CSE 250
Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Divide and Conquer
Textbook Ch 15

Announcements

- WA1 is Due Wednesday @ 11:59pm

Recap

- **Recursion:** A big problem made up of one or more instances of a smaller problem
  - Factorial: f(n) = n * f(n-1)
  - Fibonacci: f(n) = f(n-1) + f(n-2)
  - Towers of Hanoi: move(n) = move(n-1), move(1), then move(n-1) again
- **Inductive Proofs:**
  - Come up with a hypothesis
  - Prove it on the base case
  - Assume it works for n' < n; Prove for n based on that assumption

Inductive Proof for Towers of Hanoi

- Base case is one ring. I can move one ring.
- Assume I can move n-1 rings; Can I prove that I can move n? Yes
  - Move n - 1 (which we can do based on our assumption)
  - Move 1 ring
  - Move n - 1 (which we can do based on our assumption.
  - Therefore, if we can move n - 1, we can move n.

** Note this is just a proof that we can solve it for any value of n. The actual number of steps required can also be shown by induction and will be covered in recitation*

Fibonacci

What is the complexity of `fib(n)`?

```
def fib(n: Int): Long =
    if(n < 2){ 1 }
    else { fibb(n-1) + fibb(n-2) }
```

Fibonacci

$$T(n) = \begin{cases} \Theta(1) & \textbf{if } n < 2 \\ T(n-1) + T(n-2) + \Theta(1) & \textbf{otherwise} \end{cases}$$

Solve for $T(n)$...How?

Divide and Conquer

Remember the Towers of Hanoi…

Divide and Conquer

Remember the Towers of Hanoi…

1. You can move *n* blocks if you know how to move *n*-1 blocks

Divide and Conquer

Remember the Towers of Hanoi…

1. You can move *n* blocks if you know how to move *n*-1 blocks
2. You can move *n*-1 blocks if you know how to move *n*-2 blocks

Divide and Conquer

Remember the Towers of Hanoi…

1. You can move *n* blocks if you know how to move *n*-1 blocks
2. You can move *n*-1 blocks if you know how to move *n*-2 blocks
3. You can move *n*-2 blocks if you know how to move *n*-3 blocks

Divide and Conquer

Remember the Towers of Hanoi…

1. You can move *n* blocks if you know how to move *n*-1 blocks
2. You can move *n*-1 blocks if you know how to move *n*-2 blocks
3. You can move *n*-2 blocks if you know how to move *n*-3 blocks
4. You can move *n*-3 blocks if you know how to move *n*-4 blocks

Divide and Conquer

Remember the Towers of Hanoi…

1. You can move *n* blocks if you know how to move *n*-1 blocks
2. You can move *n*-1 blocks if you know how to move *n*-2 blocks
3. You can move *n*-2 blocks if you know how to move *n*-3 blocks
4. You can move *n*-3 blocks if you know how to move *n*-4 blocks

…

You can always move 1 block

Divide and Conquer

To solve the problem at n:

Divide and Conquer

To solve the problem at *n*:

Divide the problem into smaller problems (size *n*-1 and 1 in this case)

Divide and Conquer

To solve the problem at *n*:

Divide the problem into smaller problems (size *n*-1 and 1 in this case)

Conquer the smaller problems

Divide and Conquer

To solve the problem at *n*:

Divide the problem into smaller problems (size *n*-1 and 1 in this case)

Conquer the smaller problems

Combine the smaller solutions to get the bigger solution

Merge Sort

Input: An array with elements in an unknown order.

Output: An array with elements in sorted order.

Merge Sort - Questions

Divide (break the array into smaller arrays)

What's the smallest list I could try to sort?

Merge Sort - Questions

Divide (break the array into smaller arrays)

What's the smallest list I could try to sort? size n = 1

Merge Sort - Questions

Divide (break the array into smaller arrays)
What's the smallest list I could try to sort? size n = 1

Conquer (sort the smaller arrays)
How do I sort it?

Merge Sort - Questions

Divide (break the array into smaller arrays)
What's the smallest list I could try to sort? size n = 1

Conquer (sort the smaller arrays)
How do I sort it? It's already sorted!!!

Merge Sort - Questions

Divide (break the array into smaller arrays)
What's the smallest list I could try to sort? size n = 1

Conquer (sort the smaller arrays)
How do I sort it? It's already sorted!!!

Combine (combine the sorted arrays into a bigger sorted array)
How can I do this, and how long does it take?

Merge Sort - Questions

Divide (break the array into smaller arrays)
What's the smallest list I could try to sort? size n = 1

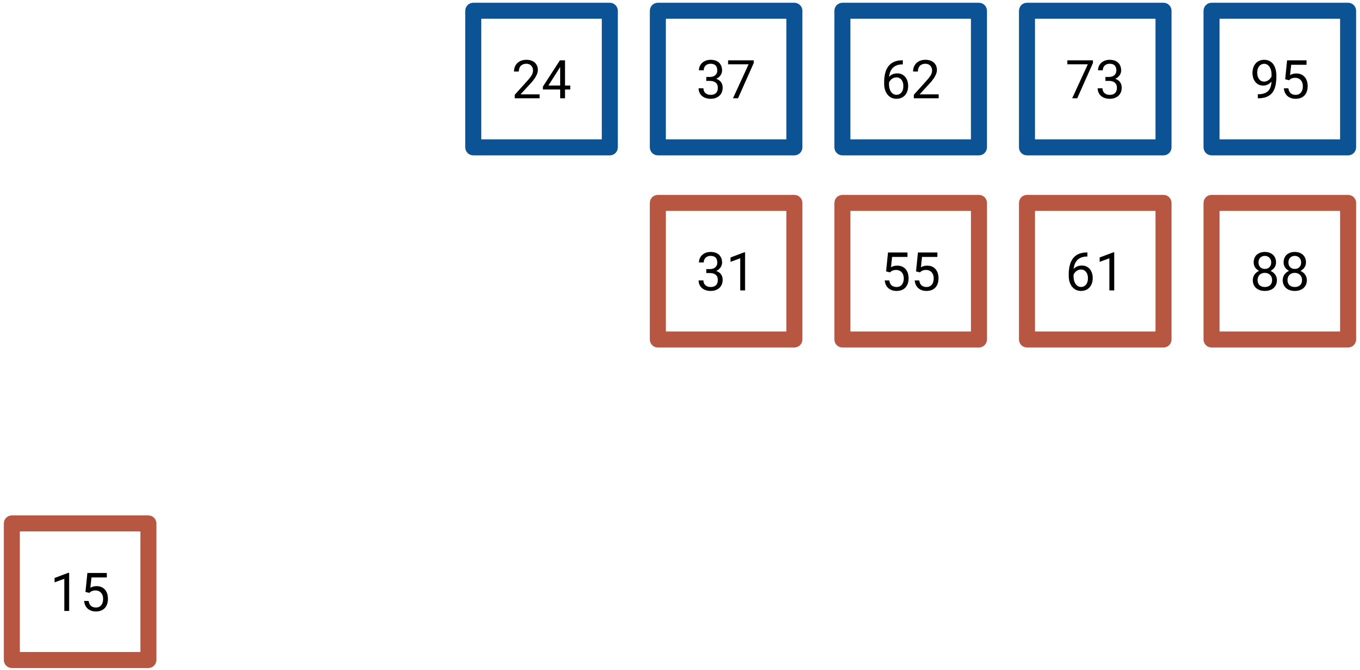
Conquer (sort the smaller arrays)
How do I sort it? It's already sorted!!!

