CS 3100
Data Structures and Algorithms 2
Lecture 11: D&C: Median of Medians

Co-instructors: Robbie Hott and Ray Pettit

Spring 2024

Readings in CLRS 4th edition:

- Section 4.5

Announcements

- PA2 due next Friday, March 1, 2024
- Quizzes 1-2 coming February 29, 2024
  - Both quizzes taken the same day
  - Information on our class website
  - If you have SDAC, please schedule for 1 exam (*not a quiz*)
- Office hours
  - Prof Hott Office Hours: Mondays 11a-12p, Fridays 10-11a and 2-3p
  - Prof Pettit Office Hours: Mondays and Fridays 2:30-4:00p
  - TA office hours posted on our website
  - Office hours are not for "checking solutions"

Divide and Conquer

[CLRS Chapter 4]

Divide:

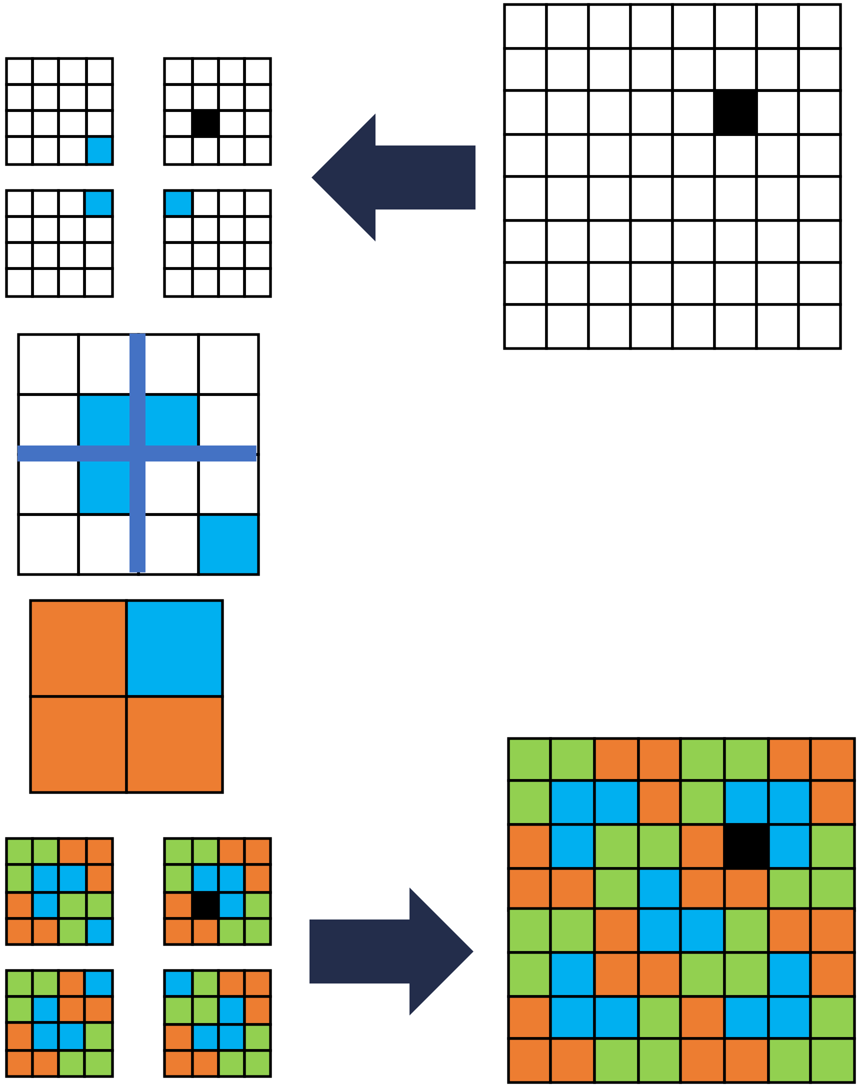
- Break the problem into multiple subproblems, each smaller instances of the original

Conquer:

- If the suproblems are "large":
  - Solve each subproblem recursively
- If the subproblems are "small":
  - Solve them directly (base case)

Combine:

- Merge solutions to subproblems to obtain solution for original problem



Quicksort

Like Mergesort:

- Divide and conquer algorithm
- $O(n \log n)$ run time (on expectation)

Unlike Mergesort:

- **Divide** step is the hard part
- Typically faster than Mergesort (often is the basis of sorting algorithms in standard library implementations)

Quicksort

General idea: choose a pivot element, recursively sort two sublists around that element

Divide: select pivot element p, Partition(p)

Conquer: recursively sort left and right sublists

Combine: nothing!

Partition Procedure (Divide Step)

Input: an unordered list, a pivot p

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

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

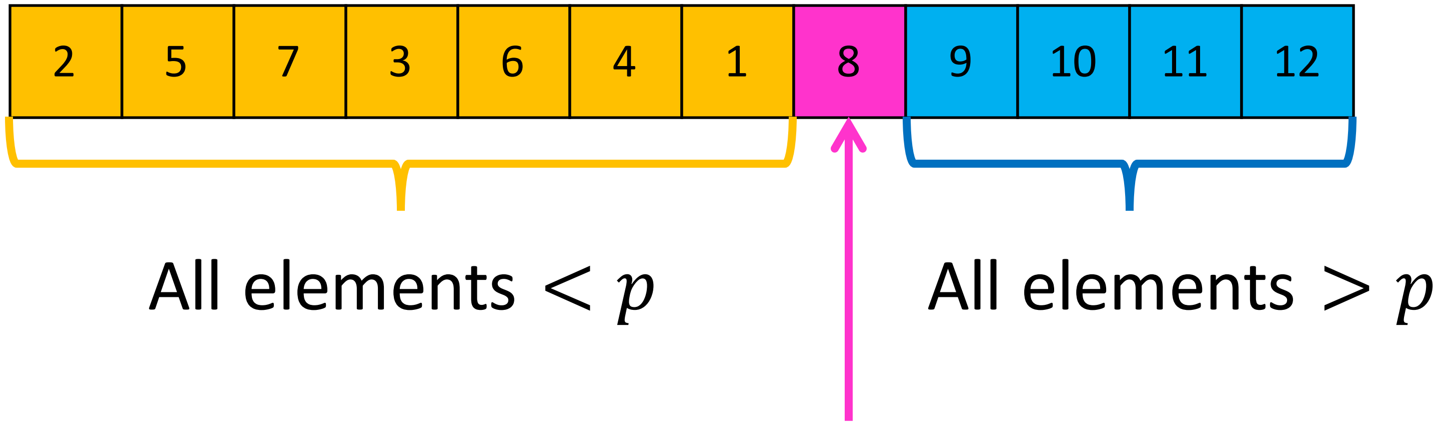
5	7	3	1	2	4	6	8	12	10	9	11

Partition Procedure Summary

1. Choose the pivot p to be the first element of the list
2. Initialize two pointers Begin (just after p), and End (at end of list)
3. While Begin < End:
   - If value of Begin $<\ p$, advance Begin to the right
   - Otherwise, swap value of Begin value with value of End value, and advance End to the left
4. If pointers meet at element $< p$: swap p with pointer position
5. Otherwise, if pointers meet at element $> p$: swap p with value to the left

