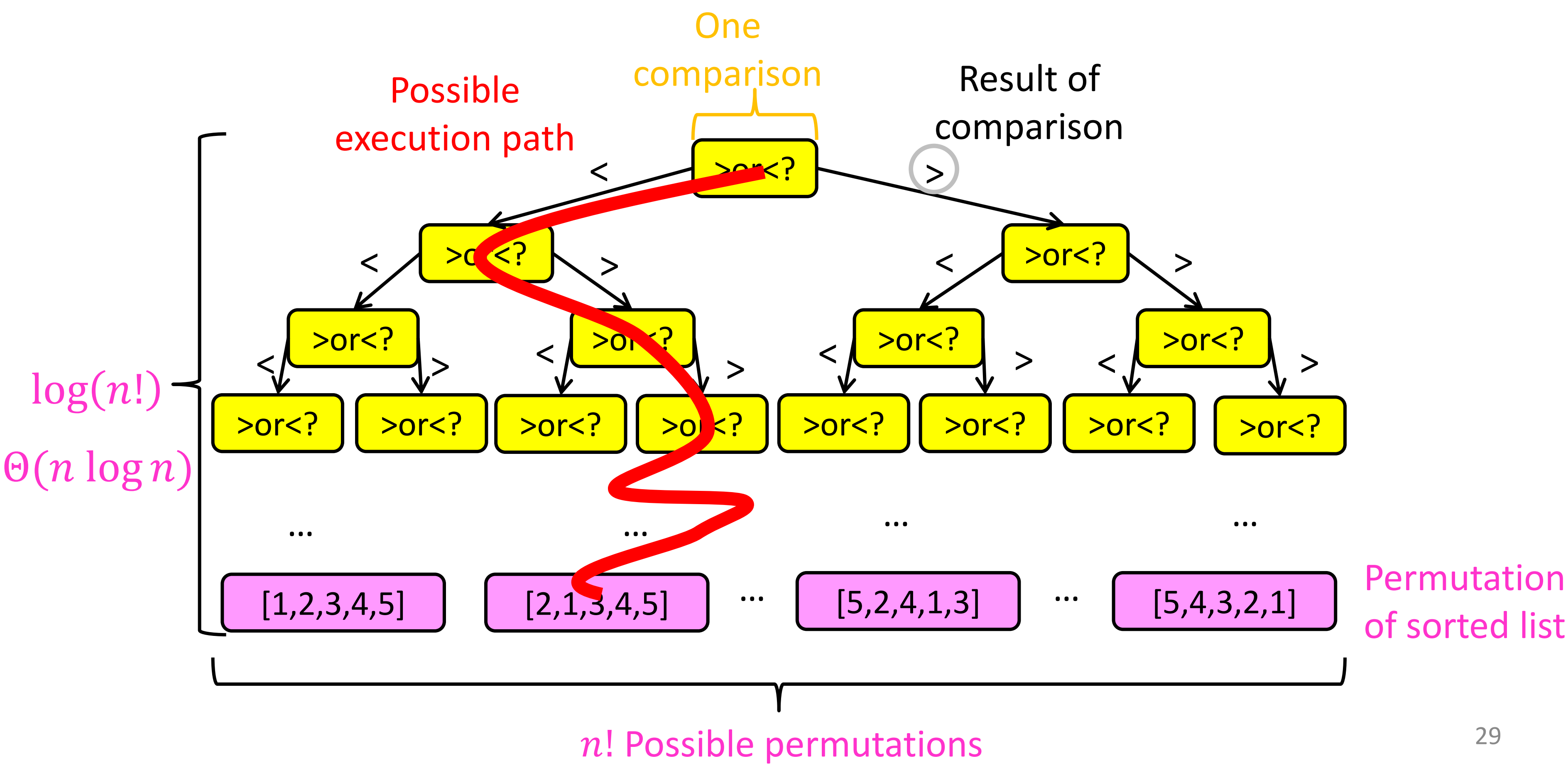
Strategy: Decision Tree

- Sorting algorithms use comparisons to figure out the order of input elements
- Draw tree to illustrate all possible execution paths