Combine (combine the sorted arrays into a bigger sorted array)
How can I do this, and how long does it take? Merge…

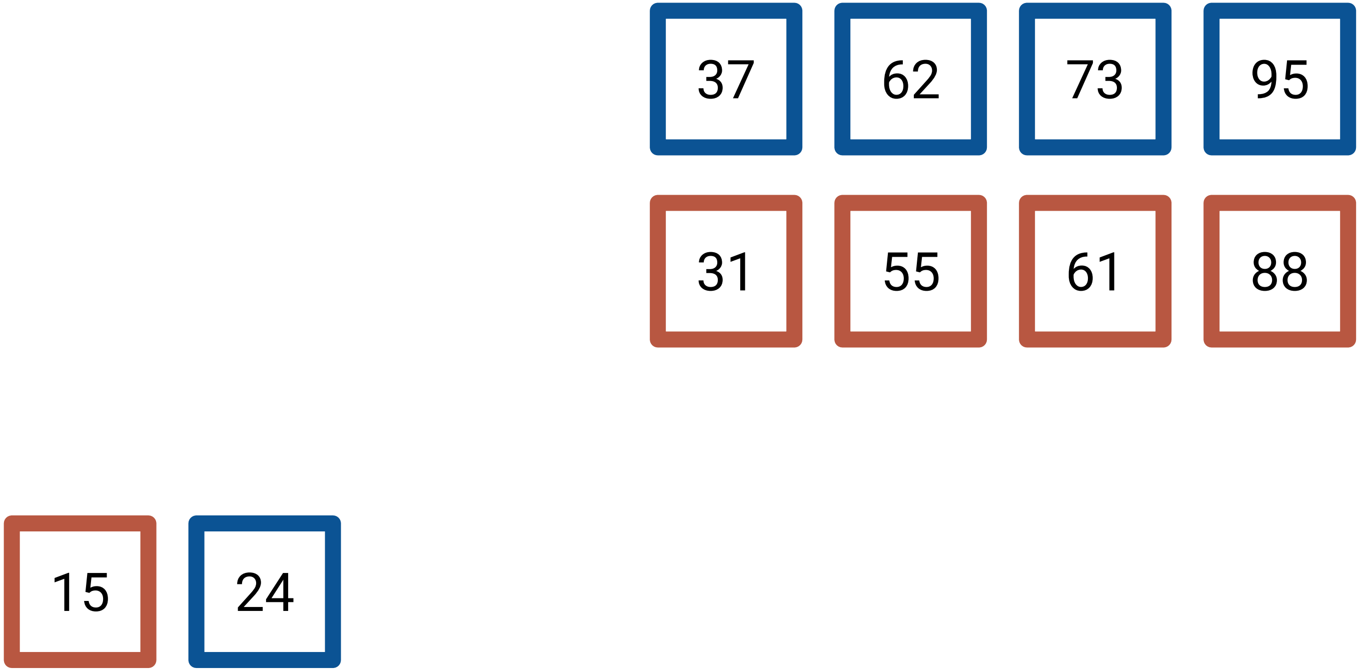
How do we Merge Two Sorted Arrays?

24	37	62	73	95
15	31	55	61	88

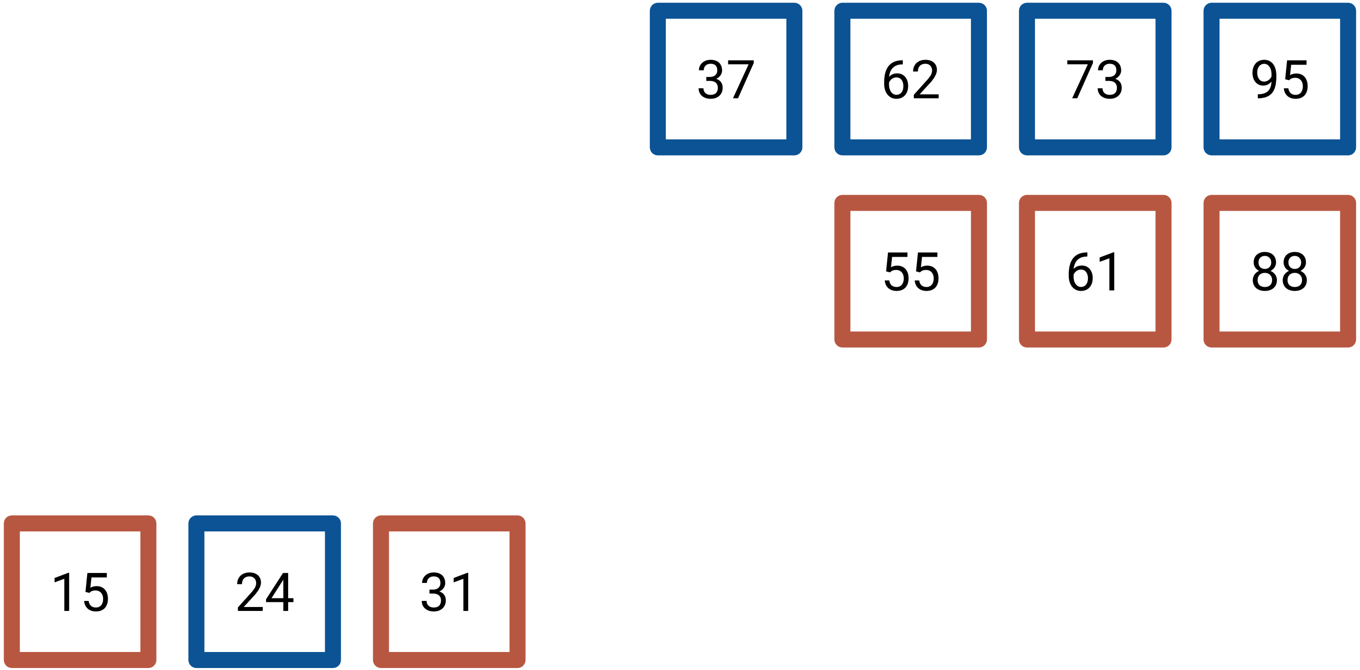
How do we Merge Two Sorted Arrays?



How do we Merge Two Sorted Arrays?



How do we Merge Two Sorted Arrays?