Run time? $\Theta(n)$

Conquer Step



Exactly where it belongs!

Recursively sort Left and Right sublists

Quicksort Run Time (Optimistic)

If the pivot is the median:

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

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

Then we divide in half each time

$$T(n) = 2T(n/2) + n = \Theta(n \log n)$$

Quicksort Run Time (Worst-Case)

If the pivot is the extreme (min/max):

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 1 each time

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= n + (n-1) + \cdots + 2 + 1 \\ &= \frac{n(n+1)}{2} = \Theta(n^2) \end{aligned}$$

Good Pivot

What makes a good pivot?

- Roughly even split between left and right
- Ideally: median

Can we find median in linear time?

- Yes! Quickselect algorithm

Quickselect Algorithm

Algorithm to compute the i^{th} order statistic

- i^{th} smallest element in the list
- 1^{st} order statistic: minimum
- n^{th} order statistic: maximum
- $(n/2)^{\text{th}}$ order statistic: median

Quickselect Algorithm

Finds i^{th} order statistic

General idea: choose a pivot element, partition around the pivot, and recurse on sublist containing index i

Divide: select pivot element p, Partition(p)

Conquer:

- if $i =$ index of p, then we are done and return p
- if $i <$ index of p recurse left. Otherwise, recurse right

Combine: Nothing!

Partition Procedure (Divide Step)

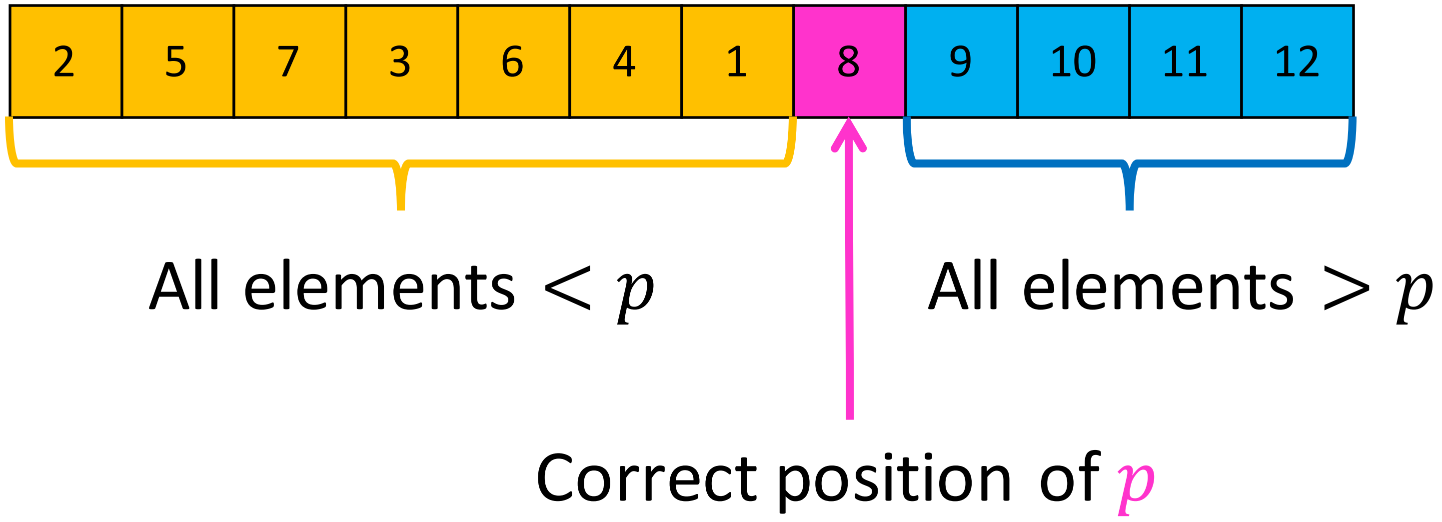
Input: an unordered list, a pivot p

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

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

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

Conquer Step



Recurse on sublist that contains index i

(add index of the pivot to i if recursing right)

Quickselect Run Time

If the pivot is always the median:

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

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

Then we divide in half each time

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

$$S(n) = O(n)$$

Quickselect Run Time

If the partition is always unbalanced:

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 1 each time

$$S(n) = S(n-1) + n$$

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

How to Choose the Pivot?

Good choice: $\Theta(n)$

Bad choice: $\Theta(n^2)$

Good Pivot

What makes a good pivot?

- Roughly even split between left and right
- Ideally: median

But this is the problem that Quickselect is supposed to solve!

Déjà vu?

What's next: an algorithm for choosing a "decent" pivot (median of medians)

Good Pivot for Quickselect

What makes a good Pivot for Quickselect?