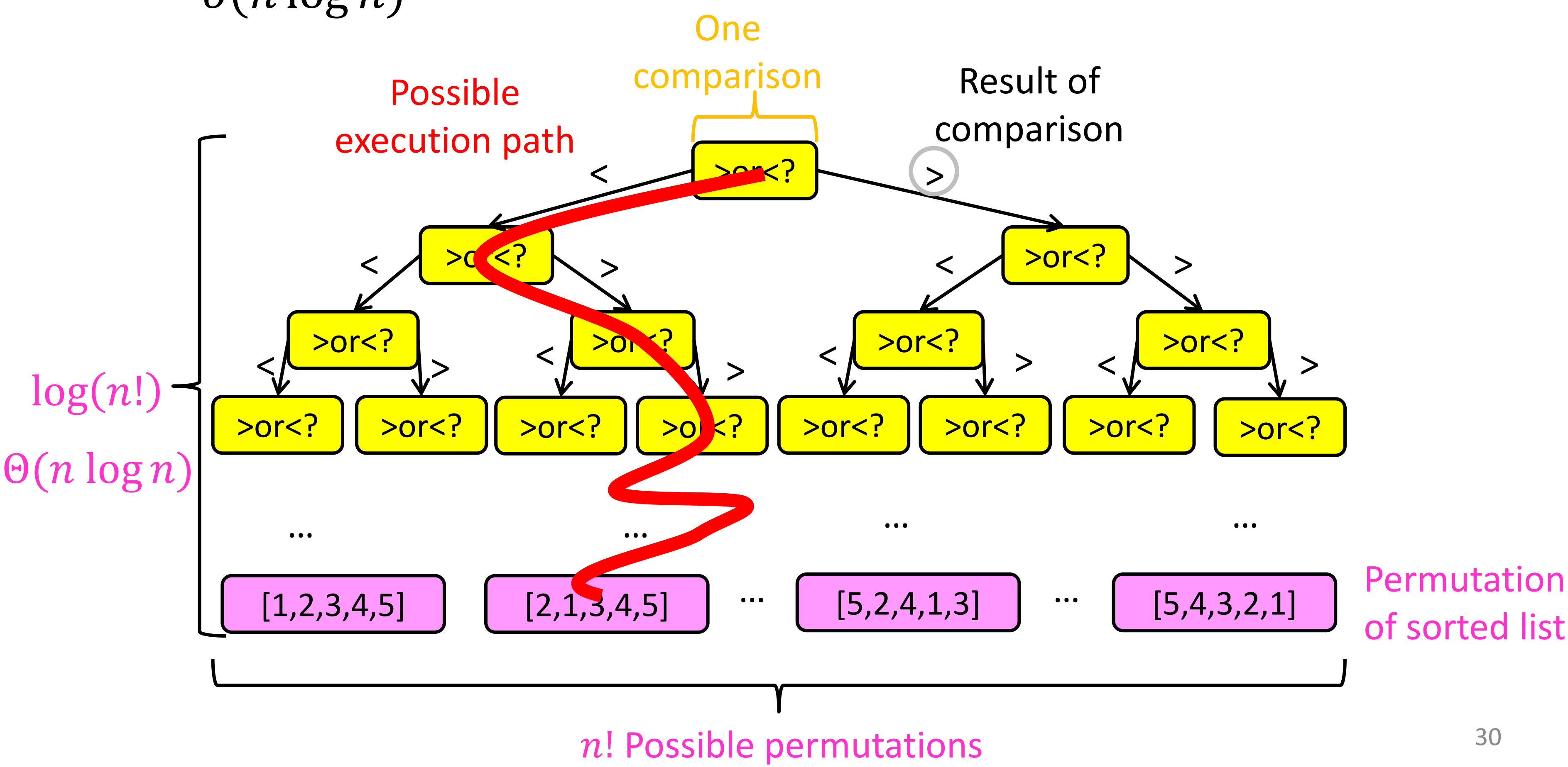
Strategy: Decision Tree

- Worst case run time is the longest execution path
- i.e., “height” of the decision tree



Strategy: Decision Tree

- Conclusion: Worst Case Optimal run time of sorting is $\Theta(n \log n)$
  - There is no (comparison-based) sorting algorithm with run time $o(n \log n)$



Sorting, so far

- Sorting algorithms we have discussed:
  - Mergesort $O(n \log n)$ Optimal!
  - Quicksort $O(n \log n)$ Optimal!
- Other sorting algorithms
  - Bubblesort $O(n^2)$
  - Insertionsort $O(n^2)$
  - Heapsort $O(n \log n)$ Optimal!

Speed Isn't Everying

- Important properties of sorting algorithms:
- Run Time
  - Asymptotic Complexity
  - Constants
- In Place (or In-Situ)
  - Done with only constant additional space
- Adaptive
  - Faster if list is nearly sorted
- Stable
  - Equal elements remain in original order
- Parallelizable
  - Runs faster with many computers

Mergesort

- **Divide**:
  - Break n-element list into two lists of $^n/_2$ elements
- **Conquer**:
  - If $n > 1$: Sort each sublist recursively
  - If $n = 1$: List is already sorted (base case)
- **Combine**:
  - Merge together sorted sublists into one sorted list

Run Time?