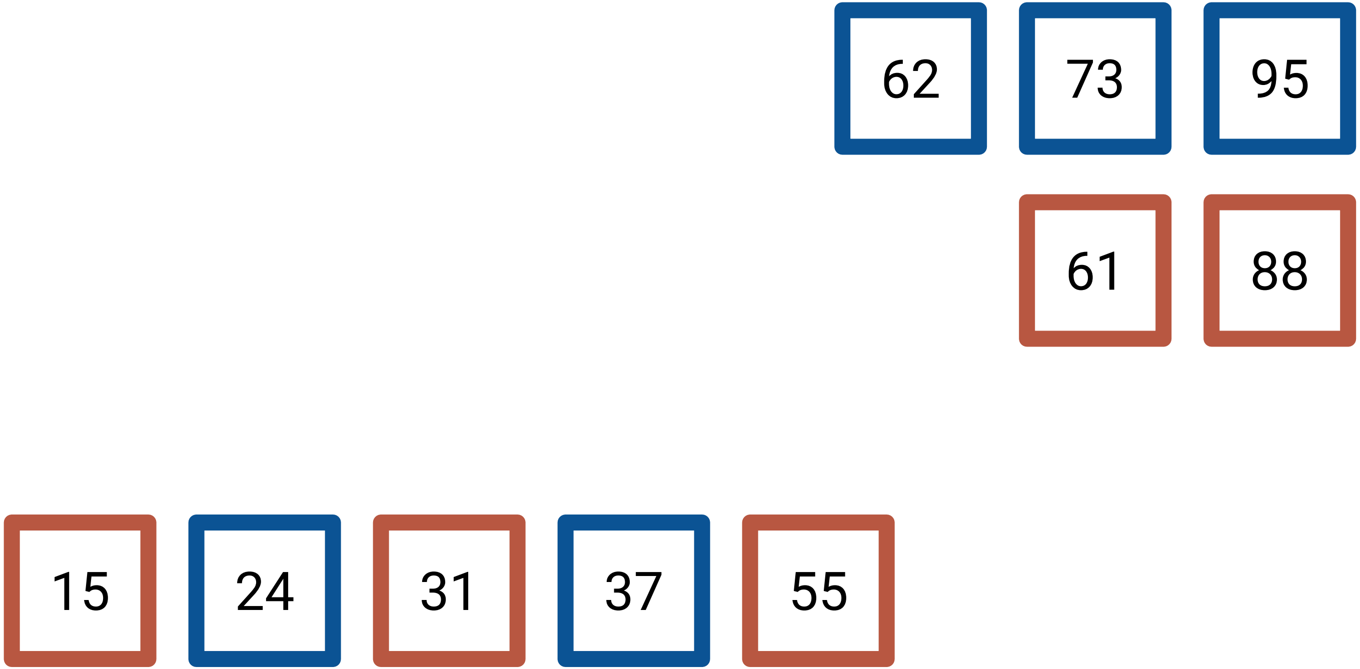
How do we Merge Two Sorted Arrays?

62 73 95

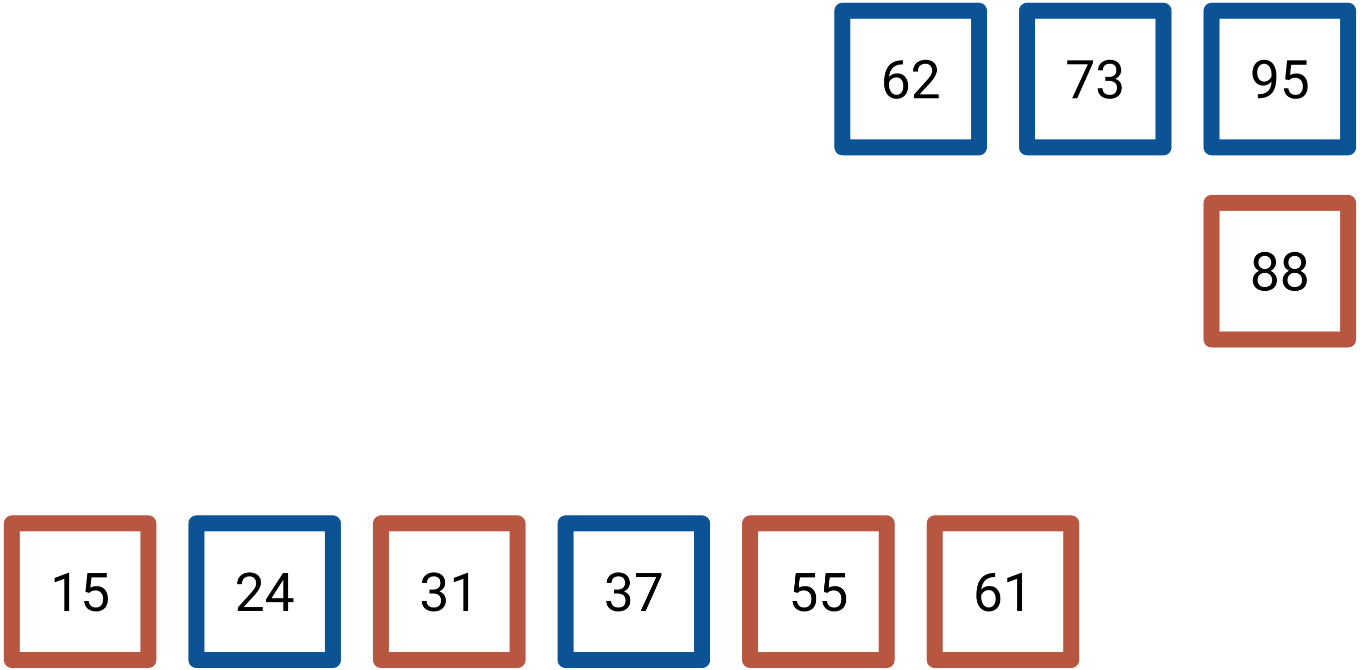
55 61 88

15 24 31 37

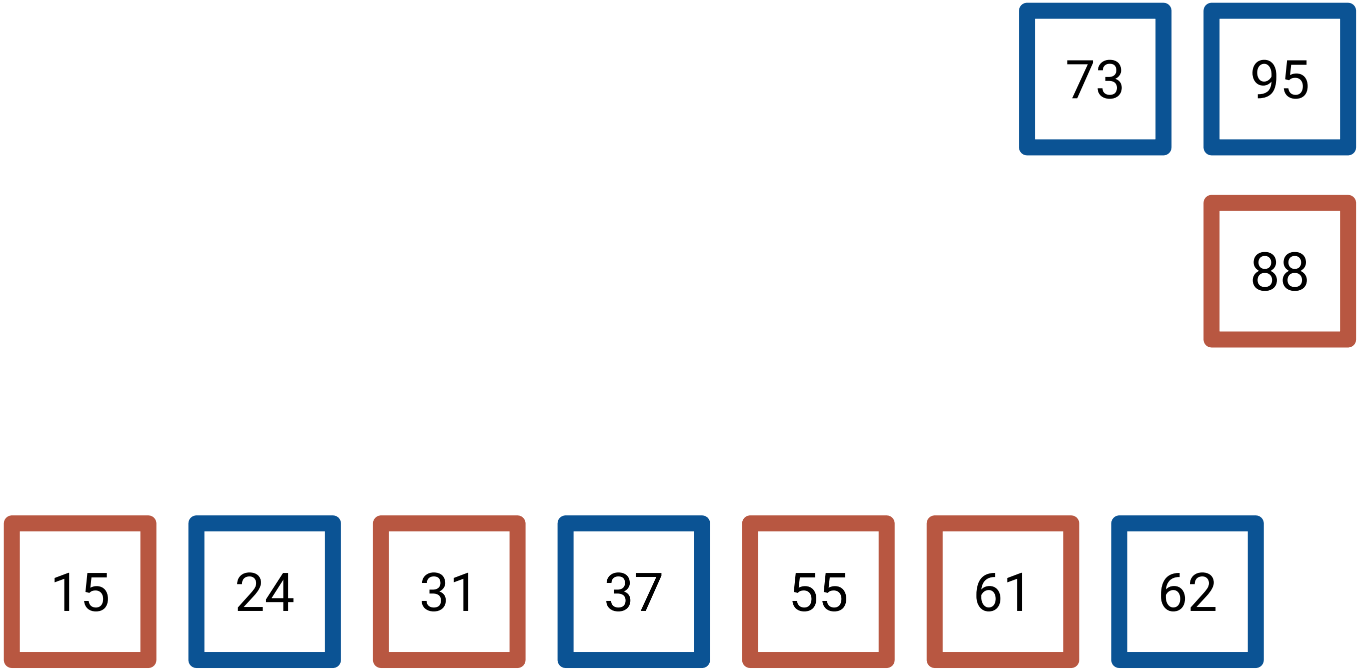
How do we Merge Two Sorted Arrays?