- Roughly even split between left and right
- Ideally: median

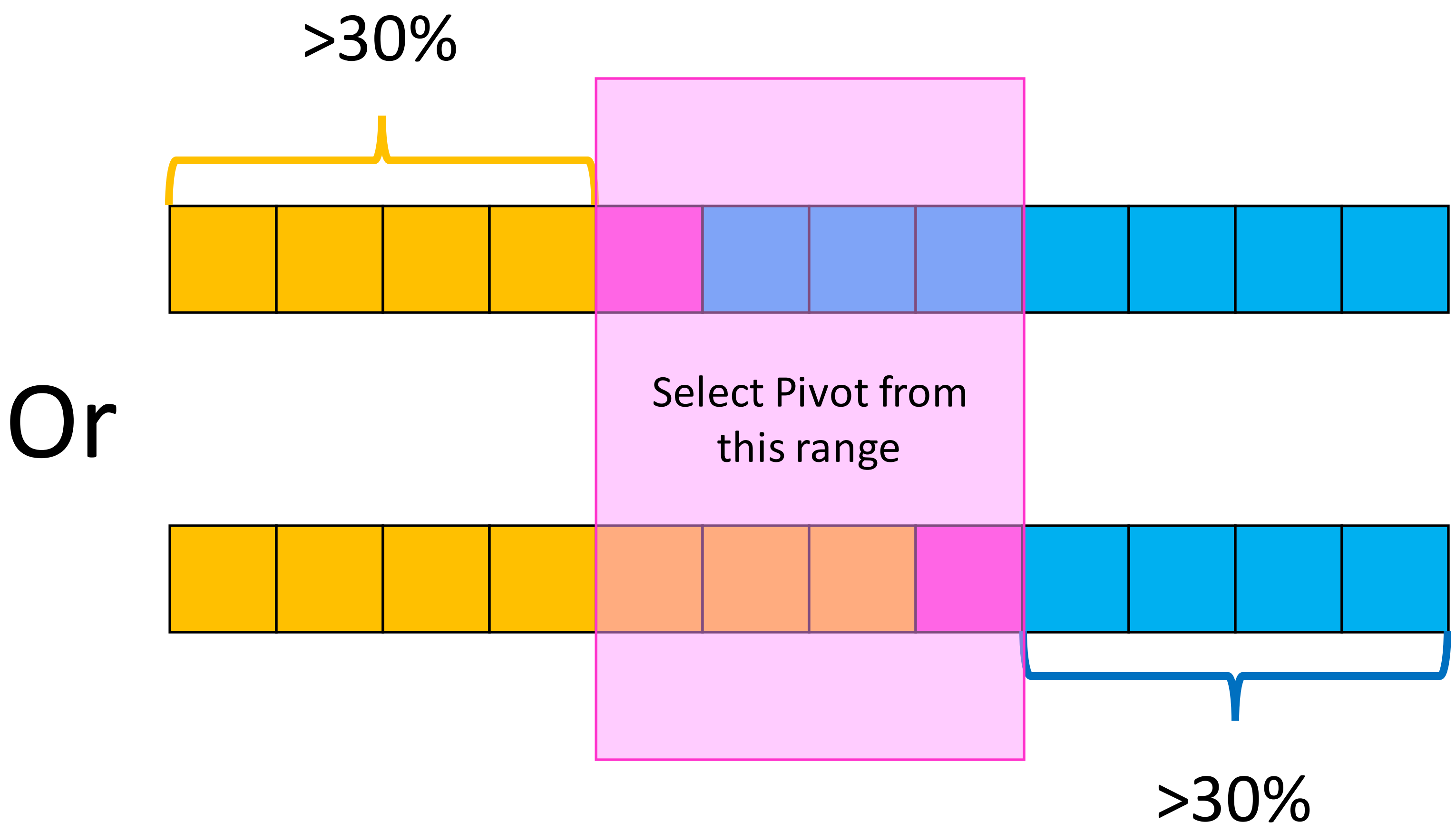
Déjà vu?

Here's what's next:

- First, **median of medians** algorithm
  - Finds something close to the median in $\Theta(n)$ time
- Second, we can prove that when its result used with Quickselect's partition, then Quickselect is guaranteed $\Theta(n)$
  - Because we now have a $\Theta(n)$ way to find the median, this guarantees Quicksort will be $\Theta(n \lg n)$
- Notes:
  - We have to do all this for every call to Partition in Quicksort
  - We could just use the value returned by median of medians for Quicksort's Partition

Good Pivot

Decent pivot: both sides of Pivot >30%



Median of Medians

Fast way to select a "good" pivot

Guarantees pivot is greater than ≈30% of elements and less than ≈30% of the elements

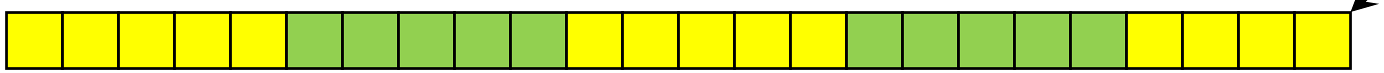
- I.e. it's in the middle 40% (±20% of the true median)

Main idea: break list into blocks, find the median of each blocks, use the median of those medians

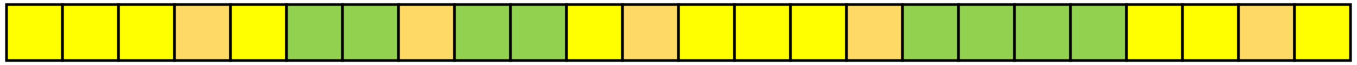
Median of Medians

1. Break list into chunks of size 5

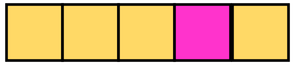
List could be long, many more than 5 chunks!