$\Theta(n \log n)$

Optimal!

In Place?

No

Adaptive?

No

Stable?

Yes!
(usually)

Merge

- **Combine**: Merge sorted sublists into one sorted list
- We have:
  - 2 sorted lists (L_1, L_2)
  - 1 output list (L_{out})

Adaptive:
If elements are equal, leftmost comes first

While (L_1 and L_2 not empty):

    If $L_1[0] \leq L_2[0]$:
        L_{out}.append(L_1.pop())
    Else:
        L_{out}.append(L_2.pop())

L_{out}.append(L_1)

L_{out}.append(L_2)

Mergesort

- **Divide**:
  - Break n-element list into two lists of $n/_2$ elements
- **Conquer**:
  - If $n > 1$: Sort each sublist recursively
  - If $n = 1$: List is already sorted (base case)
- **Combine**:
  - Merge together sorted sublists into one sorted list

Run Time?

$\Theta(n \log n)$

Optimal!

In Place?	Adaptive?	Stable?	Parallelizable?
No	No	Yes! (usually)	Yes!

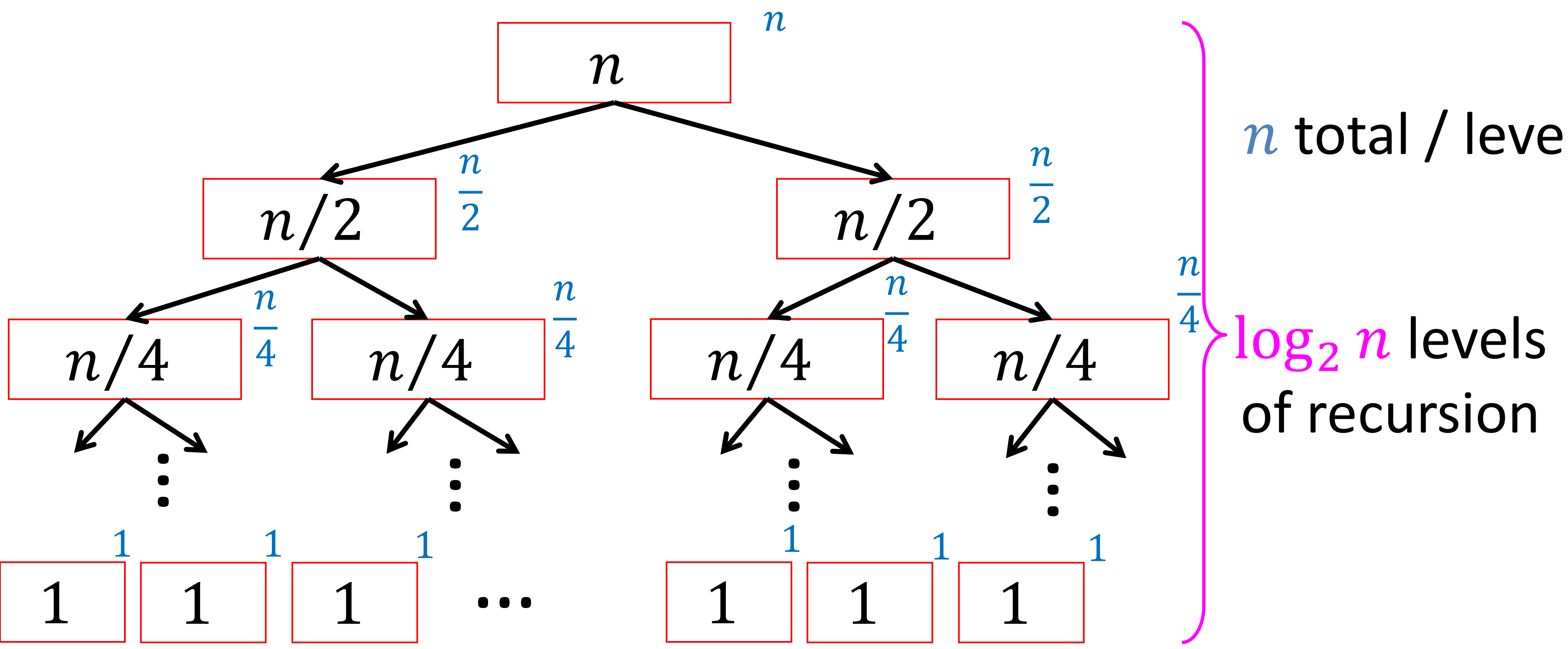
Parallelizable:
Allow different machines to work on each sublist

Mergesort

- **Divide**:
  - Break n-element list into two lists of ${}^{n}/_{2}$ elements
- **Conquer**:
  - If $n > 1$:
    - Sort each sublist recursively
  - If $n = 1$:
    - List is already sorted (base case)
- **Combine**:
  - Merge together sorted sublists into one sorted list