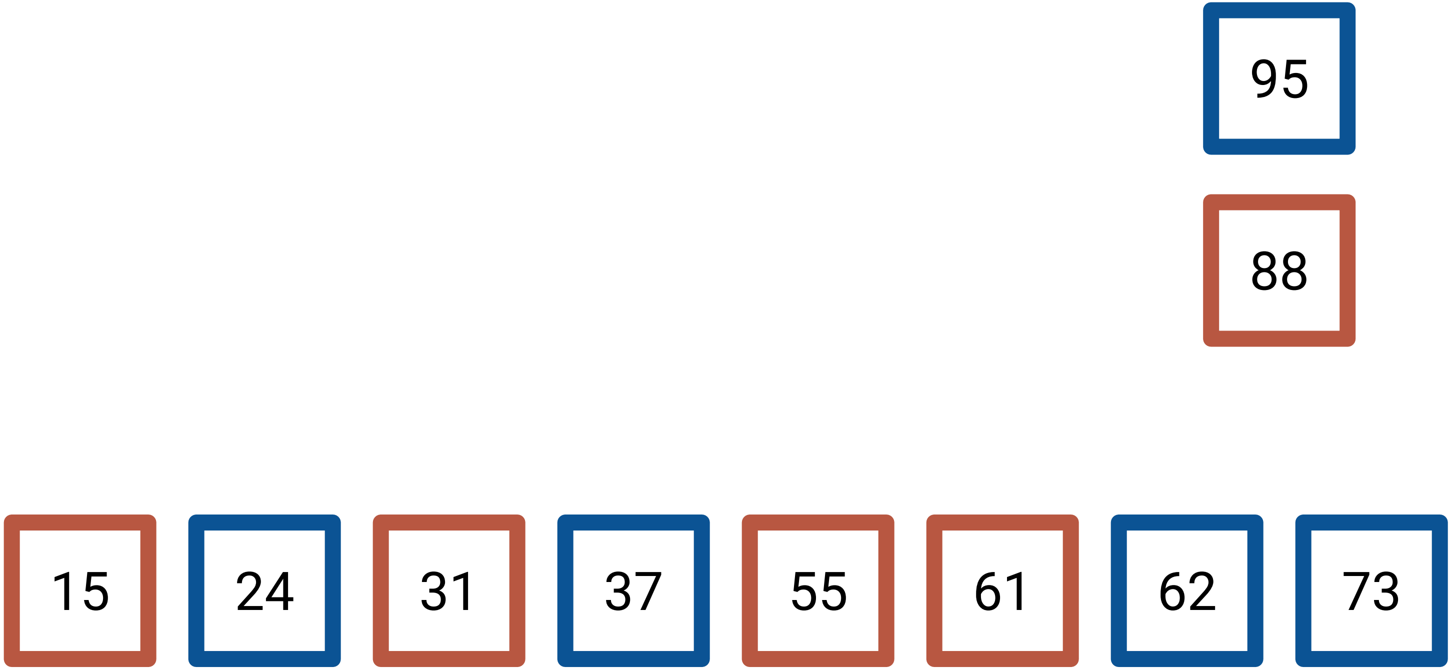
How do we Merge Two Sorted Arrays?



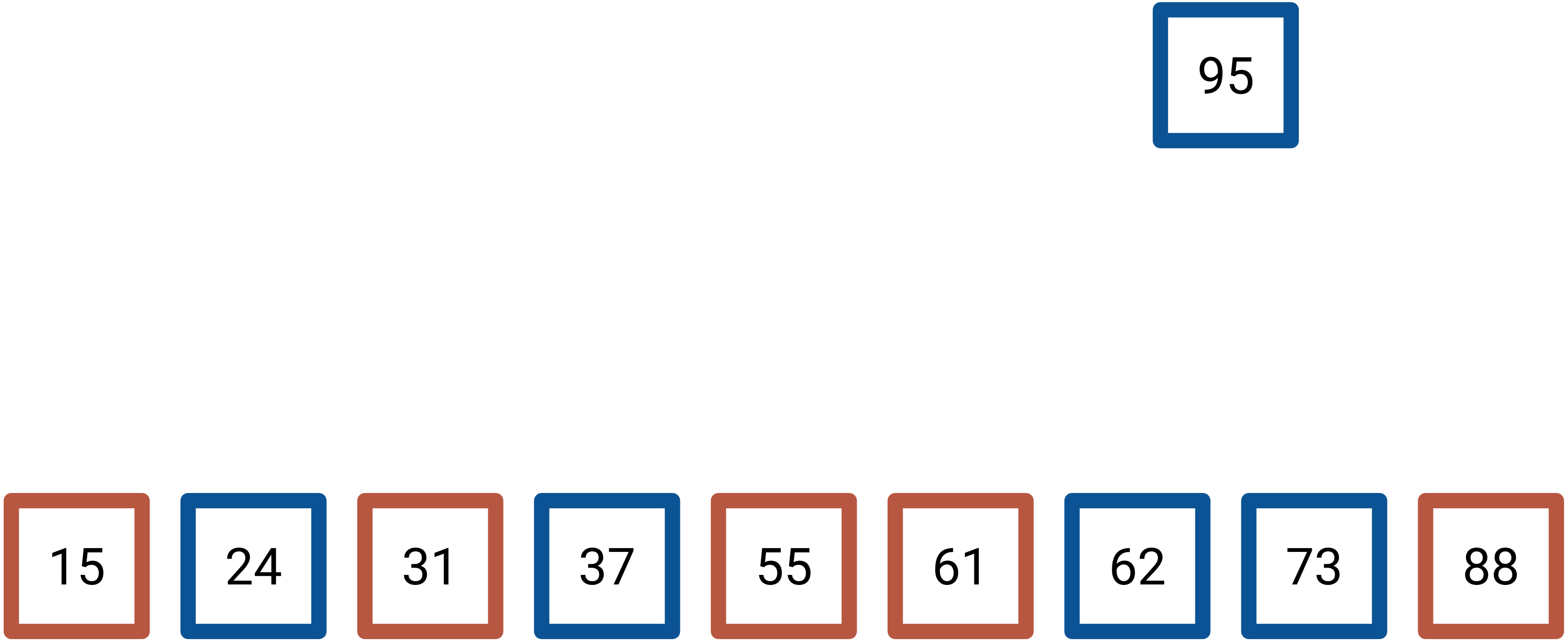
How do we Merge Two Sorted Arrays?



How do we Merge Two Sorted Arrays?



How do we Merge Two Sorted Arrays?



How do we Merge Two Sorted Arrays?

15	24	31	37	55	61	62	73	88	95

How do we Merge Two Sorted Arrays?

What was the complexity?

15	24	31	37	55	61	62	73	88	95

How do we Merge Two Sorted Arrays?

What was the complexity?

Each comparison was $\Theta(1)$...

15	24	31	37	55	61	62	73	88	95

How do we Merge Two Sorted Arrays?

What was the complexity?

Each comparison was $\Theta(1)$...