2. Find the median of each chunk
(using insertion sort: n=5, max 20 comparisons per chunk)

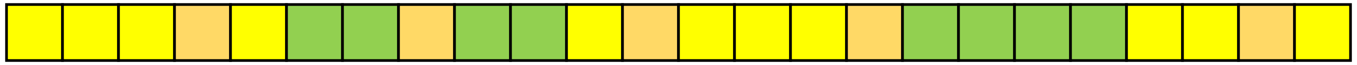


3. Return median of medians (using Quickselect, this algorithm, called recursively, on list of medians)



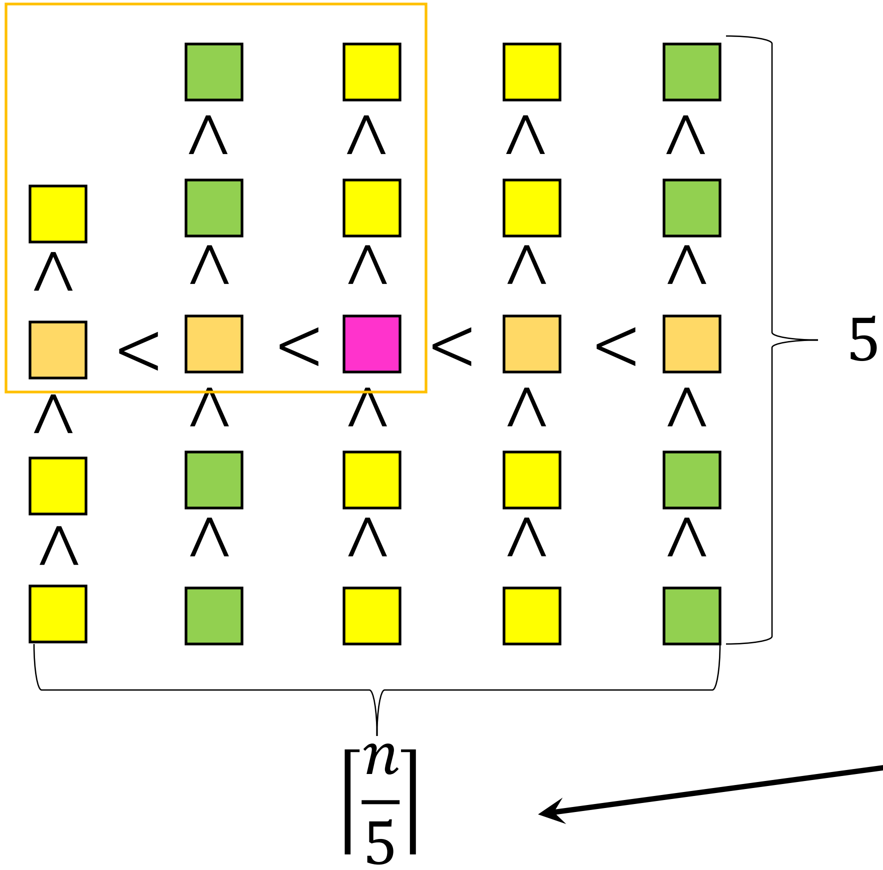
List could be long, many more than 5 medians!

Why is this good?



Each chunk sorted, chunks ordered by their medians

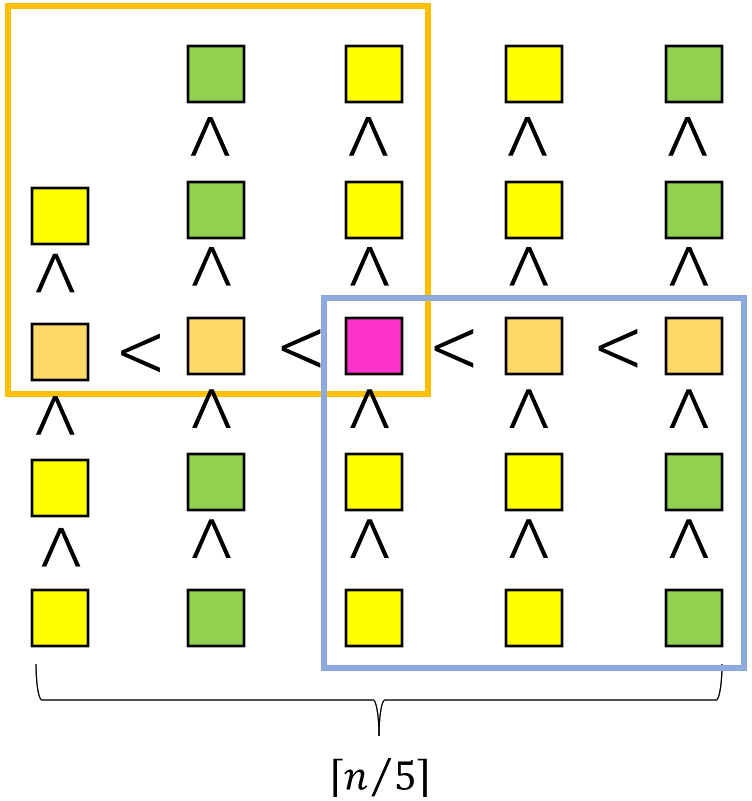
MedianofMedians is Greater than all of these



List could be long, so not a small number!

Why is this good?

MedianofMedians is larger than all of these



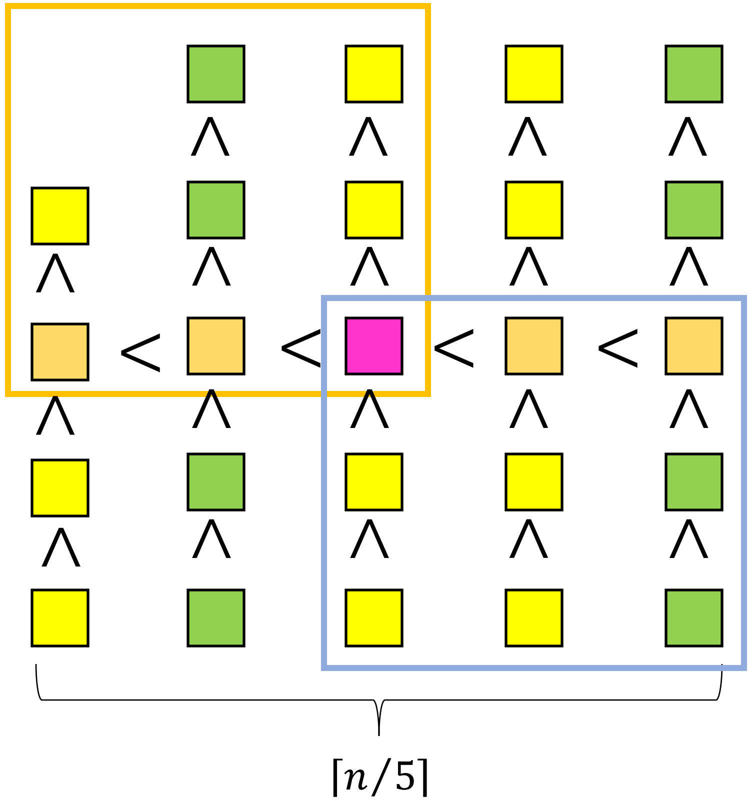
Elements smaller than MedianofMedians: $3\left(\left\lceil \frac{1}{2}\cdot\left\lceil\frac{n}{5}\right\rceil\right\rceil - 2\right) \geq \frac{3n}{10} - 6$ elements

Number of lists to the “left”

Exclude list on the endpoint, and “middle” list

Why is this good?