Mergesort (Sequential)

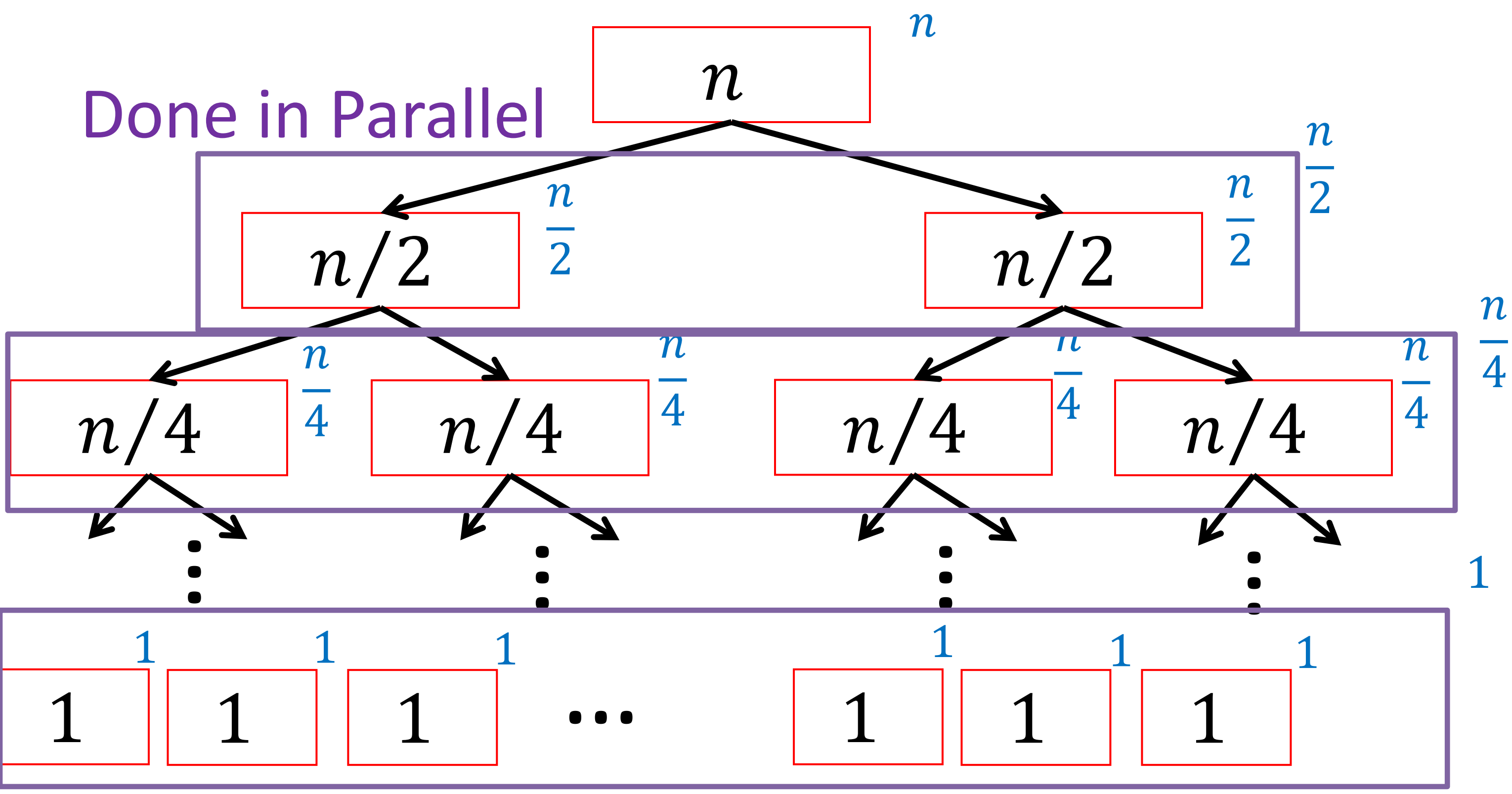
$$T(n) = 2T(\frac{n}{2}) + n$$



Run Time: $\Theta(n \log n)$

Mergesort (Parallel)

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



Run Time: $\Theta(\log n)$

Quicksort

- Idea: pick a partition element, recursively sort two sublists around that element
- Divide: select an element p, Partition(p)
- Conquer: recursively sort left and right sublists
- Combine: Nothing!

Run Time?

$\Theta(n \log n)$

Optimal!

(almost always)

In Place?	Adaptive?	Stable?	Parallelizable?
No...	No!	No	Yes!