How many comparisons? $\Theta(|red| + |blue|)$

15	24	31	37	55	61	62	73	88	95

Merge Code

```
def merge[A: Ordering](left: Seq[A], right: Seq[A]): Seq[A] = {
    val output = ArrayBuffer[A]()

    val leftItems = left.iterator.buffered
    val rightItems = right.iterator.buffered

    while(leftItems.hasNext || rightItems.hasNext) {
      if(!left.hasNext)              { output.append(right.next) }
      else if(!right.hasNext)        { output.append(left.next) }
      else if(Ordering[A].lt( left.head, right.head ))
                                     { output.append(left.next) }
      else                           { output.append(right.next) }
    }
    output.toSeq
  }
```

Divide

- We know how to combine sorted arrays
- We know that in a base case of $n = 1$ how to sort
- How do we divide our problem to get there?

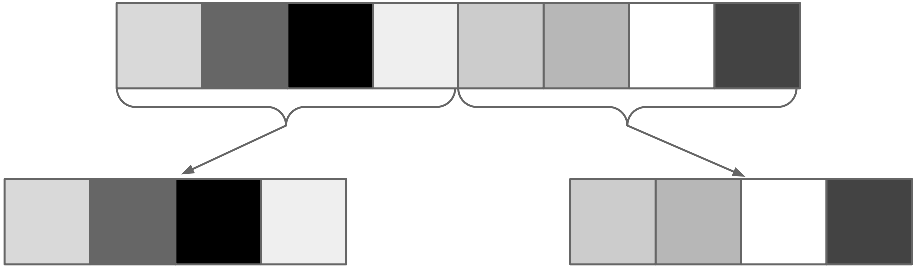
Divide

- We know how to combine sorted arrays
- We know that in a base case of n = 1 how to sort
- How do we divide our problem to get there?

Let's divide our array in half (recursively)!

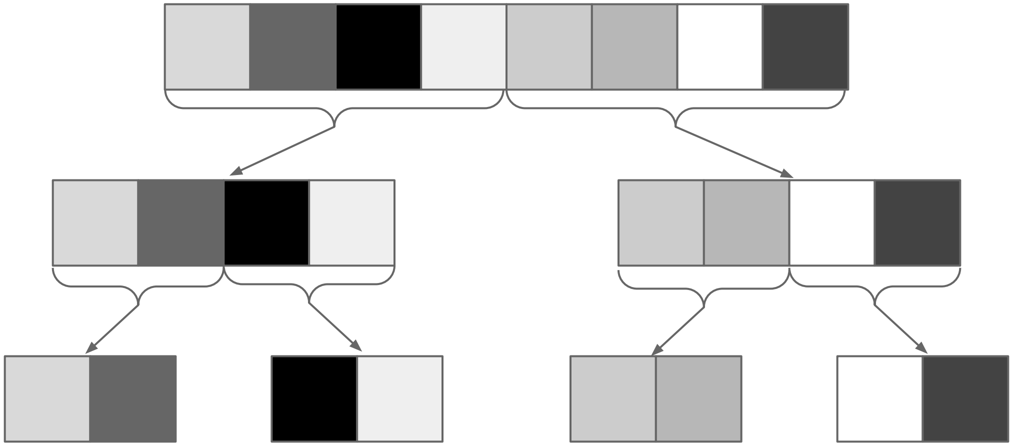
Visualization - Divide

Visualization - Divide



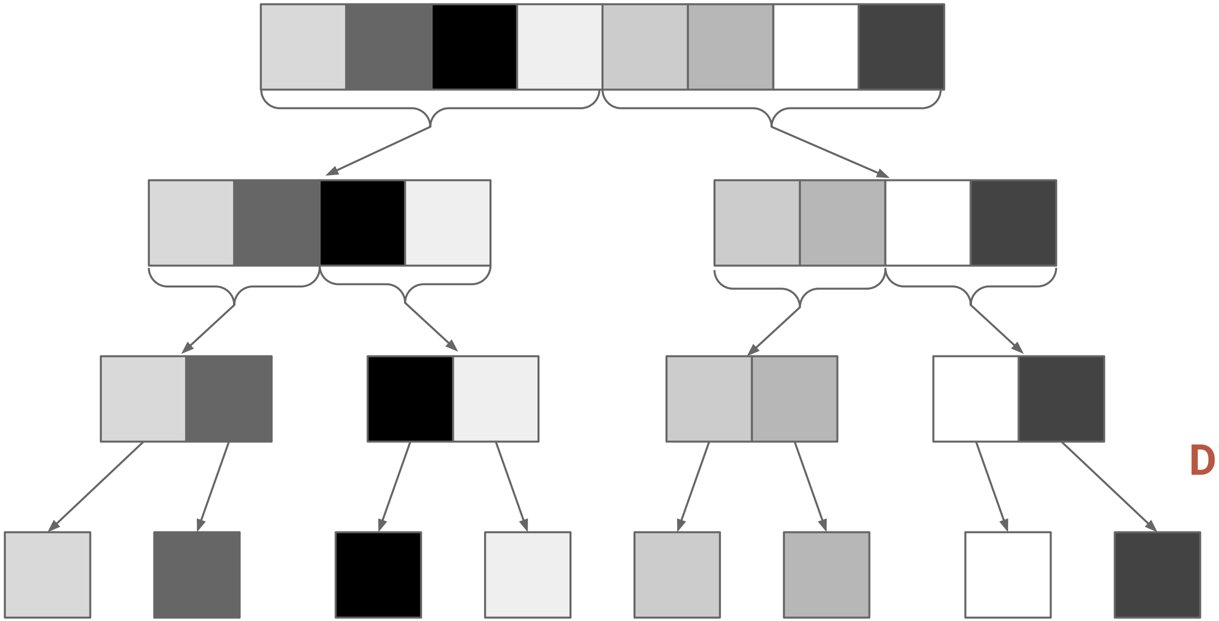
Divide the input in half

Visualization - Divide



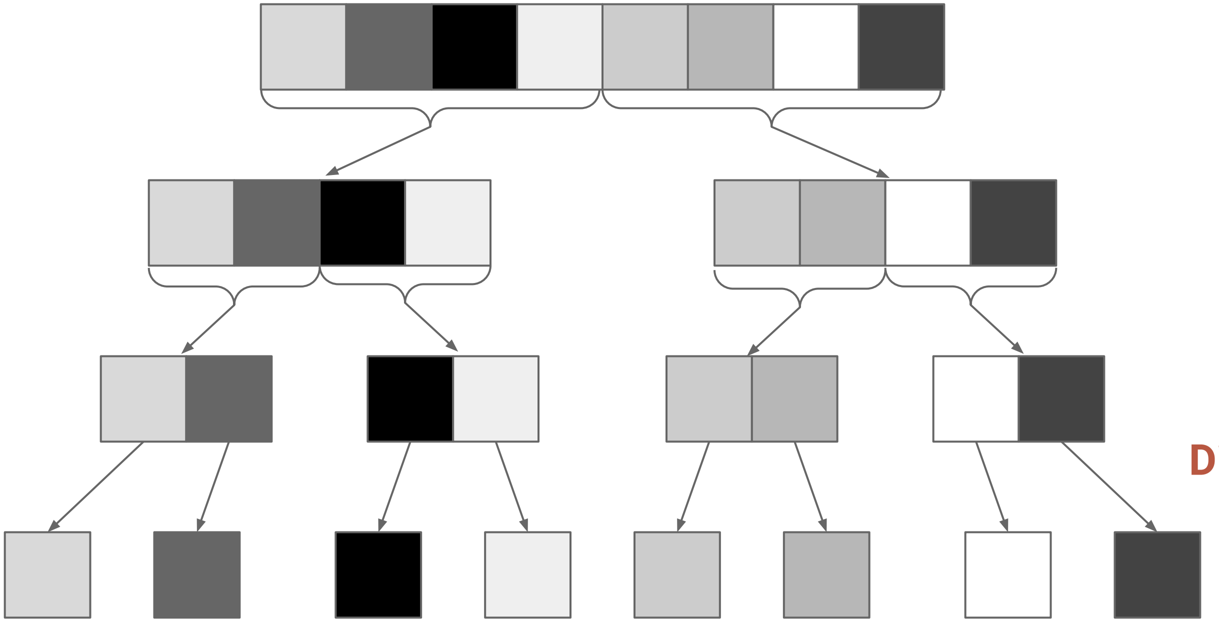
Divide each half in half

Visualization - Divide



Divide each half in half again...