MedianofMedians is larger than all of these



Elements smaller than MedianofMedians: $3\left(\left\lceil \frac{1}{2} \cdot \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2\right) \geq \frac{3n}{10} - 6$ elements

Elements greater than MedianofMedians: $3\left(\left\lceil \frac{1}{2} \cdot \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2\right) \geq \frac{3n}{10} - 6$ elements

Back to: Quickselect

Divide: select an element p using Median of Medians, Partition(p)

$M(n) + \Theta(n)$

median of medians algorithm

partition algorithm

Quickselect

Divide: select an element p using Median of Medians, Partition(p)

$$M(n) + \Theta(n)$$

Conquer: if $i =$ index of p, done, if $i <$ index of p recurse left. Else recurse right (with index $i - p$)

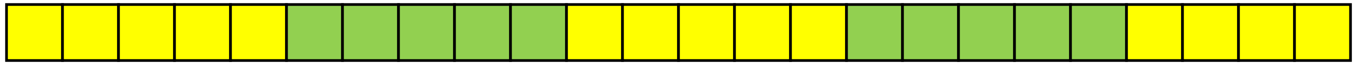
$$\leq S\left(\frac{7n}{10}\right)$$

Combine: Nothing!

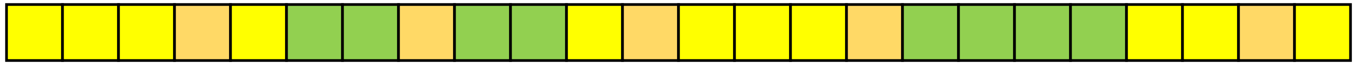
$$S(n) \leq S\left(\frac{7n}{10}\right) + M(n) + \Theta(n)$$

Median of Medians

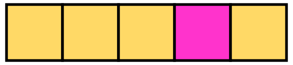
1. Break list into blocks of size 5 $\Theta(n)$



2. Find the median of each chunk $\Theta(n)$



3. Return median of medians (using Quickselect) $S\left(\frac{n}{5}\right)$



$$M(n) = S\left(\frac{n}{5}\right) + \Theta(n)$$

Quickselect

$$S(n) \leq S\left(\frac{7n}{10}\right) + M(n) + \Theta(n)$$

$$M(n) = S\left(\frac{n}{5}\right) + \Theta(n)$$

$$= S\left(\frac{7n}{10}\right) + S\left(\frac{n}{5}\right) + \Theta(n)$$

$$= S\left(\frac{7n}{10}\right) + S\left(\frac{2n}{10}\right) + \Theta(n)$$

$$\leq S\left(\frac{9n}{10}\right) + \Theta(n)$$

Because $S(n) = \Omega(n)$

CLRS gives a more rigorous proof!
See p. 203 for more details

Master theorem Case 3!

$$S(n) = O(n)$$

$$S(n) = \Theta(n)$$

Phew! Back to Quicksort

Divide: Select a pivot element, and partition about the pivot

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

Using Quickselect, always pivot about the median

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

Conquer: Recursively sort left and right sublists

If pivot is the median, list is split in half each iteration

Phew! Back to Quicksort

Divide: Select a pivot element, and partition about the pivot

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

Using Quickselect, always pivot about the median

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

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

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

A Worthwhile Choice?

Using Quickselect to pick median guarantees $\Theta(n \log n)$ worst-case run-time

Approach has very large constants

- If you really want $\Theta(n \log n)$, better off using MergeSort

More efficient approach: Random pivot

- Very small constant (very fast algorithm)
- Expected to run in $\Theta(n \log n)$ time
  - Why? Unbalanced partitions are very unlikely