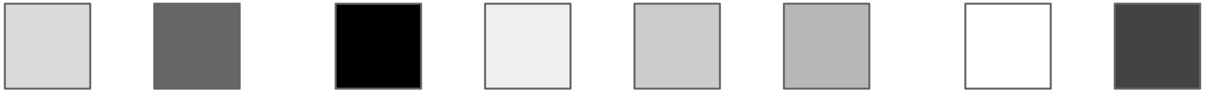
Visualization - Conquer



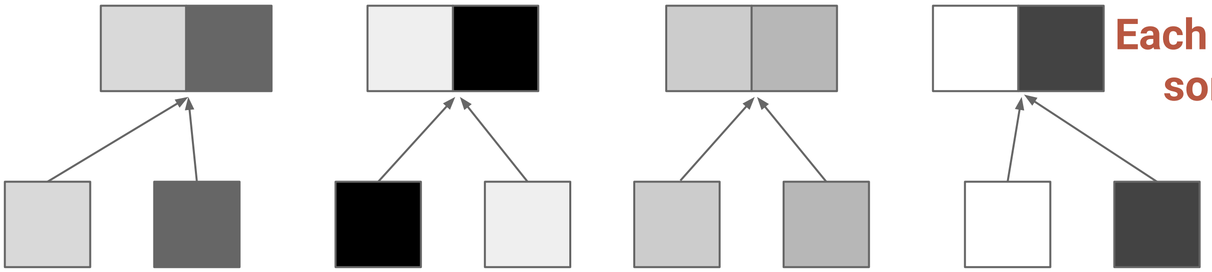
Divide each half in half again...

We can't divide in half anymore (base case)

Visualization - Combine

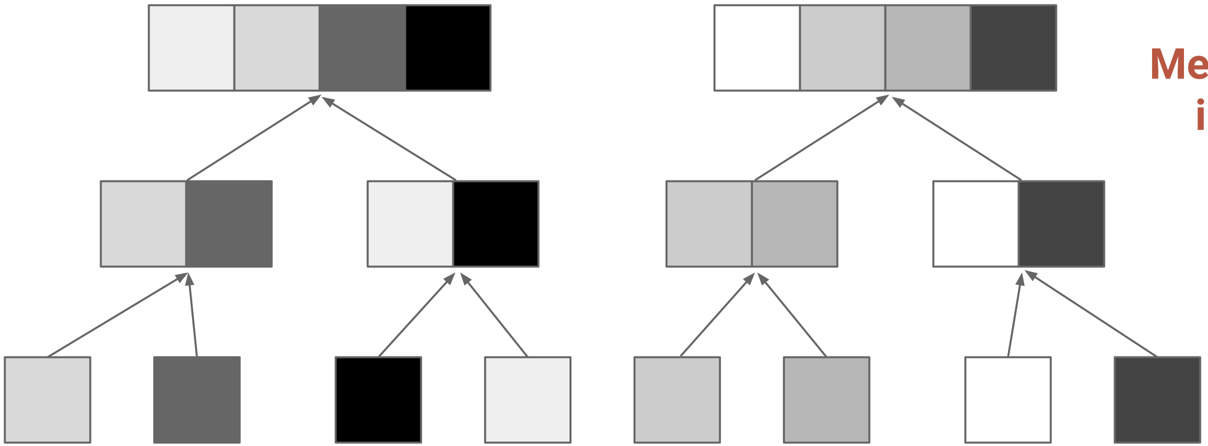


Visualization - Combine



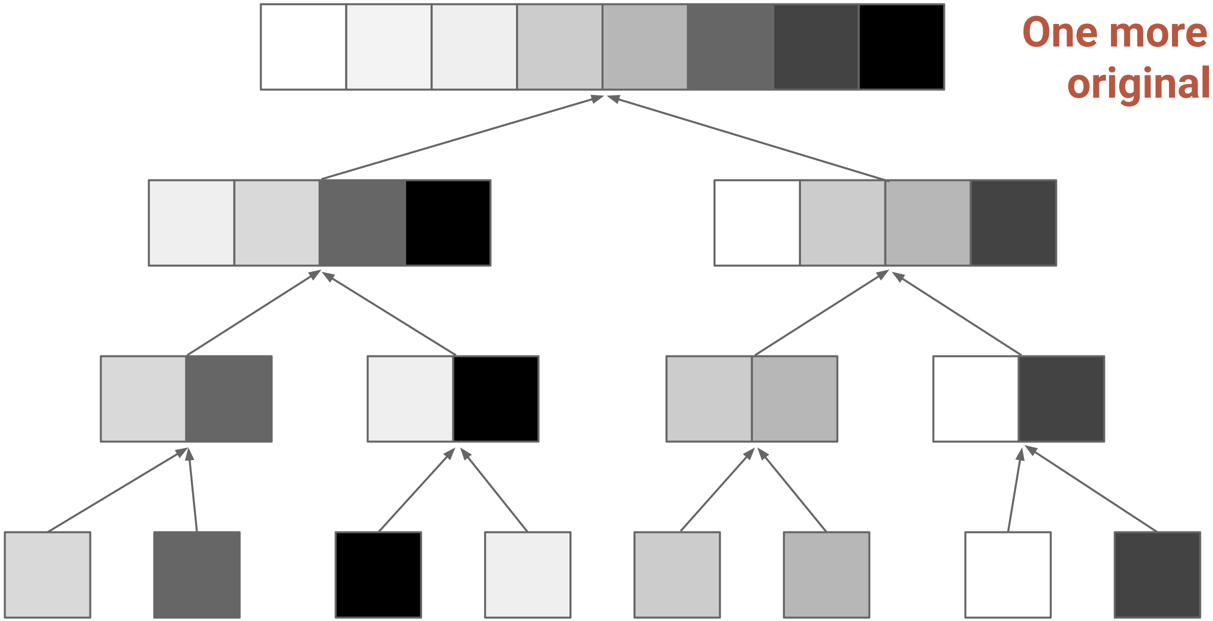
Each single item list is sorted...merge each pair into a bigger sorted list

Visualization - Combine



Merge each pair of 2 into sorted lists of size 4

Visualization - Combine



One more merge gets our original list fully sorted

Sort Code

```scala
def sort[A: Ordering](data: Seq[A]): Seq[A] =
    {
      if(data.length <= 1) { return data }
      else {
        val (left, right) = data.splitAt(data.length / 2)
        return merge(
          sort(left),
          sort(right)
        )
      }
    }
```

Complexity

If we solve a problem of size n by:

- Dividing it into a sub-problems
  - Where each problem is of size n/b (usually $b = a$)
  - …and stop recurring at $n \leq c$
  - …and the cost of dividing is $D(n)$
  - …and the cost of combining is $C(n)$

Then our total cost will be…

Complexity

$$T(n) = \begin{cases} \Theta(1) & \textbf{if } n \le c \\ a \cdot T(\frac{n}{b}) + D(n) + C(n) & \textbf{otherwise} \end{cases}$$

a subproblems of size *n*/*b*, base case of $n \le c$

divide cost of $D(n)$

and combine cost of $C(n)$

For Merge Sort

Divide: Split the sequence in half

$D(n) = \Theta(n)$ (can we do it faster?)

Conquer: Sort left and right halves

$a = 2, b = 2, c = 1$

Combine: Merge halves together

$C(n) = \Theta(n)$

For Merge Sort

$$T(n) = \begin{cases} \Theta(1) & \textbf{if } n \leq 1 \\ 2 \cdot T(\frac{n}{2}) + \Theta(1) + \Theta(n) & \textbf{otherwise} \end{cases}$$

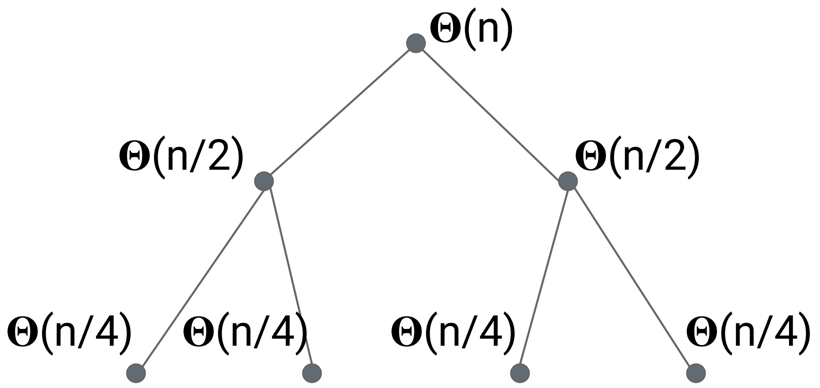
For Merge Sort

$$T(n) = \begin{cases} \Theta(1) & \textbf{if } n \leq 1 \\ 2 \cdot T(\frac{n}{2}) + \Theta(1) + \Theta(n) & \textbf{otherwise} \end{cases}$$

How do we find a closed-form hypothesis?

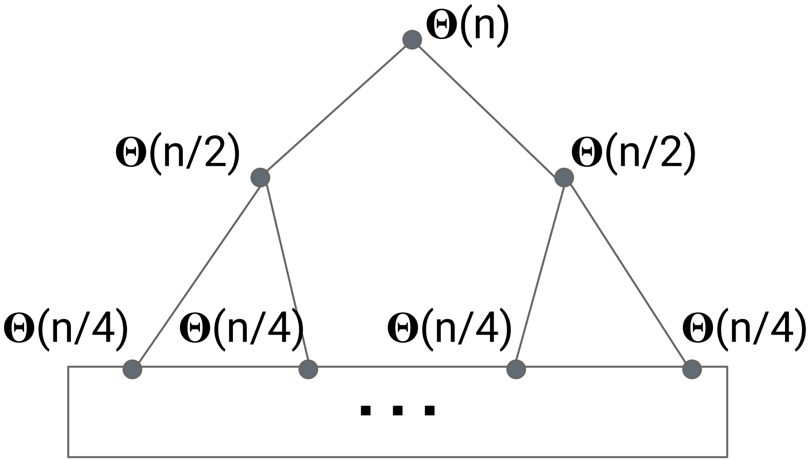
For Merge Sort: Recursion Trees

$$T(n) = \begin{cases} \Theta(1) & \textbf{if } n \leq 1 \\ 2 \cdot T(\frac{n}{2}) + \Theta(1) + \Theta(n) & \textbf{otherwise} \end{cases}$$



For Merge Sort: Recursion Trees

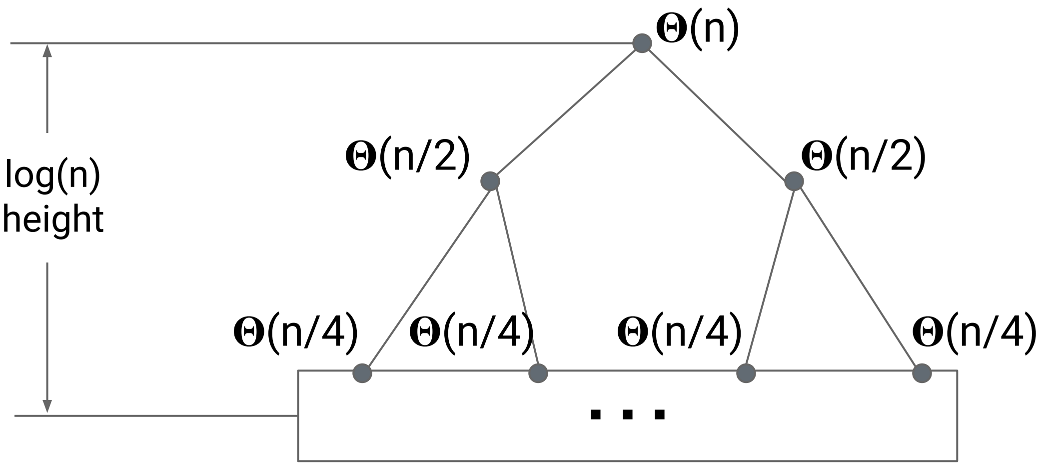
$$T(n) = \begin{cases} \Theta(1) & \textbf{if } n \leq 1 \\ 2 \cdot T(\frac{n}{2}) + \Theta(1) + \Theta(n) & \textbf{otherwise} \end{cases}$$



Each node shows $D(n) + C(n)$

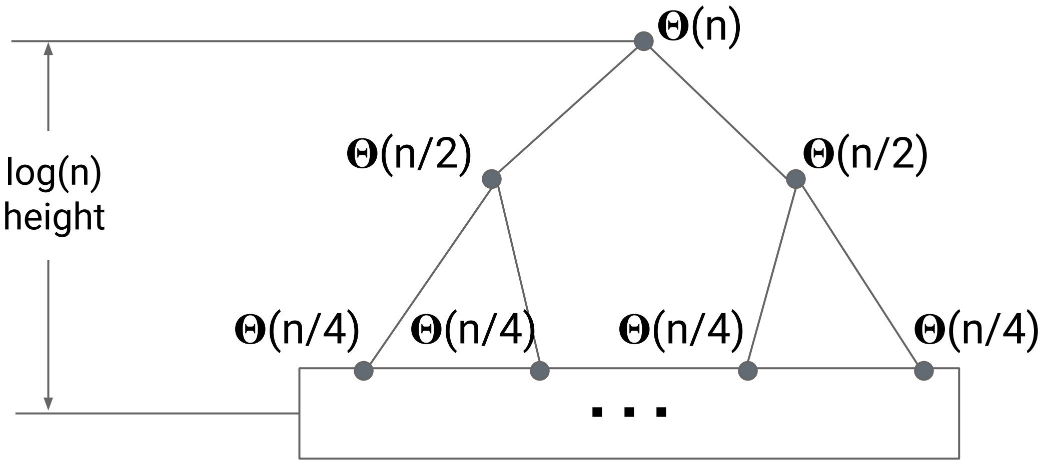
For Merge Sort: Recursion Trees

$$T(n) = \begin{cases} \Theta(1) & \textbf{if } n \le 1 \\ 2 \cdot T(\frac{n}{2}) + \Theta(1) + \Theta(n) & \textbf{otherwise} \end{cases}$$



Each node shows $D(n) + C(n)$

For Merge Sort: Recursion Trees



At level *i* there are 2^i tasks, each with runtime $\Theta(n/2^i)$, and there are log(*n*) levels.