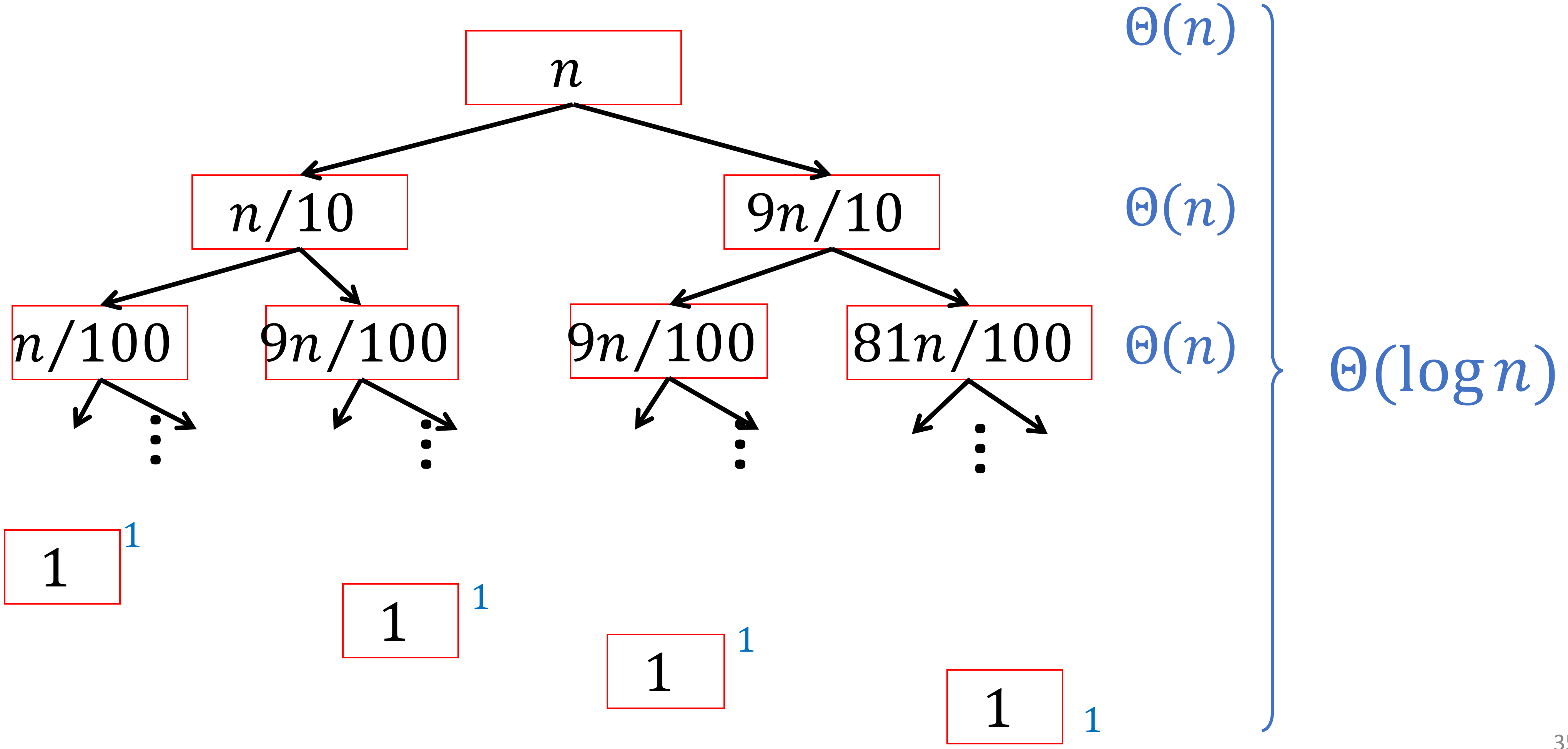
Quicksort Running Time

If the pivot is always $(n/10)^{\text{th}}$ order statistic:

$$T(n) = T(n/10) + T(9n/10) + \Theta(n)$$

Quicksort Running Time

$$T(n) = T(n/10) + T(9n/10) + \Theta(n)$$



Quicksort Running Time

If the pivot is always $(n/10)^{\text{th}}$ order statistic:

$$T(n) = T(n/10) + T(9n/10) + \Theta(n)$$
$$= \Theta(n \log n)$$

This is true if the pivot is any $(n/k)^{\text{th}}$ order statistic for any constant $k > 1$ (as long as the size of the smaller list is a <u>constant fraction</u> of the full list, we get $\Theta(n \log n)$ running time)

Quicksort Running Time

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

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

Then we shorten by d each time

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

$$= \Theta(n^2)$$

What's the probability of this occurring (for a random pivot)?

Probability of Always Choosing d^{th} Order Statistic

We must consistently select pivot from within the first d terms

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

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

Probability all pivots are among d smallest:

Very small probability!

$$\frac{d}{n} \times \frac{d}{n-d} \times \frac{d}{n-2d} \times \cdots \times \frac{d}{2d} \times 1 = \left(\frac{n}{d} \times \left(\frac{n}{d} - 1\right) \times \cdots \times 1\right)^{-1} = \frac{1}{\left(\frac{n}{d}\right)!}$$

Maximum Sum Continuous Subarray

The maximum-sum subarray of a given array of integers A is the interval $[a, b]$ such that the sum of all values in the array between a and b inclusive is maximal.

Given an array of n integers (may include both positive and negative values), give a $O(n \log n)$ algorithm for finding the maximum-sum subarray.

Divide and Conquer $\Theta(n \log n)$

5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Recursively Solve on Left

Divide in half

Recursively Solve on Right

Divide and Conquer $\Theta(n \log n)$

Largest sum that ends here + Largest sum that starts here

6	1	-7	-3	-6	-13	2	8	-12	5	13	-37	-42	-20
5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Recursively Solve on Left
19

Divide in half

Find Largest sum that spans the cut

Recursively Solve on Right
25

Divide and Conquer $\Theta(n \log n)$

Return the Max of

Left, Right, Center

6	1	-7	-3	-6	-13	2	8	-12	5	13	-37	-42	-20
5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Recursively Solve on Left
19

Divide in half

Find Largest sum that spans the cut
19

Recursively Solve on Right
25

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

Divide and Conquer Summary

Typically multiple subproblems.
Typically all roughly the same size.

Divide

- Break the list in half

Conquer

- Find the best subarrays on the left and right

Combine

- Find the best subarray that "spans the divide"
- I.e. the best subarray that ends at the divide concatenated with the best that starts at the divide

Generic Divide and Conquer Solution

```
def myDCalgo(problem):
        if baseCase(problem):
                solution = solve(problem) #brute force if necessary
                return solution
        subproblems = Divide(problem)
        for sub in subproblems:
                subsolutions.append(myDCalgo(sub))
        solution = Combine(subsolutions)
        return solution
```

MSCS Divide and Conquer $\Theta(n \log n)$

```
def MSCS(list):
    if list.length < 2:
        return list[0] #list of size 1 the sum is maximal
    {listL, listR} = Divide (list)
    for list in {listL, listR}:
        subSolutions.append(MSCS(list))
    solution = max(solnL, solnR, span(listL, listR))
    return solution
```

Types of “Divide and Conquer”

Divide and Conquer

- Break the problem up into several subproblems of roughly equal size, recursively solve
- E.g. Karatsuba, Closest Pair of Points, Mergesort…

Decrease and Conquer

- Break the problem into a single smaller subproblem, recursively solve
- E.g. Quickselect, Binary Search

Pattern So Far

Typically looking to divide the problem by some fraction (½, ¼ the size)

Not necessarily always the best!

- Sometimes, we can write faster algorithms by finding unbalanced divides.

Chip and Conquer

Divide

- Make a subproblem of all but the last element

Conquer

- Find best subarray on the left ($BSL(n-1)$)
- Find the best subarray ending at the divide ($BED(n-1)$)

Combine

- New Best Ending at the Divide:
  - $BED(n) = \max(BED(n-1) + arr[n],\ 0)$
- New best on the left:
  - $BSL(n) = \max\left(BSL(n-1),\ BED(n)\right)$