For Merge Sort: Recursion Trees

At level *i* there are 2^i tasks, each with runtime $\Theta(n/2^i)$,
and there are log(*n*) levels.

$$\sum_{i=0}^{\log(n)} \sum_{j=1}^{2^i} \Theta(\frac{n}{2^i})$$

For Merge Sort: Recursion Trees

At level *i* there are 2^i tasks, each with runtime $\Theta(n/2^i)$,
and there are log(*n*) levels.

$$\boxed{\sum_{i=0}^{\log(n)}} \sum_{j=1}^{2^i} \Theta(\frac{n}{2^i})$$

For Merge Sort: Recursion Trees

At level i there are 2^i tasks, each with runtime $\Theta(n/2^i)$,
and there are $\log(n)$ levels.

$$\sum_{i=0}^{\log(n)} \boxed{\sum_{j=1}^{2^i}} \Theta\left(\frac{n}{2^i}\right)$$

For Merge Sort: Recursion Trees

At level *i* there are 2^i tasks, each with runtime $\Theta(n/2^i)$,
and there are $\log(n)$ levels.

$$\sum_{i=0}^{\log(n)} \sum_{j=1}^{2^i} \boxed{\Theta(\frac{n}{2^i})}$$

Merge Sort Runtime

$$\sum_{i=0}^{\log(n)} \sum_{j=1}^{2^i} \Theta(\frac{n}{2^i})$$

Merge Sort Runtime

$$\sum_{i=0}^{\log(n)} \sum_{j=1}^{2^i} \Theta(\frac{n}{2^i})$$

$$\sum_{i=0}^{\log(n)} (2^i + 1 - 1)\Theta(\frac{n}{2^i})$$

Merge Sort Runtime

$$\sum_{i=0}^{\log(n)} \sum_{j=1}^{2^i} \Theta(\frac{n}{2^i})$$

$$\sum_{i=0}^{\log(n)} (2^i + 1 - 1)\Theta(\frac{n}{2^i})$$

$$\sum_{i=0}^{\log(n)} 2^i\Theta(\frac{n}{2^i})$$

Merge Sort Runtime

$$\sum_{i=0}^{\log(n)} 2^i \Theta(\frac{n}{2^i})$$

Merge Sort Runtime

$$\sum_{i=0}^{\log(n)} 2^i \Theta(\frac{n}{2^i})$$

$$\sum_{i=0}^{\log(n)} \Theta(n)$$

Merge Sort Runtime

$$\sum_{i=0}^{\log(n)} 2^i \Theta(\frac{n}{2^i})$$

$$\sum_{i=0}^{\log(n)} \Theta(n)$$

$$(\log(n) - 0 + 1)\Theta(n)$$

Merge Sort Runtime

$$\sum_{i=0}^{\log(n)} 2^i \Theta(\frac{n}{2^i})$$

$$\sum_{i=0}^{\log(n)} \Theta(n)$$

$$(\log(n) - 0 + 1)\Theta(n)$$

$$\Theta(n\log(n)) + \Theta(n)$$

Merge Sort Runtime

$$\sum_{i=0}^{\log(n)} 2^i \Theta(\frac{n}{2^i})$$

$$\sum_{i=0}^{\log(n)} \Theta(n)$$

$$(\log(n) - 0 + 1)\Theta(n)$$

$$\Theta(n\log(n)) + \Theta(n)$$

$$\Theta(n\log(n))$$

Merge Sort Runtime: Inductive Proof

Now we can use induction to prove that there is a c, n_0 s.t. $T(n) \leq c\, n\log(n)$ for any $n > n_0$

$$T(n) = \begin{cases} c_0 & \textbf{if } n \leq 1 \\ 2 \cdot T(\frac{n}{2}) + c_1 + c_2 \cdot n & \textbf{otherwise} \end{cases}$$

Merge Sort Runtime: Inductive Proof

Base Case: $T(1) \le c$

$c_0 \le c$

True for any $c > c_0$

Merge Sort Runtime: Inductive Proof

Assume: $T(n/2) \le c\,(n/2)\log(n/2)$

Show: $T(n) \le cn\log(n)$

Merge Sort Runtime: Inductive Proof

Assume: $T(n/2) \leq c\,(n/2)\log(n/2)$

Show: $T(n) \leq cn\log(n)$

$$2 \cdot T(\frac{n}{2}) + c_1 + c_2 n \leq cn\log(n)$$

Merge Sort Runtime: Inductive Proof

Assume: $T(n/2) \leq c\ (n/2) \log(n/2)$

Show: $T(n) \leq cn \log(n)$

$$2 \cdot T(\frac{n}{2}) + c_1 + c_2 n \leq cn \log(n)$$

By the assumption, and transitivity, we just need to show:

$$2c\frac{n}{2} \log\left(\frac{n}{2}\right) + c_1 + c_2 n \leq cn \log(n)$$

Merge Sort Runtime: Inductive Proof

Assume: $T(n/2) \leq c\,(n/2)\log(n/2)$
Show: $T(n) \leq cn\log(n)$

$$2 \cdot T(\frac{n}{2}) + c_1 + c_2 n \leq cn\log(n)$$

By the assumption, and transitivity, we just need to show:

$$2c\frac{n}{2}\log\left(\frac{n}{2}\right) + c_1 + c_2 n \leq cn\log(n)$$

$$cn\log(n) - cn\log(2) + c_1 + c_2 n \leq cn\log(n)$$

Merge Sort Runtime: Inductive Proof

Assume: $T(n/2) \leq c\,(n/2)\log(n/2)$

Show: $T(n) \leq cn\log(n)$

$$2 \cdot T(\frac{n}{2}) + c_1 + c_2 n \leq cn\log(n)$$

By the assumption, and transitivity, we just need to show:

$$2c\frac{n}{2}\log\left(\frac{n}{2}\right) + c_1 + c_2 n \leq cn\log(n)$$

$$cn\log(n) - cn\log(2) + c_1 + c_2 n \leq cn\log(n)$$

$$c_1 + c_2 n \leq cn\log(2)$$

Merge Sort Runtime: Inductive Proof

$$c_1 + c_2 n \leq cn \log(2)$$

Merge Sort Runtime: Inductive Proof

$$c_1 + c_2 n \leq cn \log(2)$$

$$\frac{c_1}{n \log(2)} + \frac{c_2}{\log(2)} \leq c$$

Merge Sort Runtime: Inductive Proof

$$c_1 + c_2 n \leq cn \log(2)$$

$$\frac{c_1}{n \log(2)} + \frac{c_2}{\log(2)} \leq c$$

Which is true for any

$$n_0 \geq \frac{c_1}{\log(2)} \quad \text{and} \quad c > \frac{c_2}{\log(2)} + 1$$

Next Time...

Quick Sort

Average Runtime