5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Divide

Recursively Solve on Left
25

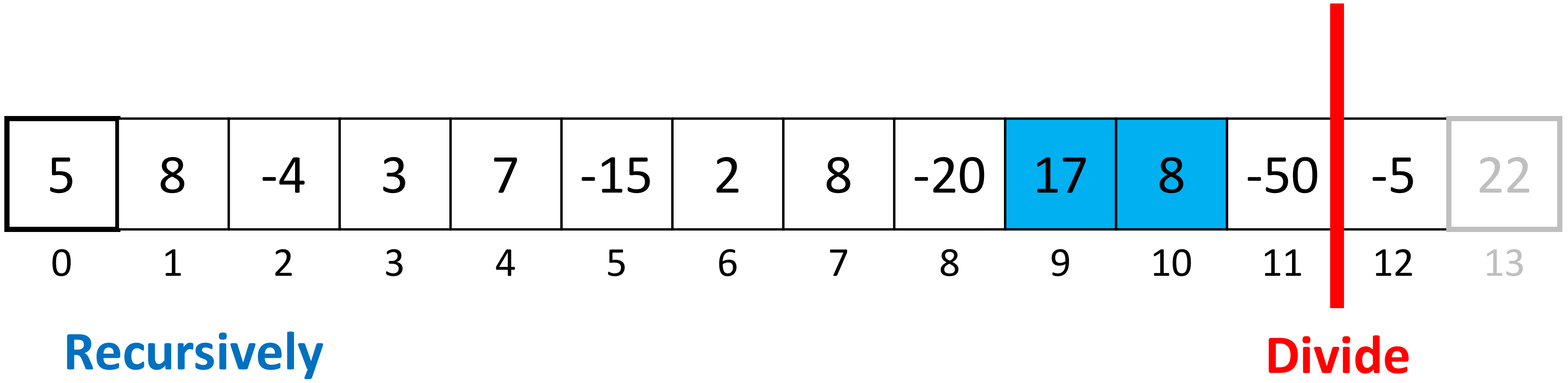
Find Largest sum ending at the cut
22

5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Divide

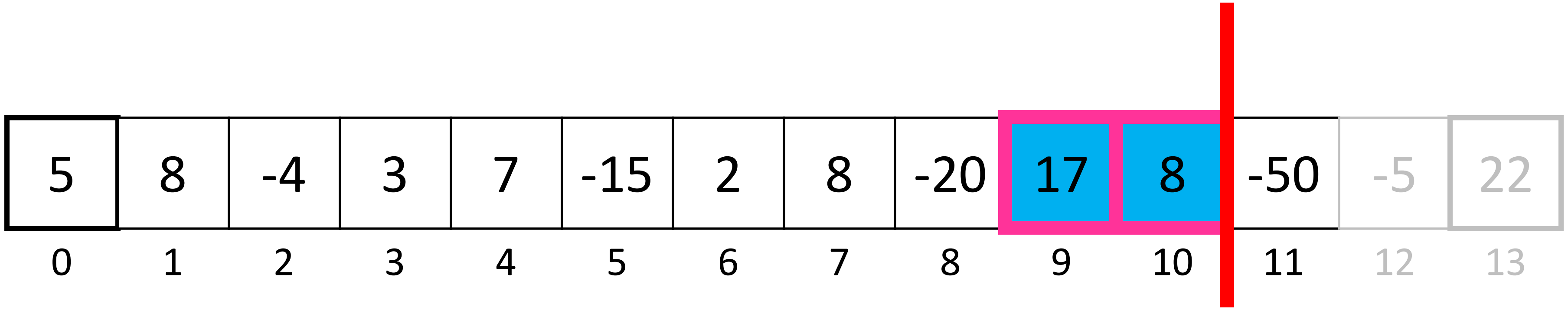
Recursively Solve on Left
25

Find Largest sum ending at the cut
0



Recursively Solve on Left
25

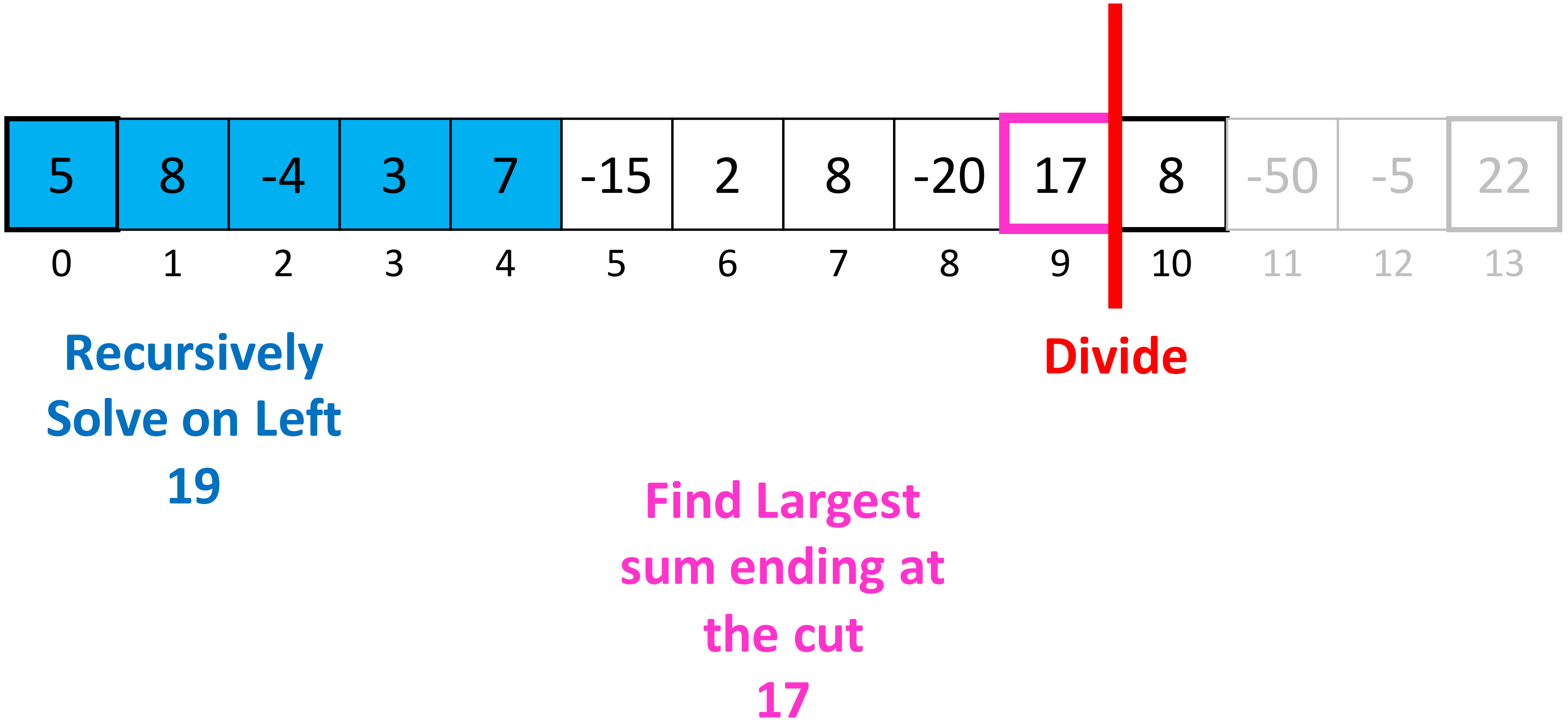
Find Largest sum ending at the cut
0

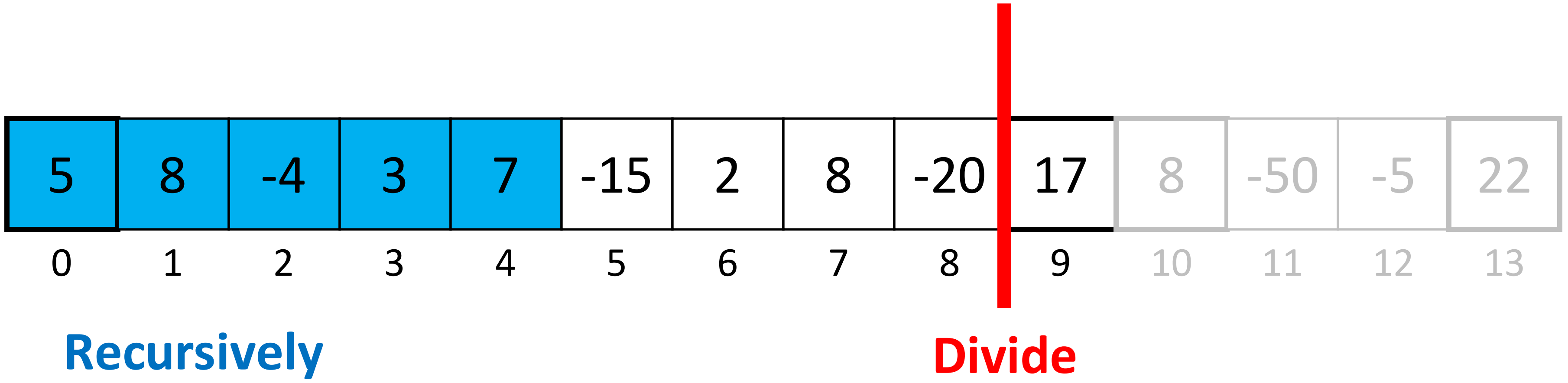


Recursively Solve on Left
25

Divide

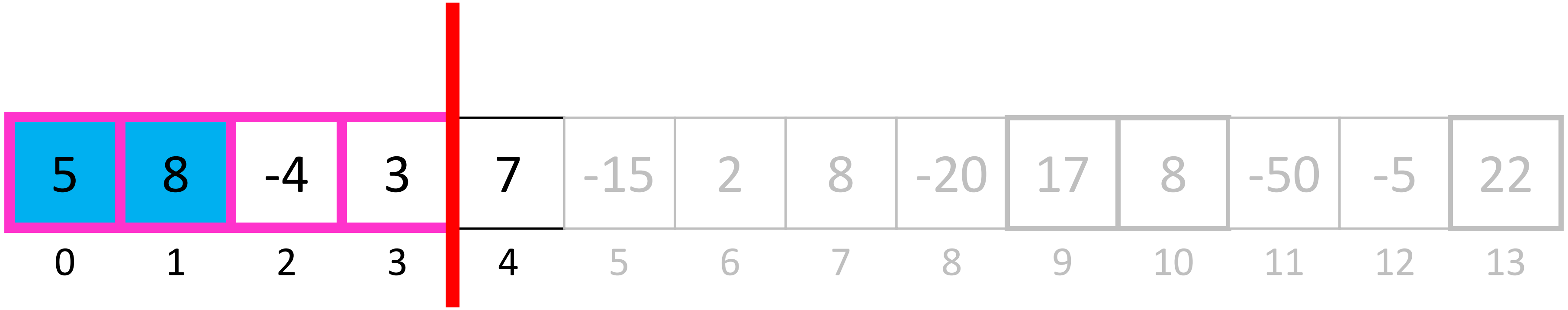
Find Largest sum ending at the cut
25


5
8
-4
3
7
-15
2
8
-20
17
8
-50
-5
22
0
1
2
3
4
5
6
7
8
9
10
11
12
13
Recursively Solve on Left
19
Divide
Find Largest sum ending at the cut
17



Recursively Solve on Left
19

Find Largest sum ending at the cut
0



Recursively Solve on Left
13

Divide

Find Largest sum ending at the cut
12

Chip and Conquer

Divide

- Make a subproblem of all but the last element

Conquer

- Find best subarray on the left ($BSL(n-1)$)
- Find the best subarray ending at the divide ($BED(n-1)$)

Combine

- New Best Ending at the Divide:
  - $BED(n) = \max(BED(n-1) + arr[n],\ 0)$
- New best on the left:
  - $BSL(n) = \max\left(BSL(n-1),\ BED(n)\right)$

Was unbalanced better? YES

Old:

- We divided in Half
- We solved 2 different problems:
  - Find the best overall on BOTH the left/right
  - Find the best which end/start on BOTH the left/right respectively
- Linear time combine

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

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

New:

- We divide by 1, n-1
- We solve 2 different problems:
  - Find the best overall on the left ONLY
  - Find the best which ends on the left ONLY
- Constant time combine

$$T(n) = 1T(n-1) + 1$$

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

MSCS Problem - Redux

Solve in $O(n)$ by increasing the problem size by 1 each time.

Idea: Only include negative values if the positives on both sides of it are "worth it"

$\Theta(n)$ Solution

5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Begin here

Remember two values:

Best So Far
5

Best ending here
5

$\Theta(n)$ Solution

5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Remember two values:

Best So Far
13

Best ending here
13

$\Theta(n)$ Solution

5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Remember two values:

Best So Far
13

Best ending here
9

$\Theta(n)$ Solution

5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Remember two values:

Best So Far
13

Best ending here
12

$\Theta(n)$ Solution

5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Remember two values:

Best So Far
19

Best ending here
19

$\Theta(n)$ Solution

5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Remember two values:

Best So Far
19

Best ending here
4

$\Theta(n)$ Solution

5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Remember two values:

Best So Far
19

Best ending here
14

$\Theta(n)$ Solution

5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Remember two values:

Best So Far
19

Best ending here
0

$\Theta(n)$ Solution

5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Remember two values:

Best So Far
19

Best ending here
17

$\Theta(n)$ Solution

5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Remember two values:

Best So Far
25

Best ending here
25